Cerebron

A Reconfigurable Architecture for Spatio-Temporal Sparse Spiking Neural Networks

Chen, Qinyu ; Gao, Chang; Fu, Yuxiang

DOI
10.1109/TVLSI.2022.3196839

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Very Large Scale Integration (VLSI) Systems

Cerebron: A Reconfigurable Architecture for Spatiotemporal Sparse Spiking Neural Networks

Qinyu Chen, *Member, IEEE*, Chang Gao, *Member, IEEE*, and Yuxiang Fu, *Member, IEEE*

***Abstract*— Spiking neural networks (SNNs) are promising alternatives to artificial neural networks (ANNs) since they are more realistic brain-inspired computing models. SNNs have sparse neuron firing over time, i.e., spatiotemporal sparsity; thus, they are helpful in enabling energy-efficient hardware inference. However, exploiting the spatiotemporal sparsity of SNNs in hardware leads to unpredictable and unbalanced workloads, degrading the energy efficiency. Compared to SNNs with simple fully connected structures, those extensive structures (e.g., standard convolutions, depthwise convolutions, and pointwise convolutions) can deal with more complicated tasks but lead to difficulties in hardware mapping. In this work, we propose a novel reconfigurable architecture, Cerebron, which can fully exploit the spatiotemporal sparsity in SNNs with maximized data reuse and propose optimization techniques to improve the efficiency and flexibility of the hardware. To achieve flexibility, the reconfigurable compute engine is compatible with a variety of spiking layers and supports inter-computing-unit (CU) and intra-CU reconfiguration. The compute engine can exploit data reuse and guarantee parallel data access when processing different convolutions to achieve memory efficiency. A two-step data sparsity exploitation method is introduced to leverage the sparsity of discrete spikes and reduce the computation time. Besides, an online channelwise workload scheduling strategy is designed to reduce the latency further. Cerebron is verified on image segmentation and classification tasks using a variety of state-of-the-art spiking network structures. Experimental results show that Cerebron has achieved at least 17.5× prediction energy reduction and 20× speedup compared with state-of-the-art field-programmable gate array (FPGA)-based accelerators.**

***Index Terms*— Field-programmable gate array (FPGA), spiking neural network (SNN), workload balancing.**

I. Introduction

OVER the past decade, the revolution of deep neural networks (DNNs) has led to the state-of-the-art performance on various tasks, such as image classification [1], semantic segmentation [2], and object detection [3]. However, DNNs are computational-intensive. They have high computational complexity and tremendous parameters, leading to a large memory footprint and power budget and are difficult to be deployed on resource-constrained platforms. DNN compression methods [4] and dedicated, efficient hardware accelerators [5], [6], [7], [8] were explored to deal with this problem. Network compression methods include data quantization [9], [10], sparsity exploration [11], [12], and compact model design [13], [14]. MobileNet [13], [15] as a typical representative of compact models, adopts depthwise separable convolution (DSC), which can achieve a comparable accuracy with much fewer multiply-and-accumulation (MAC) operations and parameters. To further improve throughput and energy efficiency, dedicated hardware accelerators for compact models [7], [8] were designed to explore computing parallelism and efficient memory organization.

Another way to enhance energy efficiency is to use spiking neural networks (SNNs). Compared to continuous-valued DNNs, SNNs adopt an event-driven computing mechanism (i.e., the membrane potentials of neurons are updated only when the input spikes arrive); thus, they have inherent spatiotemporal sparsity brought by discrete binary spikes [16] and can achieve high energy efficiency by replacing multibit MAC operations by additions. Existing works, such as IBM TrueNorth [17], Intel Loihi [18], and Tianjic [19], have shown that event-based SNNs can be efficiently implemented in custom hardware.

It is believed that the ultimate advantage of SNNs comes from their ability to fully exploit spatiotemporal event-based information [16]. Previous works have shown that SNNs can achieve competitive accuracy compared with nonspiking counterparts for some complicated image segmentation and image classification tasks [20], [21], [22], [23]. However, modern silicon implementations of SNNs still lag behind DNNs, mainly featuring lower throughput and higher energy per neuron [24]. The spatiotemporal event-based information processing paradigm leads to spatiotemporal sparsity in the networks. Exploiting spatiotemporal sparsity in hardware design usually leads to unpredictable and unbalanced workloads, and potentially irregular and redundant memory access, thus degrading efficiency. Besides, the inputs need to be received and processed across several time steps, which leads to repeated data accesses and longer processing time. Therefore, exploiting parallelism in SNNs is more challenging than DNNs. Current SNN accelerators mainly focus on accelerating traditional network

Manuscript received 25 January 2022; revised 1 June 2022 and 11 July 2022; accepted 2 August 2022. Date of publication 16 August 2022; date of current version 27 September 2022. This work was supported in part by the Science and Technology Commission of Shanghai Municipality under Grant 21DZ1100500 and in part by the Shanghai Frontiers Science Center Program under Grant 2021-2025 No. 20. *(Qinyu Chen and Chang Gao are co-first authors.) (Corresponding author: Qinyu Chen.)*

Qinyu Chen is with the Institute of Photonic Chips, University of Shanghai for Science and Technology, Shanghai 200093, China (e-mail: qinyu@usst.edu.cn).

Chang Gao is with the Department of Microelectronics, Delft University of Technology, 2628 CD Delft, The Netherlands.

Yuxiang Fu is with the School of Electronic Science and Technology, Nanjing University, Nanjing 210093, China.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3196839.

Digital Object Identifier 10.1109/TVLSI.2022.3196839

structures such as multilayer perception [25], [26], which are difficult to meet the requirements of increasingly complex applications in terms of performance and efficiency.

Therefore, we are motivated by these findings to design a reconfigurable architecture for SNNs, called Cerebron, targeting high flexibility and efficiency. Cerebron can further leverage flexibility to achieve higher efficiency. For example, Cerebron can use both event-driven characteristics (spatiotemporal sparsity exploitation) and compact model structures (DSC) to achieve higher efficiency. We also propose several optimizations to achieve high performance and low hardware overhead. The main contributions are:

1) To obtain flexibility, a reconfigurable compute engine with inter-computing-unit (CU) reconfiguration and intra-CU reconfiguration is proposed to support various spiking layers, covering operations in various hybrid-NNs. The compute engine is also designed to exploit data reuse and guarantee parallel data access when processing different types of convolution to achieve high memory efficiency.
2) To reduce the computation time, a two-step data sparsity exploration method is introduced to leverage the sparsity of discrete spikes. Besides, an online channelwise workload scheduling method is proposed to reduce the latency further.
3) The proposed design is implemented on a Xilinx XC7Z100 field-programmable gate array (FPGA) and verified by image segmentation and classification tasks. Results show at least 17.5× prediction energy reduction and 20× speedup achieved by Cerebron compared with state-of-the-art FPGA-based accelerators.

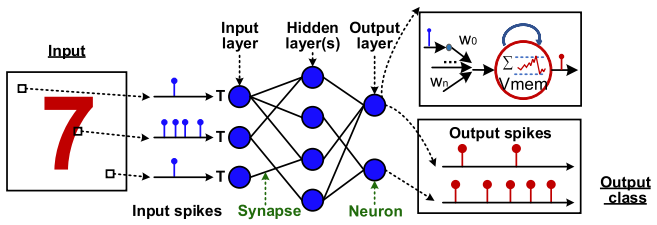


Fig. 1. SNN topology and spiking neural dynamics.

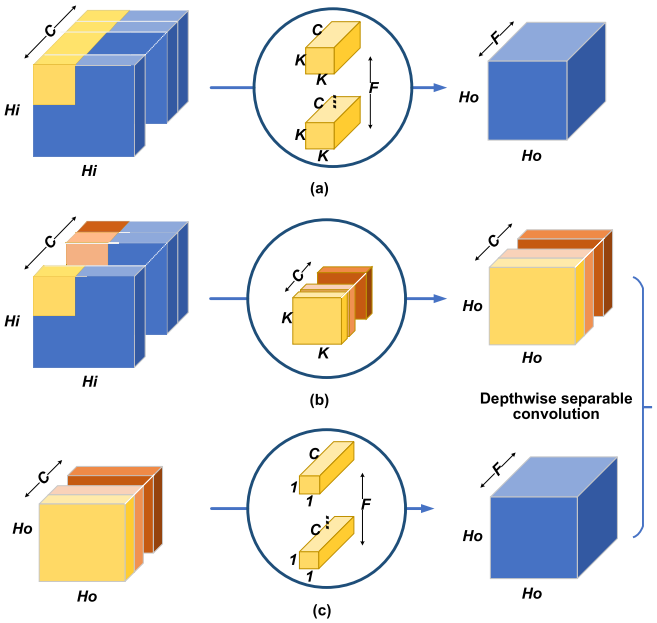


Fig. 2. Different kinds of convolutions.

II. Preliminaries and Motivations

A. Neuron Model for SNN

The integrated-and-fire (IF) neuron model is used in this work. Assume that at time step t, the IF neurons in layer l receive binary input spikes $\Theta^l(t)$, and the update mechanism of the temporary membrane potential of neurons is given by

$$V^l_{\text{temp}}(t+1) = V^l(t) + W^l\Theta^l(t) \tag{1}$$

where $V^l(t)$ is the membrane potential at time step t, $V^l_{\text{temp}}(t+1)$ is the intermediate variable from $V^l(t)$ to $V^l(t+1)$, V_{th} is the voltage threshold, and W^l is the synaptic weight, which can signify the connection between neurons.

