SaARSP: An Architecture for Systolic-Array Acceleration of Recurrent Spiking Neural Networks

JEONG-JUN LEE, WENRUI ZHANG, YUAN XIE, and PENG LI, University of California, Santa Barbara, California

Spiking neural networks (SNNs) are brain-inspired event-driven models of computation with promising ultra-low energy dissipation. Rich network dynamics emergent in recurrent spiking neural networks (R-SNNs) can form temporally based memory, offering great potential in processing complex spatiotemporal data. However, recurrence in network connectivity produces tightly coupled data dependency in both space and time, rendering hardware acceleration of R-SNNs challenging. We present the first work to exploit spatiotemporal parallelisms to accelerate the R-SNN-based inference on systolic arrays using an architecture called SaARSP. We decouple the processing of feedforward synaptic connections from that of recurrent connections to allow for the exploitation of parallelisms across multiple time points. We propose a novel time window size optimization (TWSO) technique, to further explore the temporal granularity of the proposed decoupling in terms of optimal time window size and reconfiguration of the systolic array considering layer-dependent connectivity to boost performance. Stationary dataflow and time window size are jointly optimized to trade off between weight data reuse and movements of partial sums, the two bottlenecks in latency and energy dissipation of the accelerator. The proposed systolic-array architecture offers a unifying solution to an acceleration of both feedforward and recurrent SNNs, and delivers 4,000X EDP improvement on average for different R-SNN benchmarks over a conventional baseline.

CCS Concepts: • **Hardware → Emerging architectures**; **Hardware accelerators;**

Additional Key Words and Phrases: Spiking neural networks, accelerators, computer architecture

ACM Reference format:
Jeong-Jun Lee, Wenrui Zhang, Yuan Xie, and Peng Li. 2022. SaARSP: An Architecture for Systolic-Array Acceleration of Recurrent Spiking Neural Networks. *ACM J. Emerg. Technol. Comput. Syst.* 18, 4, Article 68 (October 2022), 23 pages.
https://doi.org/10.1145/3510854

1 INTRODUCTION

As a brain-inspired computational model, **spiking neural networks (SNNs)** have emerged as a strong candidate for future AI platform. SNNs are considered the third-generation of artificial neural networks [35] and more closely resemble biological neurons in the brain than the more conventional second-generation ANNs based on commonly used activation functions such as sigmoid or **rectified linear unit (ReLU)**. We refer to the second-generation of neural networks as

This material is based upon work supported by the National Science Foundation under Grants No. 1948201 and No. 2000851.
Authors' address: J.-J. Lee, W. Zhang, Y. Xie, and P. Li, Electrical and Computer Engineering, University of California Santa Barbara, CA, USA, 93106; emails: {jeong-jun, wenruizhang, yuanxie, lip}@ucsb.edu.

1550-4832/2022/10-ART68
https://doi.org/10.1145/3510854

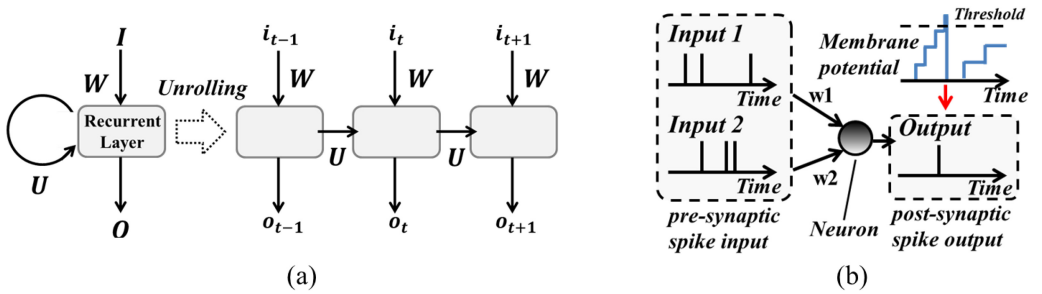

Fig. 1. (a): Schematic representation of unrolling in a recurrent layer. **W**: feedforward connection. **U**: recurrent connection. **I**/**O**: input/output. (b): Schematic representation of a spiking neuron operation.

non-spiking ANNs or simply ANNs in this article. In ANNs, signals are continuous-valued and model the averaged input and output firing rates of different neurons. A key distinction between SNNs and ANNs is that the former class of neural networks explicitly models all-or-none firing spikes across both space and time, as seen in biological neural circuits. Apart from (firing) rate based coding, SNNs can go beyond the more conventional ANNs in terms of exploring a diverse family of *temporal codes* and are intrinsically better positioned for processing complex *spatiotemporal* data [6, 35, 36]. Furthermore, biologically-inspired [5, 8, 59] and backpropagation based SNN training methods [27, 32, 47, 50, 53] have emerged and demonstrated competitive performances for various image and speech tasks.