If $V^l_{\text{temp}}(t+1)$ exceeds a predefined threshold V_{th}, an output spike $\Theta^l(t+1)$ will be produced. The membrane potential at $t+1$ would be updated by the reset-by-subtraction method [27]. The updating rules are described as

$$\Theta^{l+1}(t) = U\left(V^l_{\text{temp}}(t+1) - V_{\text{th}}\right) \tag{2}$$

where $U(x)$ denotes a unit step function. As shown in Fig. 1, SNNs are organized in cascaded layers and are executed over time steps with inputs encoded in spike trains. The spikes propagate through the network until reaching the output.

B. Depthwise Separable Convolutions

DSC [28] factorizes the standard convolution into a depthwise convolution and a pointwise convolution. Fig. 2 describes how standard, depthwise, and pointwise convolution work. In the standard convolution, the input feature map has to do a convolution with F filters with a size of $K \times K \times C$. The DSC comprises two steps. The first step is the depthwise convolution, where only 2-D convolution is performed on each input channel individually with $K \times K$ kernels. The second step is the pointwise convolution, which can be regarded as a standard convolution with 1×1 kernels. Compared to standard convolution, DSC considerably reduces the number of operations and parameters. One of the typical applications of DSC is the compact model MobileNet, which can be run much faster than traditional DNNs.

C. Spatiotemporal Sparsity and Event-Based Workloads in SNNs

In traditional DNNs, sparsity exists both in weights and feature maps. Some state-of-the-art pruning methods can significantly increase the weight sparsity in networks with comparable accuracy. Feature sparsity is the zeros existing in the feature maps and is mainly caused by the activation function (e.g., ReLU). The sparsity level is defined by the

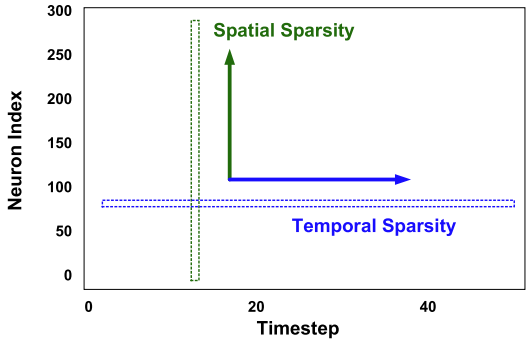


Fig. 3. Spike activity of a spiking convolution layer when running 28 × 28-16c5-32c3-8c1(branch1)-10c3(branch2)-fc network on MNIST dataset.

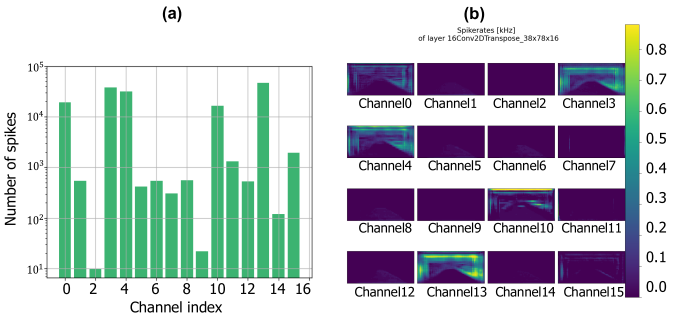


Fig. 4. Demonstration of the workload. (a) Spike summation of different channels. (b) Spike rate distribution of different channels in a certain CONV layer. The data are collected during segmenting a frame in a driving video, and the output is the segmentation mask for the road.

fraction of zeros in filters or features. Operations involved with zeros have no impact on the final results and thus can be skipped to reduce computational workloads. In DNNs, sparsity only exists in the spatial dimension.

In contrast, SNNs run over multiple time steps. The sparsity exists not only spatially across neurons but also temporally over time steps, i.e., spatiotemporal sparsity. In SNNs, weight sparsity has the same definition as that of DNNs. However, feature sparsity is not present in SNNs but replaced by the neuron state sparsity caused by discrete spikes. The neuron state is binary, where 1 denotes that a spike is produced and its fan-out connections are active, 0 denotes no spike produced, and its fan-out connections are inactive. As mentioned before, a spike is only produced when the accumulated membrane potential exceeds a predefined threshold. The spike rate of SNNs is defined as the fraction of neurons that produce a spike across all time steps, which directly affects the sparsity level. Fig. 3 shows the spike activity of a convolution layer when running on samples in the MNIST dataset. Most neurons do not fire at each time step, showing high-level spatiotemporal sparsity (∼93%).

SNNs have intrinsically event-driven workloads since the workload associated with a layer strongly relates to the spike rate of neurons and the number of active fan-out connections per neuron [24]. The dynamic, active connections between neurons introduce an unpredictable workload pattern. Fig. 4 shows the input neuron state maps of different channels. We found that the spike rate varies considerably among these input channels. The unbalanced spike rates among the input channels will correspondingly lead to unbalanced workloads. In summary, due to weight pruning and the randomness of input spikes, the sparsity is irregular and unpredictable, which is difficult for accelerators to leverage. Besides, the unbalanced workload caused by the sparsity will reduce the hardware efficiency.

III. Related Work

SNNs are intensive in computing and data access. A variety of SNN acceleration methods have been designed to improve the efficiency of SNN computation and can be categorized into three types. The first type is to deploy SNNs on commercial platforms such as GPUs [29], [30], [31]. However, GPUs can only achieve maximum efficiency when having a large amount of parallelizable computation and memory access, which is in contrast with the event-driven nature of SNNs. The second direction is to build specific hardware for SNNs [17], [18], [19], [24], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41]. In this direction, various SNN hardware implementations have recently been proposed, which can be divided into two categories in terms of topologies: general mesh and feedforward [42]. For the general mesh, existing large-scale neuromorphic hardware systems, such as IBM TrueNorth [17], Intel Loihi [18], and Tianjic [19], can support a mesh of neurons by routers and schedulers. The SNN networks are distributed among the neurocores, and each neurocore is responsible for storing a portion of the weights and computing that portion of the SNN topology. These accelerators are biologically plausible but usually need large area costs. Some accelerators (e.g., SIES [43], S2N2 [36], and Spinalflow [24]) choose the feedforward approach. Their neurocores are arranged in a cascaded fashion or configurable processing element (PE) array. The accelerators belonging to this kind require less area cost and achieve higher computing resources utilization. Our work Cerebron also falls into this category and is also largely complementary. The third category is to explore emerging devices or materials that can be adaptable to the event-driven properties of SNNs, e.g., optics [44], [45], memristors [46], [47], and spintronics [48], [49]. Using the property of novel devices or materials may potentially boost efficiency. For example, Feldmann *et al.* [45] designed a fully optical neuromorphic framework for implementing SNNs using phase-change materials, presenting a photonic neural network consisting of four neurons and 60 synapses. Singh *et al.* [49] built an ultralow-power architecture NEBULA for SNNs and artificial neural networks (ANNs) inference using a spintronics-based magnetic tunnel junction neuron model.

IV. Architecture Design and Data Processing

A. Architecture Overview

The top-level architecture of Cerebron is shown in Fig. 5. It is composed of a controller, on-chip buffers, an address generator, and a compute engine. The on-chip buffer caches neuron states, membrane potential (VMEM), and synaptic

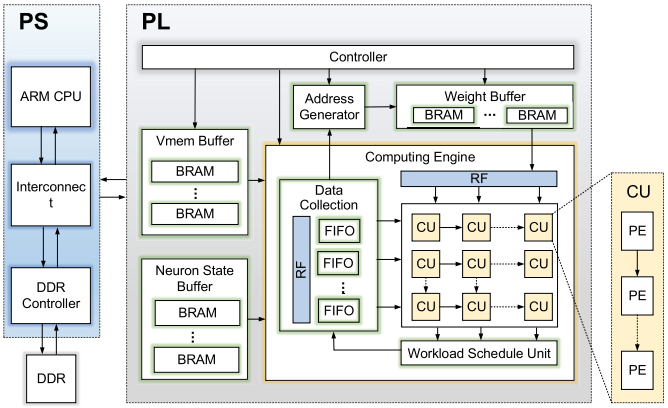


Fig. 5. Proposed system architecture.

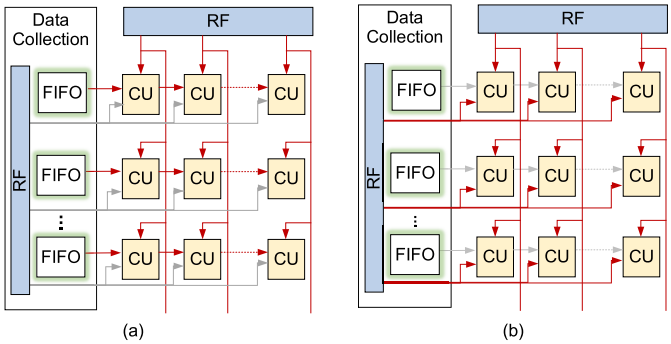


Fig. 6. Reconfigurable CU array with (a) systolic mode and (b) unicasting mode.

weights. A Xilinx Direct Memory Access (DMA) IP block controlled by the host is used to manage the I/O communications between the accelerator and the host. Input neuron states are streamed to the accelerator and stored in the neuron state buffer. The address generator produces the addresses of the corresponding weights and membrane potentials. The compute engine also consists of a CU array, a data collection unit, and a workload scheduling unit.

In the compute engine, the CU array has $M \times N$ CUs, and each CU is primarily composed of L PEs. The CU array is designed to be compatible with standard convolutions, depthwise convolutions, and pointwise convolutions to obtain versatility and flexibility. It is also optimized to support three kinds of data reuse: weight reuse, input neuron state reuse, and overlap reuse, which can vastly reduce memory access and eliminate the memory access conflict problem. The data collection unit contains multiple first input–first outputs (FIFOs) and register files to buffer data. The input neuron states and synaptic weights are loaded into the CU array to realize weight reuse and feature reuse. The overlap reuse is processed by transmitting overlaps in the data collection unit. The realization of data reuse by the data collection unit is further discussed in Section IV-D. The workload scheduling unit accumulates the spikes generated by CUs in each column to record the spike summation of each output channel, i.e., the spike summation of each input channel in the subsequent convolution. By comparing the spike summations of different input channels, the workload in the convolution can be scheduled channelwisely to obtain the balance.

B. Reconfigurable Compute Engine

1) Inter-CU Reconfiguration: The systolic array is a specialized network of homogeneous PEs designed to process massive parallel computations [50]. In a typical systolic array, PEs get their inputs from neighboring PEs without frequently accessing data from memory. This is why the systolic array can achieve high throughput with relatively low memory bandwidth. Due to this advantage, many previous works adopt systolic arrays with output stationary dataflow for accelerating DNNs [51], [52]. However, the naive systolic array with output stationary dataflow has some shortcomings. On the one hand, it is not suitable for processing all kinds of convolution operations. For example, from a data reuse perspective, the depthwise convolution cannot be processed in a naive systolic manner because each 2-D input feature map is only convolved with one filter; in other words, PEs cannot get inputs from their neighboring PEs. Only the PEs in the first column are busy, resulting in a considerable reduction of the utilization ratio and degrading the performance. On the other hand, the naive systolic array cannot fully exploit the parallelism in neural networks. Each PE undertakes a convolution computation, limiting the parallelism on sliding input windows' dimensions, such as channel dimension or vector dimension. Previous works [7], [8] designed configurable adder trees to support different convolutions and exploit the inherent parallelism in convolutions. The reconfigurable adder tree can be reconfigured to process x additions, each adding y input data simultaneously, making it compatible with the parallelism in various convolutions. However, this parallelism brought by adder trees might result in the difficulty of processing sparsity.

In this work, we design a reconfigurable CU array to support different convolutions and exploit the parallelism. Our computing structure combines the advantages of the systolic array and adder tree-based structure, exploiting higher parallelism and keeping flexible and sparsity friendly. The proposed reconfigurable CU array can be reconfigured to different modes by changing the connections from the data collection unit to each CU and between neighboring CUs. Fig. 6 describes the details of the two reconfigurable modes, including systolic mode and unicasting mode. In the systolic mode [Fig. 6(a)], the input neuron state maps are loaded from FIFOs in the data collection unit. They are sent to the CU array from the left input port and horizontally shifted to the CUs. The output neuron states and updated membrane potentials are shifted to the right edge of the array. Each CU column has a weight bus, which is connected to a group of register files. The weight bus broadcasts the weights to CUs in this column. In the unicasting mode [Fig. 6(b)], each CU has its own connection to the register files, and the input neuron state maps can be directly loaded from the register files in the data collection unit. In this way, depthwise convolution can be deployed on the CU array while keeping all the CUs busy.

2) Intra-CU Reconfiguration: Within a CU, the PE can be reconfigured to support both different functions

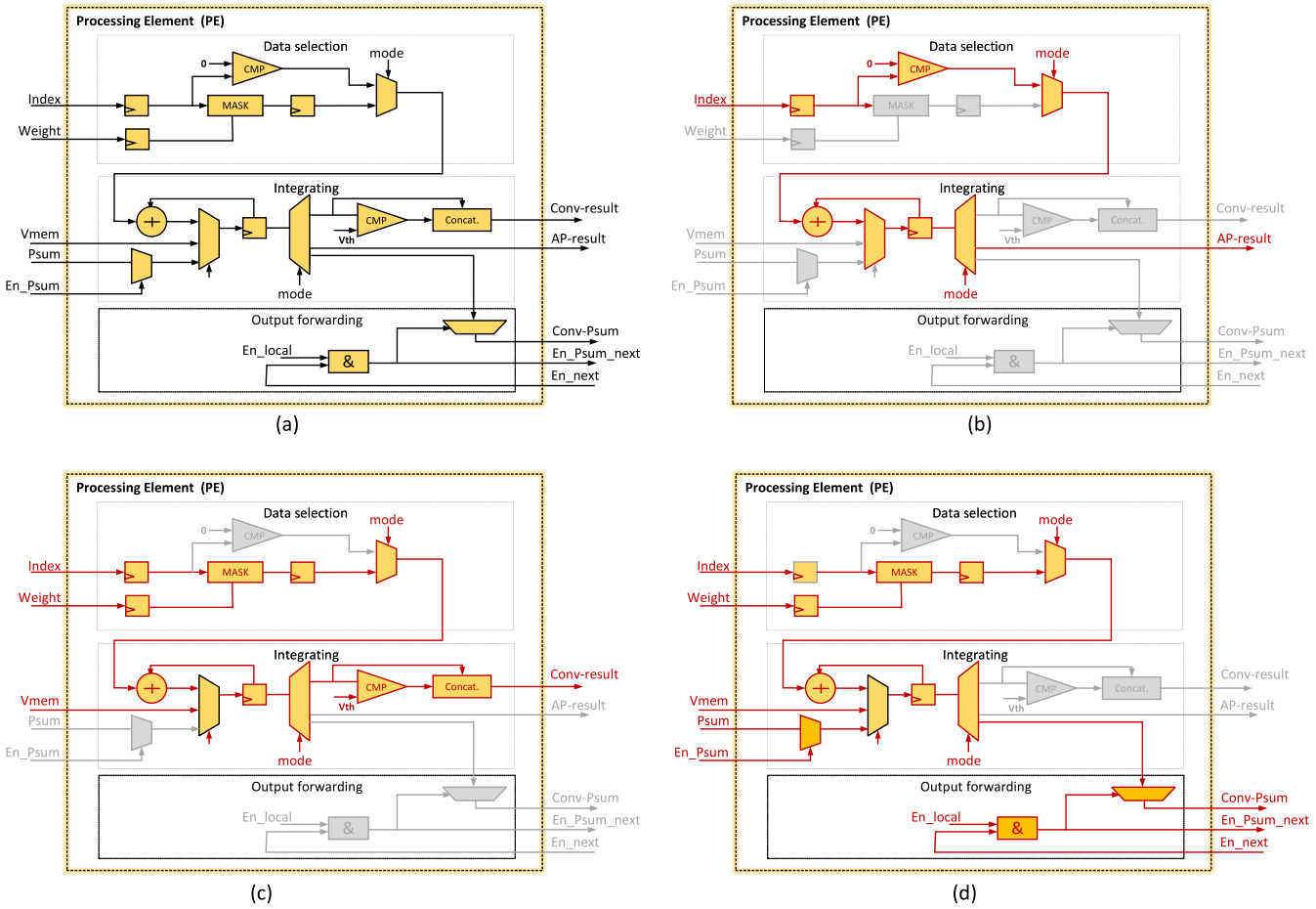


Fig. 7. Reconfigurable PE: (a) PE architecture, (b) datapath under average-pooling mode, (c) datapath under stand-alone convolution mode, and (d) datapath under cascade convolution mode.

according to the configuration word mode, which includes the average-pooling mode [Fig. 7(b)], stand-alone convolution mode [Fig. 7(c)], and cascade convolution mode [Fig. 7(d)]. Under the average-pooling mode, PEs can execute the average-pooling function independently. Under the stand-alone convolution mode, PEs can undertake the convolution independently, mainly used in the depthwise convolution. Under the cascade convolution mode, PEs are configured to work with other PEs in the same CU to process the convolution, adopted in the standard and pointwise convolutions. The cascade convolution mode obtains the parallelism on the channel dimension.

As shown in Fig. 7(a), the PE mainly consists of an accumulator and several comparators. Most of these units are reused in different modes. Therefore, the unused PE area is relatively small. The PE structure can be decomposed into three modules: data selecting, integrating, and output forwarding. The data selecting module selects the nonzero data to support zero skipping, thus reducing operations and power consumption. The integrating module performs the accumulation. In the stand-alone and cascade convolution modes, the inputs of the accumulation are the synaptic weights from the aligned pairs, while in the average-pooling mode, the inputs are the input neuron states from the aligned pairs. The aligned pair denotes the pair where the input neuron states and weights are both nonzero data. The output forwarding module is designed for the cascade convolution mode. Since the sparsity is irregular, the workload of each PE might not be equal, and the output forwarding module works as an output regulator to arrange the transmission of the partial sums between PEs. Within a CU, the output enable signal from the next PE En_next will be transferred to the current PE and works together with the local output result enable signal En_local to determine when the output enable signal En_Psum_next of the current PE should be forwarded to the next PE.