The spiking nature of operation also set off the development of event-driven neuromorphic processors in both academia [2, 14, 15, 17, 24, 29, 37, 39, 41, 43, 44, 52] and industry including IBM's TrueNorth [2] and Intel's Loihi [17] neuromorphic chips. While there is a body of spiking neural network hardware design work using emerging devices [54], this work is primarily focused on digital SNN accelerators.

While feedforward neural networks are widely adopted, we emphasize that recurrent spiking neural networks (R-SNNs) more realistically match the wiring structure of biological brains, e.g., that of the 6-layer minicolumns of the cerebral cortex. Hence, R-SNNs form a powerful bio-inspired computing substrate that is both dynamical and recurrent. More specifically, R-SNNs can extract complex spatiotemporal features via dynamics created by recurrence across different levels and implement powerful *temporally based distributed memory* motivated by the essential role of working memory in cognitive processes for reasoning and decision-making [18].

Figure 1(a) shows a recurrent layer being unrolled through time. Unlike feedforward neural networks, the output from the recurrent layer becomes its input at the next time point. This property of recurrent layers allows for processing and storing received input information through time at different spatial locations, making them well suited for sequential learning tasks [51, 58]. Clearly, R-SNNs are significantly more complicated than their feedforward counterparts. Fortunately, R-SNN architectures and training methods with state-of-the-art performances in various image, speech, and natural language processing tasks have emerged [4, 56].

While there exists a large body of DNN hardware accelerator and dataflow work based on the non-spiking ANN models, e.g., [11, 13, 19, 22, 42], much less prior work has been devoted to SNN hardware accelerator architectures. TrueNorth [2] and Loihi [17], the two best-known industrial neuromorphic chips, are based on a many-core architecture with an asynchronous mesh supporting sparse core-to-core communication. Each core emulates 256 spiking neurons (TrueNorth) or 1,024 spiking neural compartments (Loihi) in a sequential manner. Hence, one key drawback of these two designs is that each core lacks any parallelism. While being a very important research problem, SNN dataflow has not been extensively studied before. For instance, hardware

acceleration of SNNs is performed by simply adopting a dataflow commonly used for ANN accelerators in a temporally sequential manner [7, 52], which however fails to consider spatiotemporal characteristics of SNN operation and the tradeoffs involved. The recent SNN architecture SpinalFlow is tailored for spiking neural networks using a novel compressed, time-stamped, and sorted spike input/output representation, exploiting high degree of network firing sparsity [38]. However, SpinalFlow only deals with temporally coded SNNs in which each neuron is allowed to fire once, limiting achievable accuracy of decision making [16, 28]. Furthermore, this restricted form of temporal coding makes SpinalFlow inapplicable to a large class of rate-coded and other temporally-coded SNNs in which spiking neurons fire more than once. Importantly, note that all these aforementioned SNN dataflow architectures only target feedforward SNNs [7, 38, 52].

This work is motivated by the lack of optimized accelerator architectures for general **recurrent spiking neural networks (R-SNNs)**. We propose the *first* architecture for **s**ystolic-**a**rray **a**cceleration of **r**ecurrent **sp**iking neural networks, dubbed *SaARSP*, to efficiently support spike-based spatiotemporal learning tasks.

The main contributions of this work are:

- Unlike the prior work that targets only feedforward SNNs and/or limits how temporal information is coded, the proposed SaARSP architecture is applicable to the most general R-SNN models.
- We demonstrate a novel decoupling scheme to separate the processing of feedforward and recurrent synaptic connections, allowing for parallel computation over multiple time points and improved data reuse.
- We characterize the temporal granularity of the proposed decoupling by defining a batch size called the *time window size*, which specifies the number of simultaneously processed time points, and show that **time window size optimization *(*TWSO*)*** significantly impacts the overall accelerator performance.
- Stationary dataflow and time window size are jointly optimized to trade off between weight data reuse and movements of partial sums, the two bottlenecks in latency and energy dissipation of the accelerator.
- We configure the systolic array considering layer-dependent connectivity structure to boost latency and energy efficiency.

We evaluate the proposed SaARSP architecture and dataflows by developing an R-SNN architecture simulator and using a comprehensive set of recurrent spiking neural network benchmarks. Compared to a conventional baseline that does not explore the proposed decoupling, the SaARSP architecture improves energy-delay product (EDP) by 4,000X on average for different benchmarks.

2 BACKGROUND

Conventional deep learning has demonstrated superb outcomes in many machine learning tasks. Nevertheless, SNNs have emerged as a promising biologically plausible alternative. With event-driven operations, SNN models running on dedicated neuromorphic hardware can improve energy efficiency by orders of magnitude for a variety of tasks [2, 17] . Recurrence in recurrent SNNs (R-SNNs) enable the formation of temporally based distributed memory, making them well suited for processing highly complex spatiotemporal data. In this section, we provide a brief background on spiking neural networks, including R-SNNs.

2.1 Spiking Neurons

To mimic biological neurons in the nervous system, a spiking neuron model, such as the most prevailing **leaky integrate-and-fire (LIF)** model [21] is adopted, as shown in Figure 1(b). Common to

literally all spiking neural models, operations in one spiking neuron comprise three main steps at each time point t: (1) integration of pre-synaptic spike inputs, (2) update of the post-synaptic membrane potential, and (3) conditional generation of post-synaptic spike output (action potential). As shown in Figure 1(b) and during Step (1), if a particular pre-synaptic neuron fires, the induced pre-synaptic current will be integrated by the post-synaptic neuron. From a modeling perspective, in this case, the corresponding synaptic weight between the two neurons, or more generally a quantity determined by the weight, will be added (accumulated) to the post-synaptic membrane potential. After integrating all pre-synaptic currents, in Step (2), the post-synaptic spiking neuron updates its membrane potential by adding to it the sum of integrated synaptic currents. Temporal decaying of the membrane potential can also be included if the neural model is leaky. In the last step, the spiking neuron compares its updated membrane potential with a pre-determined firing threshold and generates an output spike (action potential) if the membrane potential exceeds the threshold, as shown in Figure 1(b). The same process repeats for all time points involved in a given spike-based task.

2.2 Spiking Neural Networks

Spiking neurons are wired to form a network. The aforementioned temporal processing of individual spiking neurons are brought into a network setting in which neurons communicate with each other and perform computation by receiving and generating stereotyped all-or-none spikes both spatially and temporally. This article considers the most general (deep) multi-layer recurrent spiking neural network (R-SNN) architecture, which comprises multiple feedforward or recurrent layers with inter-layer feedforward connections. The proposed accelerator architecture intends to accelerate SNN inference on a layer-by-layer basis.

2.2.1 Feedforward Spiking Layers. Feedforward SNNs are special cases of the more general R-SNNs. The feedforward synaptic weights between a layer and its preceding layer can be represented by a weight matrix $\mathbf{W}$. At a given time point, the binary pre-synaptic/post-synaptic spikes of all neurons in the layer are now vectors of ones and zeros. For SNN acceleration on digital accelerators, it is a common practice to adopt the LIF model [21] with a zero-th order synaptic response model discretized over time, leading to three steps of layer processing at each time point t_k:

Step 1: Feedforward synaptic input integration:

$$\mathbf{f}_i^l[t_k] = \sum_{j=1}^{M^{l-1}} \mathbf{W}_{ji}^{F,l} \times \mathbf{s}_j^{(l-1)}[t_k] \tag{1}$$

Step 2: Membrane potential update:

$$\mathbf{v}_i^l[t_k] = \mathbf{v}_i^l[t_{k-1}] + \mathbf{f}_i^l[t_k] - \mathbf{V}_{leak}^l \tag{2}$$

Step 3: Conditional spike output generation:

$$\mathbf{s}_i^{(l)}[t_k] = \begin{cases} 1, & \text{if } \mathbf{v}_i^l[t_k] \geq \mathbf{V}_{th}^l \rightarrow \mathbf{v}_i^l[t_k] = 0 \\ 0 & \text{else} \qquad\quad \rightarrow \mathbf{v}_i^l[t_k] = \mathbf{v}_i^l[t_k], \end{cases} \tag{3}$$

where the $(l-1)$th layer and lth layer are the pre- and post-synaptic layers, and j and i are the neuron index in the two layers, respectively, as shown in Figure 2(a). $\mathbf{f}_i^l[t_k]$, $\mathbf{v}_i^l[t_k]$, and $\mathbf{s}_i^{(l)}[t_k]$ denote the integrated pre-synaptic spike inputs, membrane potential, and spike output of the ith neuron in layer l at time t_k, respectively. $\mathbf{W}_{ji}^F$ is the feedforward synaptic weight between neurons i and j of the two layers, and M^l is the number of neurons in layer l. $\mathbf{V}_{th}$ and $\mathbf{V}_{leak}$ are the firing