C. Two-Step Data Sparsity Exploitation

The nonzero aligned data pairs should be extracted before accumulating to exploit both input neuron state sparsity and weight sparsity. The operations involved with zero operands should be skipped to save energy and processing time.

Some works first compressed the data using popular sparse encoding formats, e.g., CSC [5], ECOO [50], and then extracted and compared the indices of nonzero features and weights to get the nonzero aligned data pairs. However, SNNs are not suitable for encoding binary neuron states since the indices of nonzero data lead to high memory overhead since

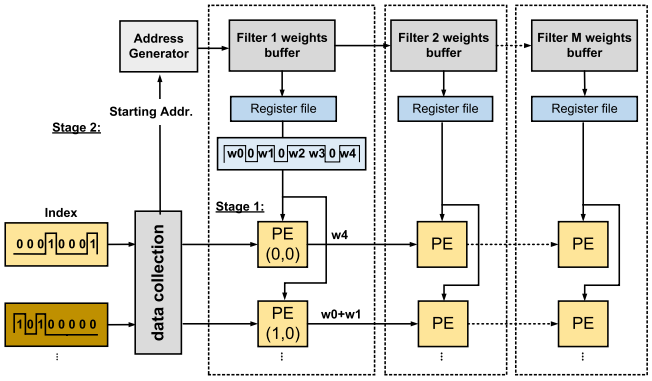


Fig. 8. Proposed two-step data sparsity exploitation.

they require multiple bits [36]. The previous work [35] adopted a one-step sparsity exploitation method to skip the input neuron state sparsity. The addresses of weights that correspond to the nonzero neuron state are calculated, and the weights are read from the buffers in one step. However, this requires several copies of the weights; otherwise, it will lead to memory access conflict. PEs in the same column receive different rows of input neuron states to reuse filters. However, the various sparsity existing in these neuron state rows leads to different weight addresses, thus leading to conflicts.

In this work, a two-step data sparsity exploitation method is proposed to overcome the limitations. As shown in Fig. 8, we take a pointwise convolution example to illustrate how the two-stage data sparsity exploitation method works. In the first stage, two 1-D input neuron state vectors with length C are formed to two C-bit index signals "00010001" and "10100000" and loaded to the data collection unit.

The two index signals will be dispatched to PE (0, 0) and PE (1, 0) in the CU array. Meanwhile, in the second stage, the starting address of each input spike vector is sent to the address generator to calculate the corresponding weight address. In this way, the weight vectors "$w_0 0 w_1 0 w_2 w_3 0 w_4$" are read from the weight buffer and transmitted to the register files in the CU array. The index signals and the weight vector are adopted as mutual masks to determine the weights that have the exact location of aligned pairs, where both the index and weight are nonzero data. Within one clock cycle, the nonzero data will be filtered out, saved to the register file, and waiting to be accumulated. In the PE (0, 0), only the synaptic weight w_4 is filtered out to accumulate; in the PE (1, 0), the synaptic weights w_0 and w_1 are filtered out.

In such an approach, memory access conflicts are avoided when reusing weights. Furthermore, the CU array mainly involves accessing small register files instead of frequently accessing large buffers [static random access memory (SRAM)]. The energy efficiency could be improved accordingly.

D. Data Reuse Supporting Different Convolutions

Different convolutions usually involve different accumulation patterns, which brings an obstacle to the hardware design, especially in terms of data reuse. Fig. 9 describes how these convolutions achieve data reuse on the compute engine.

1) Data Reuse in the Standard Convolution: The standard convolution can be exploited in weight reuse, feature reuse, and overlap reuse. The details of data reuse in the standard convolution are shown in Fig. 9(a). The weight reuse and input neuron state map reuse are obtained by broadcasting the same weights to CUs in the same CU column and transferring the same inputs to CUs in the same row. In the standard convolution, the input neuron state maps required for the three convolution operations SM1, SM2, and SM3 are overlapped with each other. The overlapped part of SM1 and SM2 is called 3-D vertical overlap, and the overlapped part of SM2 and SM3 is denoted as 3-D horizontal overlap. If it is a naive design, the overlapped parts of the input neuron state maps would be stored as several copies, resulting in a waste of memory; also, repeated memory accesses are introduced, leading to unnecessary energy consumption.

In this design, all input neuron state map elements need to be read from the buffer only once. The vertical-FIFOs (v-FIFOs) and horizontal-FIFOs (h-FIFOs) in the data collection units are used to store the vertical overlap and horizontal overlap, respectively, temporarily. We take the first convolution involved with SM1 as an example. C_{ij} denotes the input vector along the channel dimension. In the first period, C_{11}, C_{21}, and C_{31} are loaded from buffers and sent to the CU array to initiate the three convolutions. Meanwhile, copies of C_{11}, C_{21}, and C_{31} are streamed into v-FIFOs. In the second period, the copy of C_{21} in v-FIFO2 is sent to the first CU row. In the third period, the copy of C_{31} in v-FIFO3 is sent to the first CU row. Then, in the fourth period, C_{12}, C_{22}, and C_{32} are loaded from the buffers while holding two copies in v-FIFOs and h-FIFOs. The copies of C_{22} and C_{32} in v-FIFOs are sent to the first CU row in the following two periods. In the next three periods, C_{13}, C_{23}, and C_{33} are operated in the same manner. The copies stored in the h-FIFOs provide inputs for the neighboring right-hand convolution involved with SM3.

2) Data Reuse in the Pointwise Convolution: The pointwise convolution can be regarded as a particular case of the standard convolution, i.e., the standard convolution with 1×1 kernel size. In this case, overlapped input neuron state maps do not exist when the filters slide over the inputs. They are only related to weight reuse and feature reuse. Fig. 9(b) shows the details of data reuse in the pointwise convolution. The weight reuse and input neuron state map reuse in the pointwise convolution can be achieved using the same approach used in standard convolutions. The weights of the filters are broadcasted to CUs in the same column, and the inputs are transferred to CUs in the same row.

3) Data Reuse in the Depthwise Convolution: The depthwise convolutions are composed of multiple individual 2-D standard convolutions. Each 2-D convolution only has one filter, indicating that the input neuron state reuse does not exist. In this case, only the weight reuse and overlap reuse are exploited. The input neuron state maps cannot be fetched from the neighboring CUs because the input neuron state maps are not reused in the depthwise convolution. Therefore, the CU array is reconfigured to $unicasting$ mode to increase the parallelism. As shown in Fig. 9(c), each CU in the array is