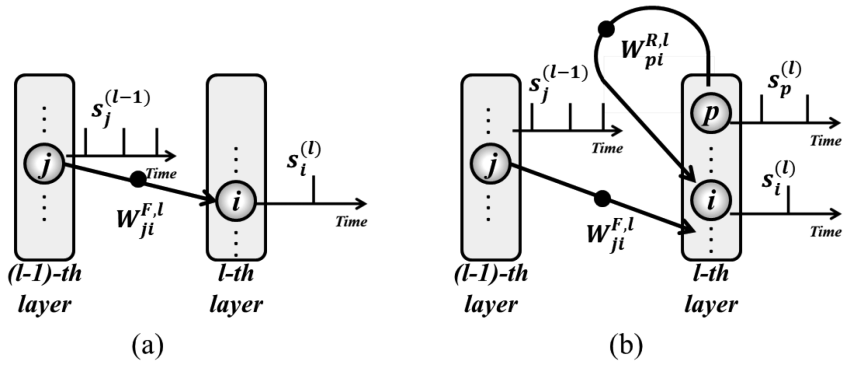

Fig. 2. (a): Schematic for layer operation in SNN. (b): Schematic for layer operation in R-SNN.

threshold and leaky parameter, respectively. In the above, the processing of each spiking neuron follows the three steps described in the previous subsection.

Step 1 of the layer processing can be seen as a matrix-vector multiplication while all other computations are scalar operations. Therefore, **Step 1** is computationally expensive and is the dominant complexity of hardware acceleration.

2.2.2 Recurrent Spiking Layers. While exploiting recurrence in the network connectivity, the network includes rich dynamics, and R-SNNs can implement temporally based local memory, as depicted in Figure 2(b). Processing a recurrent layer follows the same three-step procedure of feedforward layers with one difference. **Step 1** of feedforward layers only performs integration of feedforward synaptic inputs from the preceding layer, whereas it must consider two different types of synaptic connections for a recurrent layer: (1) feedforward inputs from the preceding layer, and (2) lateral recurrent inputs within the same layer. The latter of the two is computed in an additional step:

Step 1*: Recurrent input integration for recurrent layers:

$$\mathbf{r}_i^l[t_k] = \sum_{p=1}^{M^l} \mathbf{W}_{pi}^{R,l} \times \mathbf{s}_p^{(l)}[t_{k-1}] \tag{4}$$

$$\mathbf{f}_i^l[t_k] = \mathbf{f}_i^l[t_k] + \mathbf{r}_i^l[t_k], \tag{5}$$

where $\mathbf{s}_p^{(l)}[t_{k-1}]$ is the spike output of neuron p of layer l at time t_{k-1}, $\mathbf{r}_i^l[t_k]$ is the integrated recurrent synaptic inputs for neuron i, and $\mathbf{W}_{pi}^{R,l}$ is the recurrent synaptic weight between neurons p and i in the recurrent layer l. In (5), $\mathbf{r}_i^l[t_k]$ is added to the $\mathbf{f}_i^l[t_k]$ computed in (1) to form the total integrated inputs for neuron i. While recurrence adds significantly to the computing capability of an R-SNN, the computation of $\mathbf{r}_i^l[t_k]$, however introduces additional tightly coupled data dependencies both in space and time, i.e., across different neurons (space) and different time points (time), which is tackled by the proposed architecture as discussed in Section 3.

2.3 Computational Potential of R-SNNs

While R-SNNs produce complex network dynamics via recurrent connections, they have gathered significant recent research interests that perceive R-SNNs as a promising biologically inspired paradigm for time series data processing, speech recognition, and predictive learning. In particular,

the recurrent connections in an R-SNN form temporally based local memory, opening up opportunities for supporting broad ranges of spatiotemporal learning tasks.

Recent advances in R-SNN network architectures and end-to-end supervised training methods have led to high-performance R-SNNs that are not attainable using unsupervised biologically plausible learning mechanisms such as **spike-timing dependent plasticity (STDP)**. For example, deep R-SNNs have been trained using recent SNN backpropagation techniques [4, 56, 57], achieving state-of-the-art accuracy on commonly used spike-based neuromorphic image and speech recognition datasets such as MNIST [31], Fashion-MNIST [55], N-TIDIGITS [3], TI46 speech corpus [33], and Sequential MNIST and TIMIT [20]. Belec et al. [4] presented a powerful R-SNN architecture, called **long short-memory spiking neural networks (LSNNs)**, comprising multiple distinct leaky integrate-and-fire (LIF) recurrent spiking neural populations. Promising supervised and reinforcement learning, and **Learning-to-Learn (L2L)** capabilities have been demonstrated using LSNNs.