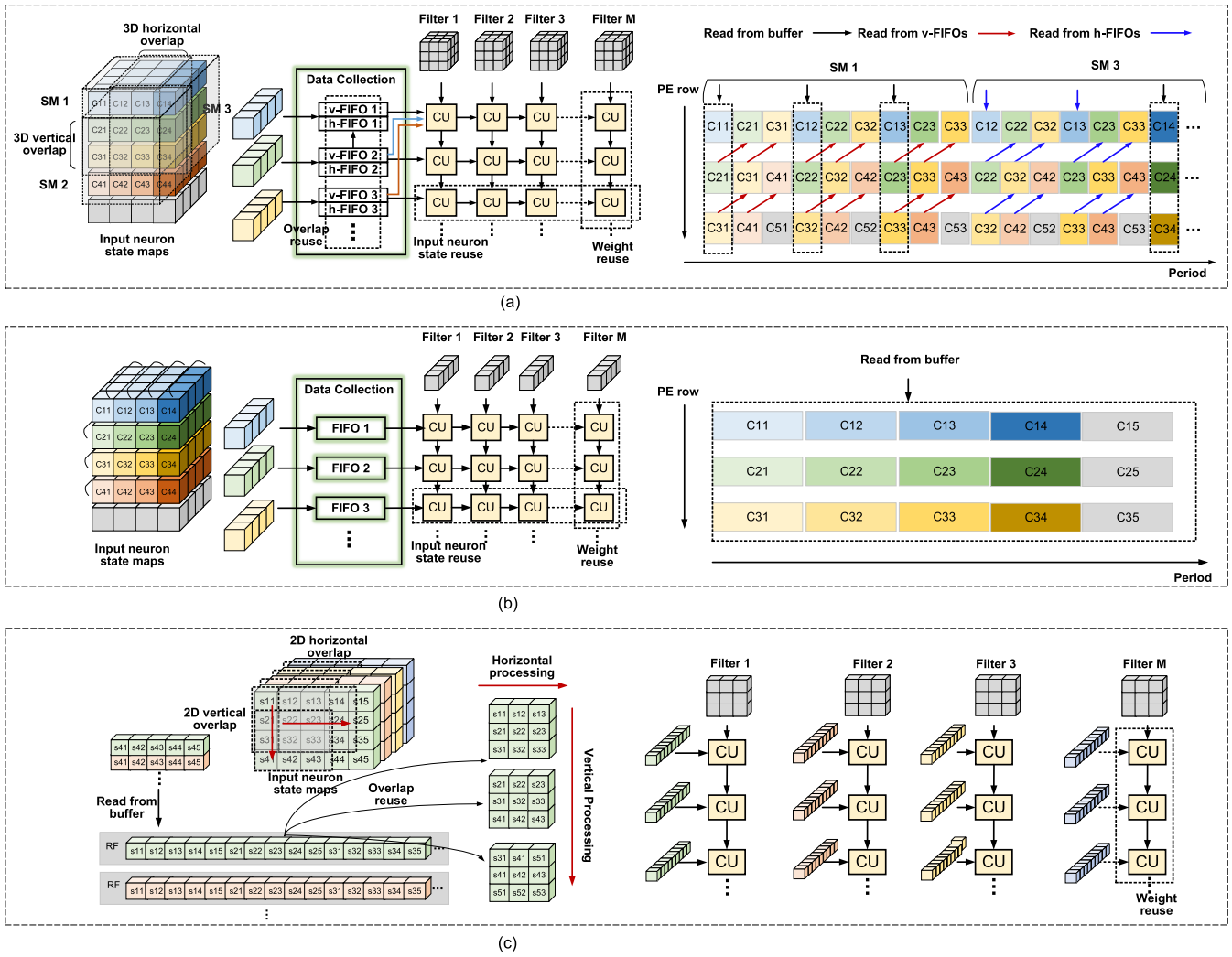


Fig. 9. Data reuse supporting different convolutions: (a) standard convolutions, (b) pointwise convolutions, and (c) depthwise convolutions.

designed to have an extra interface to the outside, connected to a group of register files.

Each CU column performs one 2-D convolution independently. The weight reuse is obtained by broadcasting the weights to CUs in the same CU column. The overlap here is divided into vertical overlap and horizontal overlap. We take a 2-D convolution as an example. The first multiple rows of the input neuron state map are read from the buffer directly, and then, they are reshaped into a vector and stored in the register file. The inputs required for computing an output point can be directly fetched from the register files. In this case, the horizontal overlap reuse can be realized. As for the vertical overlap reuse, when the input neuron states in the first row are no longer involved in the convolution, they will be replaced by the input neuron states in the next row while keeping other intermediate rows in the register files. Each time only the useless rows will be replaced.

E. Exploiting Workload Balance in Spiking Convolutions: Online Channelwise Workload Scheduling

Fig. 4 shows the considerably unbalanced sparsity among the input neuron state maps of different channels. This unbalance will reduce the throughput when processing convolutions. In the standard convolution, PEs undertake the computations over the input channels. Within a CU, the PE, which computes with the most sparse inputs, first finishes its task, whereas the PE processing with the most nonzero inputs becomes the bottleneck of the hardware throughput. Similarly, in the depthwise convolutions, each CU column performs one slice of depthwise convolution (i.e., 2-D convolution) individually. Since the input sparsity of each 2-D convolution is different, the CU column, which computes the 2-D convolution with the most sparse inputs, first finishes its task, whereas the CU column processing 2-D convolution with the most nonzero inputs becomes the foot dragger.

We propose an online workload scheduling method targeting spiking convolutions (seen in Algorithm 1) to deal with this problem. The workload is scheduled with the minimum granularity of the 2-D convolution (i.e., a separable slice of depthwise convolution or partial channel convolution of standard convolutions). The scheduling task is mainly undertaken by the workload scheduling unit, primarily composed of a workload accumulator, a serial full comparison sorting module, and a scheduling table.

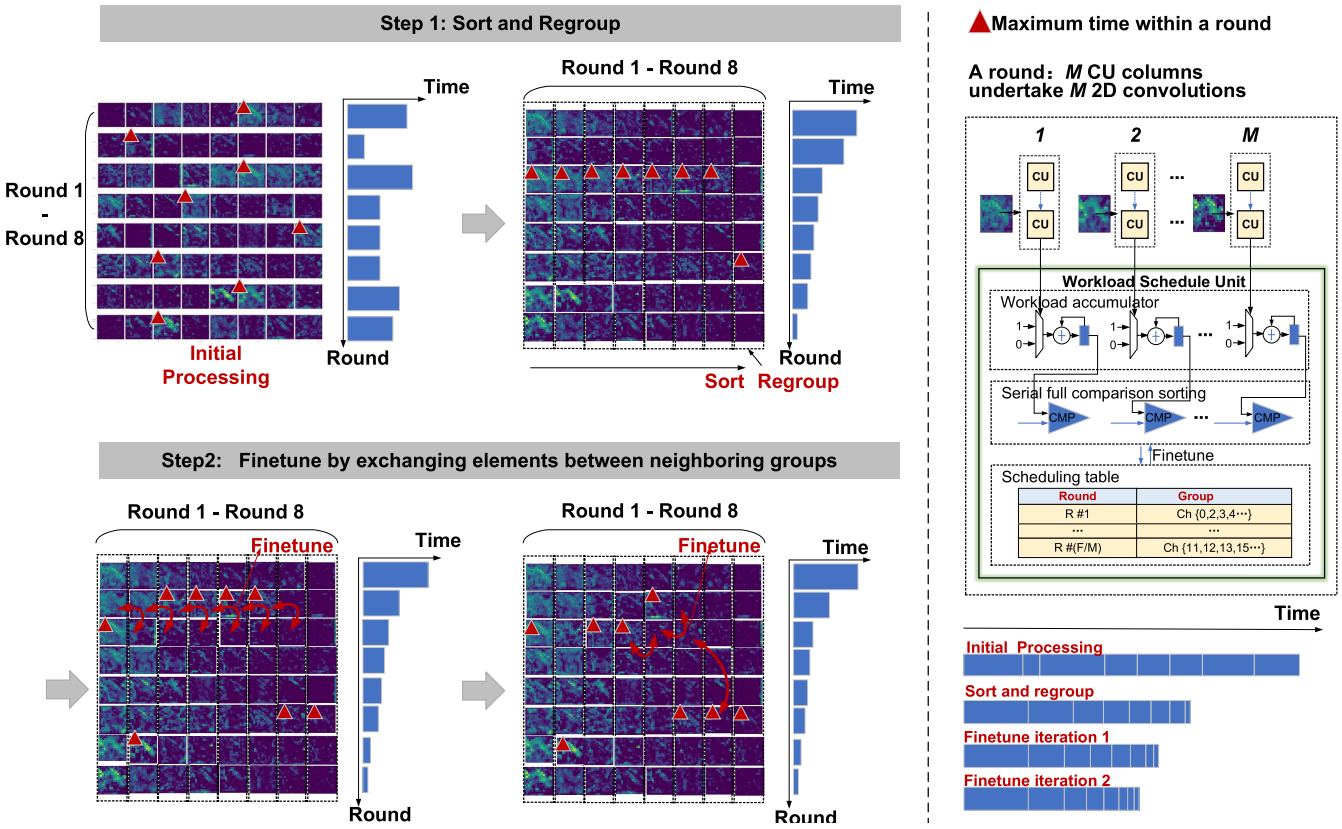


Fig. 10. Demonstration of the proposed workload scheduling method using a simple example.

The preparation work is to get the spike number of the input neuron state map in each 2-D convolution; in other words, we need to accumulate the spikes generated by the neurons in each output map in the previous convolution. The output generated by each CU will be transferred sequentially to the workload accumulator, each positive signal indicates an active connection, and makes the workload number add by 1. The first step is sorting and regrouping. Every round, we will get a list containing M output spike summations, and then, M values in the list will be sorted in order. Here, we adopt the serial full comparison sorting method, and the latency is about $2M$ cycles each round. The latency of all rounds except the last one is hidden by the computations in CUs. The last $2M$-cycle delay is negligible as it takes more than thousands of cycles to finish the convolution. Now, we get F/M sorted lists, each containing M elements. Then, the value in the same position of each list will be put together to regroup M new lists; each new list has F/M elements. The sorting and regrouping results will be stored in the scheduling table. The second step is fine-tuning. The maximum element in each new group will be selected and compared with the maximum element of the previous group using comparators in the serial full comparison module. If it is larger, these two elements will swap places. The fine-tuning results will be saved back to the scheduling table. The third step is adjusting. To guarantee the parallelism, we need to keep the element in the list equal to Q. The value Q is the number of PEs within a CU when processing standard convolutions, whereas the number of CU columns is in depthwise convolutions. Q is compared with F/M to determine whether the lists need to be split or concatenated with others.

Fig. 10 shows the procedure of the proposed workload scheduling method working on depthwise convolution. Assume that the CU array has eight CU columns ($M = 8$), and there is a pointwise convolution with 64 output neuron state maps ($F = 64$). These 64 output neuron state maps as the inputs of 64 2-D convolutions in the subsequent depthwise convolution need to be scheduled to achieve the workload balance. The input neuron state map with most spikes within a round dominates the processing time (denoted as the red triangle). The scheduling method can gather the 2-D convolution with similar levels of spike summation together into one processing round, thus reducing the workload unbalance. It is observed that the processing time is gradually reduced with the scheduling process.