2.4 R-SNN Accelerators

While SNNs have gathered siginificant research interest as promising bio-inspired models of computation, only a very few works have been done on SNN hardware accelerators [15, 24, 38, 39, 45], particularly array-based accelerators [1, 6, 23, 49, 52] due to the fact that the spatiotemporal nature of the spikes make it difficult to design an efficient architecture. Importantly, these limited existing works have primarily focused on feedforward SNNs where there exist very limited works that are capable of executing R-SNNs [2, 17, 37]. For example, [1, 23, 49, 52] introduced a systolic-array-based accelerator for spiking-CNNs. However, these works are only targeting feedforward networks where efficient method to handle recurrence, which produces tightly-coupled data dependency in both time and space, has not been proposed. There is a key difficulty in developing optimized hardware architecture as strict spatiotemporal dependency resides in R-SNNs. Furthermore, RNN accelerator designs and techniques are incompatible with R-SNNs due to unique properties of spiking models, such as disparity in data representation which lead to different trade-offs in data sharing and storage compared to non-spiking models.

There exist few types of neuromorphic hardware that are capable of executing R-SNNs [2, 17, 37]. Akopyan et al. [2] and Davies et al. [17] are the two best-known industrial large-scale neuromorphic chips, based on a many core architecture. Both chips are fully programmable and have capability of executing R-SNNs, as Akopyan et al. [2] is based on intra-core crossbar memory and long-range connections through an inter-core network and Davies et al. [17] adopt neuron-to-neuron mesh routing model [9, 37] presented a multicore neuromorphic processor chip that employs hybrid analog/digital circuit. With novel architecture that incorporates distributed and heterogeneous memory structures with a flexible routing scheme, [37] can support a wide range of networks including recurrent network.

However, existing architectures are limited to process R-SNNs in a time-sequential manner, which requires alternating access to two different types of weight matrices for every time point, i.e., feedforward weight matrix and recurrent weight matrix. The major shortcomings in the above architectures originate from the stereotype that, both the feedforward and recurrent inputs must be accumulated to generate final activations at a given time point, which are the recurrent inputs to the next time point, before the next time point can be processed. For example, large-scale multicore neuromorphic chips, IBM's TrueNorth with 256M synapses [2] and Intel's Loihi with 130M synapses [17], are based on the assumption that all weights of the network are fully stored on-chip. Using a very large core memory can reduce inefficiencies, i.e., lack of parallelism and weight reuse, which makes the weight reuse and data movement less important compared to many other practical cases.

On the other hand, the main idea of our article allows parallel compute over multiple time points as opposed to existing architectures and maximizes weight reuse benefits. Our main target is to minimize data-movement energy cost for memory-intensive SNN accelerators, especially for a practical case that the accelerator cannot load entire weight matrices. Without the proposed idea to minimize the data movement, processing R-SNNs require alternating access to two different types of weight matrices for every time point. Based on the fact above, we defined the baseline architecture (without decoupling) which keeps the essential ideas of existing SNN accelerator works and extended them for recurrent SNNs. With inherent advantages in exploiting spatiotemporal parallelism as will discussed in Section 3.2, our main accelerator is based on systolic array architecture. This work is motivated by the lack of optimized accelerator architectures for general recurrent spiking neural networks (R-SNNs).

3 SAARSP: PROPOSED ARCHITECTURE

We present the proposed architecture for **s**ystolic-**a**rray **a**cceleration of **r**ecurrent **sp**iking neural networks, dubbed *SaARSP*, which accelerate a given R-SNN in layer-by-layer manner. SaARSP addresses the data dependencies introduced by temporal processing in R-SNNs via decoupled feedforward/recurrent synaptic integration scheme and a novel time window size optimization to enable an optimal time-domain parallelism, and hence reduces latency and energy. It supports both the **output stationary (OS)** and **weight stationary (WS) dataflows** for maximized data reuse.