V. Experimental Results

A. Experimental Setup

Methods for direct training of SNNs have made tremendous progress recently but are often computationally expensive and challenging to scale up to deeper networks. Conversion of DNNs to SNNs is a more straightforward way to obtain an SNN with equivalent accuracy to the DNN. This method transfers the pretrained ANN parameters to a network of the same topology as ANN but uses spiking neurons. In this work,

Algorithm 1 Channelwise Workload Scheduling Mechanism

Require:
F, the number of 2D convolutions in a convolution layer;
M, the number of CU columns;
S, the spike number of each 2D input neuron state map;
T, the iteration number.
Ensure:
G, the balanced task group.
// Step 1: Sorting and Regrouping.
for $i = 1; i \leq F/M; i++$ **do**
  C_i = Sort(S_i).
end for
for $i = 1; i \leq M; i++$ **do**
  for $j = 1; j \leq F/M; j++$ **do**
    Add the element C_{ij} to L_i;
  end for
end for
// Step 2: Finetuning
for $iter = 1; iter \leq T; iter++$ **do**
  for $i = 1; i \leq F/M; i++$ **do**
    if $Max(L_i) \geq Max(L_{i+1})$ **then**
      Swap places.
    end if
  end for
end for
// Step 3: Adjusting
if $F/M \geq G$ **then**
  Split each list L into F/MG sublists .
else
  Concatenate every F/MG list L into a list.
end if

for converting the DNNs to the spiking version, we use an open-source SNNtoolbox,[1] which implements the conversion methods described in [27].

We evaluate the effectiveness of the proposed design using two kinds of tasks, including image classification and image segmentation. When testing image classification tasks, we use two representative models of the spiking version on the MNIST and CIFAR-10 datasets. For the MNIST dataset, an in-house ConvNet model with architecture 28 × 28-16C3-32C3-16C3-10 is used for evaluation. For the CIFAR-10 dataset, we select a variant of MobileNet: a thinner MobileNet. The thinner MobileNet includes one standard convolution layer as the first layer, followed by eight DSCs, an average-pooling layer, and a fully connected layer. The detailed architecture is shown in Table I. Since the input size of CIFAR-10 is smaller than ImageNet, we change the stride of the first three depthwise convolutions from 2 to 1 to guarantee the resolution. When testing image segmentation tasks, we study the task of detecting lanes in driving videos. We use a segmentation network (referred to as SegNet hereinafter) with 160 × 80 × 3-8C3-16C3-32C3-32C3-16C3-1C3-160 × 80 × 1 structure

TABLE I
ARCHITECTURE OF THINNER MOBILENET ON CIFAR-10

Layer	Type/Stride	Filter Shape	Output Size
1	Conv / s1	3 × 3 × 3 × 16	32 × 32 × 16
2	Conv dw/ s1	3 × 3 × 16 dw	32 × 32 × 16
3	Conv / s1	1 × 1 × 16 × 32	32 × 32 × 32
4	Conv dw/ s1	3 × 3 × 32 dw	32 × 32 × 32
5	Conv / s1	1 × 1 × 32 × 64	32 × 32 × 64
6	Conv dw/ s1	3 × 3 × 64 dw	32 × 32 × 64
7	Conv / s1	1 × 1 × 64 × 64	32 × 32 × 64
8	Conv dw/ s1	3 × 3 × 64 dw	32 × 32 × 64
9	Conv / s1	1 × 1 × 64 × 128	32 × 32 × 128
10	Conv dw/ s1	3 × 3 × 128 dw	32 × 32 × 128
11	Conv / s1	1 × 1 × 128 × 128	32 × 32 × 128
12	Conv dw/ s2	3 × 3 × 128 dw	16 × 16 × 128
13	Conv / s1	1 × 1 × 128 × 256	16 × 16 × 256
14	Conv dw/ s2	3 × 3 × 256 dw	8 × 8 × 256
15	Conv / s1	1 × 1 × 256 × 512	8 × 8 × 512
16	Conv dw/ s1	3 × 3 × 512 dw	8 × 8 × 512
17	Conv / s1	1 × 1 × 512 × 512	8 × 8 × 512
18	AvgPool / s1	8 × 8	1 × 1 × 512
19	FC	512×10	10

to realize the end-to-end pixelwise prediction. The model is tested on the dataset from the MLND-Capstone project.[2]

We implement our design on a Xilinx Zynq XC7Z100 FPGA running at 200 MHz. The host program is responsible for sending synaptic weights and input neuron state maps into the programmable logic (PL) and collecting the final results.

B. Design Space Exploration in Workload Balance

The theoretical peak throughput of the accelerator on the PL part is given as

$$\text{Throughput}_{\text{peak}} = f * P \tag{3}$$

where f denotes the clock frequency of PL and P is the total number of PEs in this design. In this accelerator, $P = M \times N \times L$, where M denotes the number of CU columns, N is the number of CU rows, and L is the number of PEs within a CU. Theoretically, the peak hardware throughput is proportional to the number of PEs; however, the actual throughput is affected by the workload imbalance brought by the inherent dynamic spatiotemporal sparsity of SNNs.

As we analyzed in Section III-E, in depthwise convolution and standard convolution, the slices of 2-D convolutions have different nonzero values in the input neuron state maps, leading to unbalanced workloads. To alleviate the unbalance, we propose an online channelwise workload scheduling method. To quantify the balance effect, we define the balance ratio (BR) of the PEs adopted in [53]

$$\text{BR} = \frac{\sum_{t=1}^{T} \text{WL}_{t,\text{mean}}}{\sum_{t=1}^{T} \text{WL}_{t,\text{max}}} \tag{4}$$

$$\text{WL}_{t,\text{mean}} = \sum_{m=1}^{M} \text{WL}_t^m \tag{5}$$

$$\text{WL}_{t,\text{max}} = \max\left(\text{WL}_t^1, \text{WL}_t^2, \ldots, \text{WL}_t^M\right) \tag{6}$$

where $\text{WL}_{t,\text{mean}}$ and $\text{WL}_{t,\text{max}}$ are, respectively, the mean and max workload of PE array at time step t. The BR is obtained

[1] https://snntoolbox.readthedocs.io/

[2] https://github.com/mvirgo/MLND-Capstone/

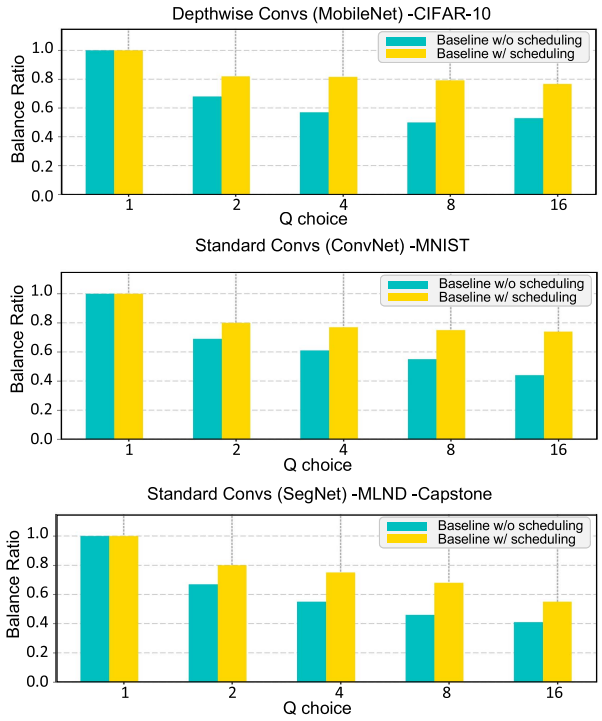


Fig. 11. Balance ratio of the accelerator versus the choice of Q in depthwise convolutions and standard convolutions.

by running over T time steps. The performance is dominated by the max workload $\mathrm{WL}_{t,\max}$. We can achieve better performance when the max workload is closer to the mean workload.

To evaluate the influence of the choice of Q on the workload balance, we conduct the experiments on depthwise convolution from MobileNet and standard convolutions from SegNet separately. As mentioned before, in depthwise convolution, Q denotes the number of PEs in a CU. Fig. 11(a) shows the BR across depthwise convolution layers of the spiking MobileNet with and without the online channelwise workload scheduling method and demonstrates how the BR changes with different choices of Q. It was observed that the workload is more balanced ($\sim$1.6$\times$) when the scheduling method is applied. Besides, the workload is naturally more imbalanced when the value of Q is increasing. As the precondition to increasing the hardware parallelism, we considered setting the Q value between 4 and 8, where the BR value is between 0.79 and 0.82.

In the standard convolution, Q denotes the number of CU columns. Fig. 11(b) and (c) shows the BR across standard convolution layers with and without workload scheduling method and demonstrate how the BR changes with different choices of Q, under two cases: ConvNet model for MNIST dataset and SegNet model for MLND-Capstone dataset. In the first case [Fig. 11(b)], the workload is more balanced ($\sim$1.3$\times$) when the workload scheduling method is applied. In the second case [Fig. 11(c)], the workload is more balanced ($\sim$1.7$\times$) when the workload scheduling method is applied. Similar to before, the workload is naturally more imbalanced when the value of Q increases. As the precondition to increasing the hardware parallelism, we considered setting the Q value

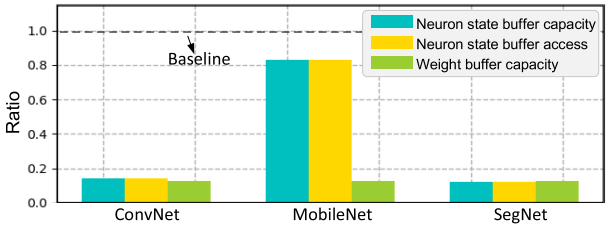


Fig. 12. Reduction on buffer capacity and access.