3.1 Decoupled Feedforward/Recurrent Synaptic Integration

As discussed in Section 2.2.2, recurrence in R-SNNs introduces tightly coupled data dependencies both in space and time, which may prevent direct parallelization in the time domain and hence limit the overall performance. We address this challenge by proposing a parallel processing of spike input integration by decoupling the integration of feedforward and recurrent synaptic inputs. The key idea here is to re-structure the spike integration process, as shown in Figure 3(b). One key observation is that while the complete processing of a recurrent layer involves temporal data dependency, the feedforward synaptic input integration, i.e., **Step 1** corresponding to (1), has no temporal data dependency and can be parallelized over multiple time points. And then, the following steps of recurrent input integration (**Step 1***), membrane potential update (**Step 2**), and spike output generation (**Step 3**) are done in a sequential manner time-step by time-step. For example, in conventinal approaches, spike integration step at a given time point t_k requires two different weight matrix, which is repeated for every time point, as depicted in Figure 3(a). However, decoupling feedforward and recurrent stage enables to reuse each of the two weight data for consecutive time points in a row, as shown in Figure 3(b). This decoupling scheme can be represented based on two macro-steps below:

Step A: Feedforward spike input integration for t_k to t_{k+TW-1} over a time window of TW points.

$$\mathbf{f}_i^l[t_k, \ldots, t_{k+TW-1}] = \sum_{j=1}^{M^{l-1}} \mathbf{W}_{ji}^{F,l} \times \mathbf{s}_j^{(l-1)}[t_k, \ldots, t_{k+TW-1}]. \tag{6}$$

Step B: Process **Step 1*, 2, and 3** for t_k to t_{k+TW-1} sequentially.

In the above, TW is the time window size which specifies the temporal granularity of the decoupling. For instance, feedforward synaptic integration step is processed first, over TW time points, followed by the rest steps in sequential manner. Processing the feedforward input integration over multiple time points as in (6) is possible since the output spikes from the preceding layer over the same time window, which are the inputs to the present layer, have already been computed at this

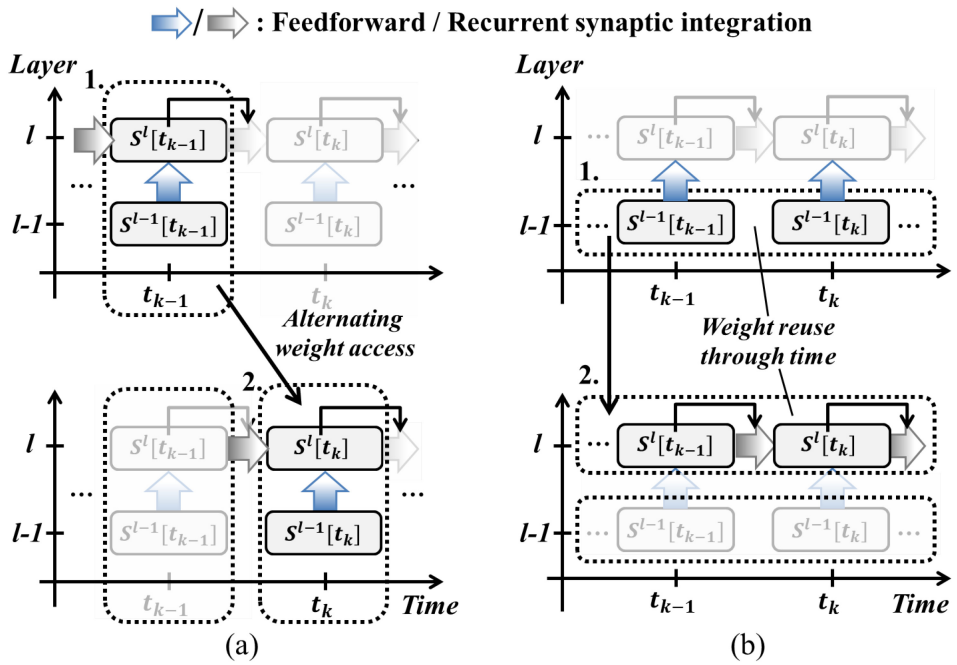

Fig. 3. Schematic representation of (a): Time-serial processing in conventional SNNs, (b): Decoupling scheme to separate feedforward and recurrent steps. Layer l: Recurrent layer.

point in the layer-by-layer processing sequence. We further introduce the optimization technique in terms of time window size in the later section.

Weight data reuse opportunity. Importantly, *this decoupling scheme opens up two weight matrix data reuse opportunities*. First all, it is easy to see that the feedforward weight matrix $\mathbf{W}^{F,l}$ can be reused across all TW time points in **Step A**. We group the rest of the steps (**Step 1***, **2,** and **3**) into **Step B** and process it sequentially. This is because that the layer's spike outputs at the present time point cannot be determined without knowing the spike outputs at the preceding time point, which feed back to the same recurrent layer via recurrent connections according to (2), (3), (4), and (5). Nevertheless, it is important to note that despite the fact that **Step B** is performed sequentially, decoupling it from **Step A** allows for reuse of recurrent weight matrix $\mathbf{W}^{R,l}$ across TW time points in **Step B**. The decoupling scheme offers a unifying solution to enhance weight data reuse for both feedforward integration and recurrent integration, and are applicable to both feedforward and recurrent SNNs.

3.2 Proposed SaARSP Architecture

Without the proposed time-domain parallelism, accelerating recurrent layers in the conventional approach effectively operates on a 1-D array that performs serial processing time point by time point, as shown in Figure 3(a). As discussed above, there exist very limited works for SNN hardware accelerators, and most of prior works have primarily focused on feedforward SNNs where the decoupling scheme has not been explored. We keep the essential ideas of existing SNN accelerator works while extending them for recurrent SNNs without time-domain parallelism, giving rise to the baseline architecture adopted throughout this article for comparison.

In contrast, Figure 4 shows the overall SaARSP architecture, comprising controllers, caches, and a reconfigurable systolic array. Memory hierarchy design is critical to the overall performance

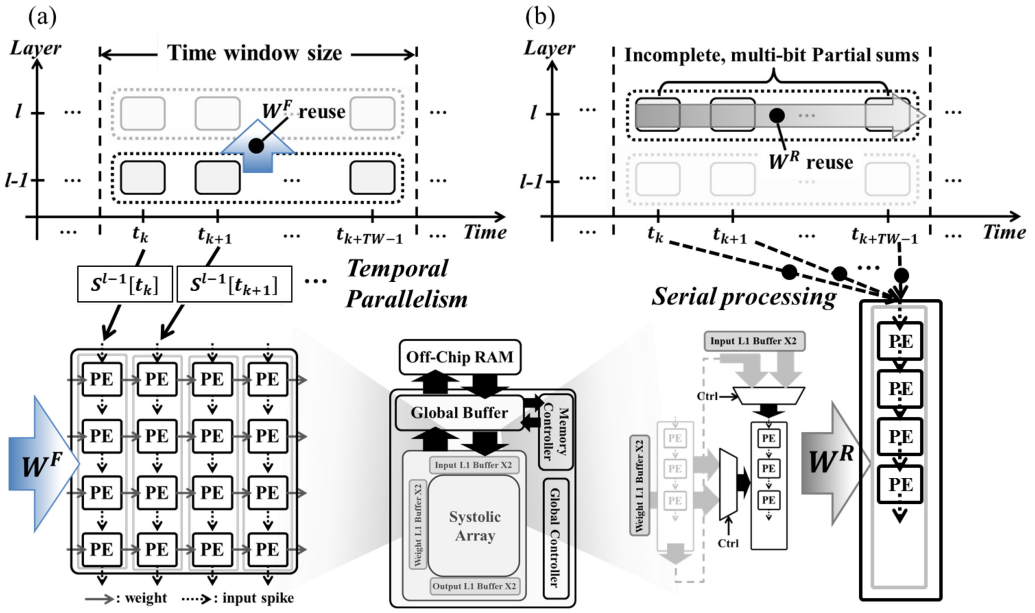

Fig. 4. Overview of the proposed SaARSP architecture. (a): Array computation for feedforward integration (**Step A**, OS dataflow) (b): Array computation for recurrent integration (**Step B**, OS dataflow).

of neural network acceleration due to its memory-intensive nature. As a standard practice, we adopt three levels of memory hierarchy, as shown in Figure 4: (1) DRAM, (2) Global buffer, and (3) double-buffered L1 cache [30, 46].We employ programmable data links with simple control logic among the PEs in the 2-D systolic array such that it can be reconfigured to a 1-D array. Each processing element (PE) consists of a scratchpad and an AC unit that accumulates the pre-synaptic weights when the corresponding input spikes are presented. The SaARSP architecture supports two different stationary dataflows based on systolic array.

Systolic Arrays. A major benefit in using systolic arrays is parallel computing in a simultaneous row-wise and column-wise manner with local communications [52]. Furthermore, systolic arrays are inherently suitable for exploiting spatiotemporal parallelism, i.e., across different neurons (space) and across multiple time-points (time). Especially, systolic arrays are well suited for our main idea to perform parallel computing across time-domain. In addition, separately, the row-wise and the column-wise unidirectional links can be adopted and optimized for the specific data disparity in SNNs.