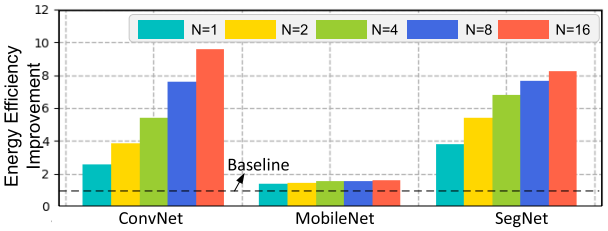


Fig. 13. Energy efficiency improvement of Cerebron with different sizes of CU rows.

between 4 and 8, where the BR value is between 0.68 and 0.75. Therefore, we set the number of CU columns as 8 and PEs within a CU as 4.

C. Memory Efficiency

Memory efficiency is mainly featured by evaluating the reduction of required buffer accessing and buffer capacity. In this design, it is mainly obtained by the overlap reuse and two-step sparsity exploitation. Fig. 12 shows the average reduction ratio in different models. From the overlap reuse perspective, nearly 9$\times$ reduction of neuron state buffer accessing and capacity is achieved on the ConvNet and SegNet since a large number of overlapped maps have existed. However, the reduction in MobileNet is relatively smaller because the pointwise convolution as the major component of MobileNet is involved with 1 $\times$ 1 kernels, which has no overlap. From the two-step sparsity exploitation perspective, it solves the weight access conflict problem and avoids using several copies of the weight buffer, thus reducing 8$\times$ of the buffer capacity.

D. Energy Efficiency and Performance

In this section, we evaluate the energy efficiency improvement of this design. Without considering off-chip data access, Fig. 13 shows that Cerebron can achieve up to 9.6$\times$ energy consumption reduction over various models compared to the accelerator without optimization. On one hand, numerous additions have been skipped, which reduces the energy consumption largely. On the other hand, the energy consumption caused by buffer access is significantly reduced. The overhead introduced by FIFOs or register files is much less than the above two aspects, which takes up less than 5% of the power. Experimental results also show that Cerebron with a larger number of CU rows (N) could obtain a higher improvement in energy efficiency. Considering that the input size of the latter layers in the network is small, we set the number of

TABLE II
COMPARISON WITH PREVIOUS WORKS

Metrics	TCAS-I'21[37]	JCST'20 [43]	ICCAD'20[38]	ASSCC'19[39]	JSA'17[40]	This work
FPGA Platform	VC707	XCVU440	XCZU9EG	XC7VX690T	Spartan-6	XC7Z100
Task	Classifi.	Classifi.	Classifi.	Classifi.	Classifi.	Classifi./Seg.
Dataset	MNIST	MNIST/CIFAR-10	MNIST	MNIST	MNIST/EEG	MNIST/CIFAR-10/MLND-Capstone
Input size	784	784/1024	784	784	784/6	784/1024/12800
Model	MLP[1]	ConvNet[2]/VGG-16	MLP/CNN[3]	MLP[4]	MLP[5]	ConvNet/MobileNet/SegNet
Accuracy	92.93%	99.16%/91.46%	98.96%/99.42%	98%	93.8%/92.7%	99.40%/91.90%/97.30%
Clock Frequency (MHz)	100	200	125	-	25	200
on-chip Power (W)	1.6	-	4.5	0.7	-	1.4
Predition Energy (mJ/frame)	5.04	-	2.34/33.84	0.7	-	0.04/14.88/1.12
Computation Time(ms/image)	3.15	-	0.52/7.69	1.09	160	0.026/10.63/0.80
Frame Per Second (KFPS)	0.32	-	1.92/0.13	0.91	0.006	38.5/0.09/1.25
Effect. Throughput (GSOp/s)	-	1.5625 (TOp/s)	-	0.73	-	40.1/44.2/45.0
Efficiency (GSOp/s/W)	-	-	-	0.95	-	28.6/31.6/32.1

[1] Classification network with 784-200-100-10 structure.
[2] Classification network with 28x28-12c5-mp-64c5-mp-10 structure.
[3] Classification network with 784-500-500-10 and 28x28-32C3-P2-32C3-P2-256-10 structure.
[4] Classification network with 784-512-384-10 structure.
[5] Classification network with 784-500-500-10 structure on MNIST dataset and another classification network with 6-50-100-2 structure on the dataset about the EEG signals for motor imagery.

CU rows as 8. In summary, the scale of Cerebron is set to $8 \times 8 \times 4$.

An essential feature of Cerebron is its ability to handle the dynamic sparsity in the temporal–spatial domain. We adopt the two-step sparsity exploitation method to eliminate the sparsity and an online channelwise workload scheduling strategy to deal with the unbalance workload introduced by the dynamic sparsity. Through analysis, the baseline is the 50-GOp/s theoretical peak throughput obtained by (4). When the two-step sparsity exploitation method was applied alone, the accelerator can achieve 0.44-, 0.48-, and 0.45-TOp/s effective throughput on ConvNet, MobileNet, and SegNet models, respectively. After the online channelwise workload scheduling method is applied, further speedup is achieved, and the accelerator can achieve 0.65, 0.52, and 0.60 TOp/s. Overall, by combining sparsity exploitation and workload scheduling, the accelerator achieved up to 13× speedup compared to the accelerator without optimization. For better illustrating Cerebron's ability to handle temporal SNNs, the effective throughput across different time steps is also tested. Taking the spiking ConvNet as an example, as shown in Fig. 14, the effective throughput changes with the time steps due to the varying active connections. Another evaluation method is based on the synaptic operation (SOP) [25]. Each SOP delivers a spike through an individual synapse. After optimizations, the throughput can reach 40.1, 44.2, and 45.0 GSOp/s on ConvNet, MobileNet, and SegNet models, respectively.

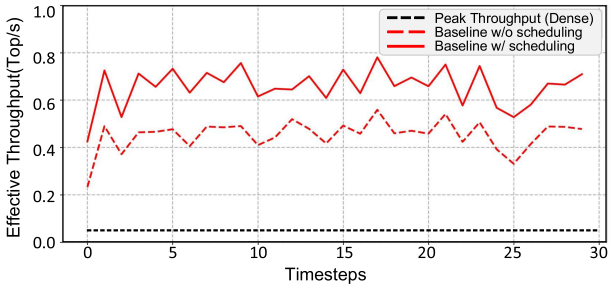


Fig. 14. Effective throughput across time steps when processing the Spiking CovNet.

E. Comparison With Previous Works and Discussion

Table II compares the performance of Cerebron with previous state-of-the-art SNN accelerators, including information such as tasks, datasets, computation time, energy consumption, and throughput. As can be seen from the table, Cerebron has maximum flexibility, which can support a variety of emerging network structures and model sizes. It has the lowest computation time (0.026 ms/image) and prediction energy (0.04 mJ/image) while achieving state-of-the-art accuracy classification accuracy (99.40%) for the MNIST classification task among these designs. When processing larger scale networks, it can achieve 10.63 ms/image, 14.88 mJ/image, and 91.90% accuracy for MobileNet on the CIFAR-10 dataset, and 0.80 ms/image, 1.12 mJ/image, and 97.30% accuracy for SegNet on the MLND-Capstone dataset. Results show that the implementation achieves at least 17.5× prediction energy reduction and 20× speedup compared with those state-of-the-art FPGA-based accelerators. The resource utilization of the XC7Z100 FPGA is summarized in Table III and the power breakdown is shown in Fig. 15.

The neuromorphic processors, such as TrueNorth and Loihi, exploited sparsity at the expense of area cost. Since they are biologically inspired designs, each neuron has occupied a fixed resource of the chip without reuse, and large areas of resources will be in the idle state. In our work Cerebron, resources can be reused by neurons and well-arranged to achieve both energy

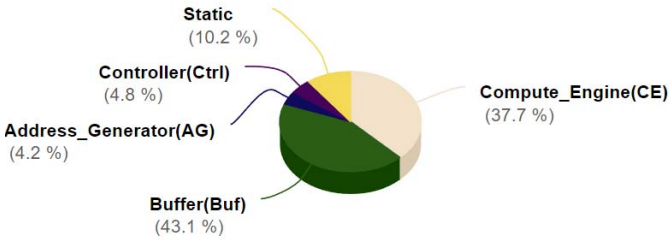


Fig. 15. Breakdown of on-chip power consumption.

TABLE III
XC7Z100 FPGA RESOURCE UTILIZATION OF CEREBRON

Metrics	LUT	FF	DSP	BRAM
Avaliable	277400	554800	2020	755
Used	85926 (75.2%@CE, 8.5%@AG 7.9%Ctrl, 8.4%@Buf)	70544 (85.3%@CE, 4.3%@AG 4.5%Ctrl, 5.9%@Buf)	0	283 (100%@Buf)
Percentage	30.97%	12.71%	0%	37.48%

efficiency and low cost by sparsity exploitation and afterward workload balance and scheduling. Besides, compared to other SNN accelerators using feedforward topology, our design is more sparse-friendly, and it exploits the spatiotemporal sparsity and further improves the throughput by scheduling the event-driven workloads.

Recently, some latest works, such as LSMCore [54] and SSO-LSM [55], are concentrating on accelerating liquid state machine (LSM), which is the spiking version of reservoir computing. As discussed in LSMCore, there are three main challenges for LSM acceleration: 1) any possible connections among neurons are required; 2) the computation and communication of all neurons are completed within one time step; 3) sparsity utilization is adopted to reduce hardware resource and latency overhead. Luckily, Cerebron can meet these requirements and can be used to execute LSMs.

VI. CONCLUSION

In this work, we introduce a reconfigurable architecture, Cerebron, for SNNs targeting flexibility and efficiency. To achieve flexibility, a reconfigurable compute engine with inter-CU and intra-CU reconfiguration is proposed to support various spiking layers. To achieve memory efficiency, the compute engine is also designed to exploit data reuse and guarantee parallel data access when processing different convolutions. To reduce the computation time, the inherent spatiotemporal sparsity is exploited. Besides, an online channelwise workload scheduling is designed to reduce the latency further. This design was implemented on a Xilinx XC7Z100 FPGA and verified on image segmentation and classification tasks, which uses spiking models, including ConvNet on MNIST, MobileNet on CIFAR-10, and SegNet on the MLND-Capstone dataset.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, 2012, pp. 1097–1105.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[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] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–14.

[5] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, 2016.

[6] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2018, pp. 21–30.

[7] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Aug. 2018.

[8] S. Yan *et al.*, "An FPGA-based mobilenet accelerator considering network structure characteristics," in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 17–23.

[9] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands, 2016, pp. 525–542.

[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 2074–2082.

[12] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, "Balanced sparsity for efficient DNN inference on GPU," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 5676–5683.

[13] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[14] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[16] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.

[17] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[18] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[19] J. Pei *et al.*, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[20] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-YOLO: Spiking neural network for energy-efficient object detection," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 7, pp. 11270–11277.

[21] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers Neurosci.*, vol. 13, p. 35, Mar. 2018.

[22] Y. Li, S. Deng, X. Dong, R. Gong, and S. Gu, "A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration," 2021, *arXiv:2106.06984*.

[23] Q. Cheni, B. Rueckauer, L. Li, T. Delbruck, and S.-C. Liu, "Reducing latency in a converted spiking video segmentation network," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

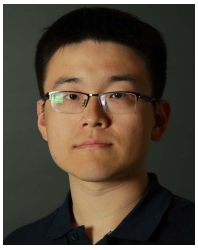
[24] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, "SpinalFlow: An architecture and dataflow tailored for spiking neural networks," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 349–362.

[25] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, Apr. 2019.

[26] J. Park, J. Lee, and D. Jeon, "A 65-nm neuromorphic image classification processor with energy-efficient training through direct spike-only feedback," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 108–119, Jan. 2020.
[27] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers Neurosci.*, vol. 11, pp. 1–12, Dec. 2017.
[28] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[29] A. K. Fidjeland and M. P. Shanahan, "Accelerated simulation of spiking neural networks using GPUs," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2010, pp. 1–8.
[30] K. Fujita, S. Okuno, and Y. Kashimori, "Evaluation of the computational efficacy in GPU-accelerated simulations of spiking neurons," *Computing*, vol. 100, no. 9, pp. 907–926, Sep. 2018.
[31] B. Kasap and A. J. van Opstal, "Dynamic parallelism for synaptic updating in GPU-accelerated spiking neural network simulations," *Neurocomputing*, vol. 302, pp. 55–65, Aug. 2018.
[32] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.
[33] S. Sen, S. Venkataramani, and A. Raghunathan, "Approximate computing for spiking neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 193–198.
[34] S. Krithivasan, S. Sen, S. Venkataramani, and A. Raghunathan, "Dynamic spike bundling for energy-efficient spiking neural networks," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2019, pp. 1–6.
[35] Q. Chen *et al.*, "A 67.5 μJ/prediction accelerator for spiking neural networks in image segmentation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 2, pp. 574–578, Feb. 2021.
[36] A. Khodamoradi, K. Denolf, and R. Kastner, "S2N2: A FPGA accelerator for streaming spiking neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2021, pp. 194–205.
[37] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021.
[38] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu, "Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.
[39] J. Zhang, H. Wu, J. Wei, S. Wei, and H. Chen, "An asynchronous reconfigurable SNN accelerator with event-driven time step update," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2019, pp. 213–216.
[40] D. Ma *et al.*, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017.
[41] D. Lee, G. Lee, D. Kwon, S. Lee, Y. Kim, and J. Kim, "Flexon: A flexible digital neuron for efficient spiking neural network simulations," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 275–288.
[42] H.-H. Lien and T.-S. Chang, "Sparse compressed spiking neural network accelerator for object detection," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 2060–2069, May 2022.
[43] S.-Q. Wang *et al.*, "SIES: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array," *J. Comput. Sci. Technol.*, vol. 35, no. 2, pp. 475–489, Mar. 2020.
[44] T. Zhou *et al.*, "Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit," *Nature Photon.*, vol. 15, no. 5, pp. 367–373, May 2021.
[45] J. Feldmann, N. Youngblood, C. D. Wright, H. Bhaskaran, and W. H. P. Pernice, "All-optical spiking neurosynaptic networks with self-learning capabilities," *Nature*, vol. 569, no. 7755, pp. 208–214, May 2019.
[46] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 5, pp. 345–358, Oct. 2018.
[47] J.-Q. Yang *et al.*, "Leaky integrate-and-fire neurons based on perovskite memristor for spiking neural networks," *Nano Energy*, vol. 74, Aug. 2020, Art. no. 104828.
[48] G. Srinivasan, A. Sengupta, and K. Roy, "Magnetic tunnel junction enabled all-spin stochastic spiking neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 530–535.
[49] S. Singh *et al.*, "NEBULA: A neuromorphic spin-based ultra-low power architecture for SNNs and ANNs," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 363–376.
[50] J. Yang, W. Fu, X. Cheng, X. Ye, P. Dai, and W. Zhao, "S^{2}Engine: A novel systolic architecture for sparse convolutional neural networks," *IEEE Trans. Comput.*, vol. 71, no. 6, pp. 1440–1452, Jun. 2021.
[51] 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, Dec. 2017.
[52] 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.
[53] C. Gao, T. Delbruck, and S.-C. Liu, "Spartus: A 9.4 TOp/s FPGA-based LSTM accelerator exploiting spatio-temporal sparsity," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 10, 2022, doi: 10.1109/TNNLS.2022.3180209.
[54] L. Wang *et al.*, "LSMCore: A 69k-synapse/mm^2 single-core digital neuromorphic processor for liquid state machine," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 1976–1989, Feb. 2022.
[55] Y. Jin, Y. Liu, and P. Li, "SSO-LSM: A sparse and self-organizing architecture for liquid state machine based neural processors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2016, pp. 55–60.

Qinyu Chen (Member, IEEE) received the B.S. degree in communication engineering from Shandong University, Jinan, China, in 2016, and the Ph.D. degree in electronic science and technology from Nanjing University, Nanjing, China, in 2021.

From 2019 to 2020, she was a Visiting Ph.D. Student with the Institute of Neuroinformatics, University of Zürich (UZH) and ETH Zürich (ETHz), Zürich, Switzerland. In 2021, she joined the Institute of Photonic Chips, University of Shanghai for Science and Technology, Shanghai, China, where she was an Associate Research Professor. She is currently a Postdoctoral Researcher with the Institute of Neuroinformatics, UZH and ETHz. Her current research interests include low-power very large-scale integration (VLSI) design for deep learning and brain-inspired algorithms.



Chang Gao (Member, IEEE) received the B.S. degree in electronics from the University of Liverpool, Liverpool, U.K., and Xi'an Jiaotong-Liverpool University, Suzhou, China, in 2015, the master's degree in analog and digital integrated circuit design from Imperial College London, London, U.K., in 2016, and the Ph.D. degree in neuroscience from the Institute of Neuroinformatics, University of Zürich and ETH Zürich, Zürich, Switzerland, in 2021.

He is currently an Assistant Professor with the Delft University of Technology, Delft, The Netherlands. His current research interest includes computer architectures for deep learning with an emphasis on recurrent neural networks.

Yuxiang Fu (Member, IEEE) received the B.S. degree in microelectronics and solid-state electronics and the Ph.D. degree in electronic science and technology from Nanjing University, Nanjing, China, in 2013 and 2018, respectively.

In 2018, he joined the School of Electronic Science and Engineering, Nanjing University, where he is currently an Associate Research Professor. His current research interests include artificial intelligence (AI)-aided computer architecture optimization and chip design network-on-chip algorithms/architectures, low-power digital systems, and 3-D integrated circuit (IC) design.