Stationary Dataflows. For non-spiking ANNs, many previous works have proposed to leverage stationary dataflows to reduce expensive data movement [12, 13, 46]. By keeping one type of data (input, weight, or output) in each PE, an input stationary (IS), weight stationary (WS), and output stationary (OS) dataflow reduces the movement of the corresponding data type. However, a stationary dataflow may result in a different impact for spiking models, considering their unique properties. We explore OS and WS flows to mitigate the movements of large volumes of multi-bit Psum and weight data, respectively. We do not consider input stationary (IS) dataflows that are commonly used in conventional DNN accelerators because binary input spikes are of low volume, and reusing binary input data offers limited benefits.

3.2.1 Output Stationary Dataflow. The two processing steps are executed on SaARSP under the OS dataflow as follows. In **Step A**, the systolic array is configured into a 2-D shape to exploit

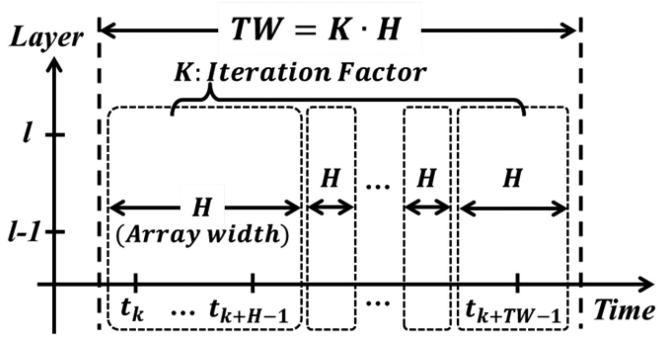

Fig. 5. Operate the array accelerator with a chosen time window size TW for K array processing iterations.

temporal parallelism, both across space and time, as illustrated in Figure 4(a). Input spikes at different time points within the time window are fetched to the corresponding columns in the array from the top. Spike input integration for different time points is processed column-wise, where input spikes propagate vertically from top to bottom. The feedforward weight matrix is fetched from the left and then propagates horizontally from left to right, enabling weight data reuse at different time points.

Since **Step B** is performed sequentially, the 2-D systolic array is reconfigured into a 1-D array to fully utilize the compute resources to maximize spatial parallelism at each individual time point as shown in Figure 4(b).

3.2.2 Weight Stationary Dataflow. In WS, weight data as opposed to Psums resides stationary in the scratchpads to maximize weight data reuse. While the weight data are in the PEs, input spikes and Psums propagate vertically through the PEs. Unlike OS, there is no horizontal data propagation in WS except when the array retrieves new weight data for computation. WS suffers from increased cost for storing and moving Psums while further maximizing weight data reuse.

3.3 Time-Window Size Optimization (TWSO)

Processing across TW time points within the chosen time window as in (6) with the decoupling scheme allows the exploitation of temporal parallelism. However, there exist a fundamental trade-off between weight reuse and Psum storage. The key advantage of decoupling, i.e., weight data reuse across time points, may be completely offset by the need for storing incomplete partial results across multiple time points [48]. Blindly decoupling can exacerbate the above trade-off, and even result in worse performance, as shown later in Figures 7 and 8.

In order to address above issue, we introduce a novel time window size optimization (TWSO) technique to address the fundamental trade-off, optimize the window size TW to maximize the latency and energy dissipation benefits. We present the first work to explore temporal-granularity in terms of TWSO, which is much powerful and flexible than solely applying the decoupling scheme [48].

TWSO Address the Fundamental Tradeoff. As discussed above, the number of time steps that can be executed simultaneously in one array iteration is limited by the array width H. To accommodate higher degrees of time domain parallelisms, we define **K** as the time-iteration factor such that $TW = K \cdot H$, i.e., the parallel integration of feedforward pre-synaptic inputs of a recurrent layer consumes K array processing iterations, as shown in Figure 5. For a given **K**, the array reuses a single weight matrix, either $\mathbf{W}^F$ or $\mathbf{W}^R$, for TW time points.

On the other hand, there may exist optimal choices for the value of TW (K). According to Figure 5 and Equation (6), the decoupling scheme batches the feedforward and recurrent input integration steps, and hence it enables reuse of both the feedforward and recurrent weight matrices $\mathbf{W}^F$ and $\mathbf{W}^R$ over multiple time points, avoiding expensive alternating access to them across **Step A** and **Step B**. However, parallel processing in the time domain can degrade the performance due to an increased amount of partial sums (Psums). Upon completion of **Step A** across many time points, there exists a large amount of incomplete, multi-bit partial sum (Psum) data waiting to be processed in **Step B**. More Psums can result in degraded performance due to the increased latency and energy dissipation of Psum data movement across the memory hierarchy. TWSO address the aforementioned fundamental tradeoff by exploring granularity with varying time-window size, and finding optimal point for the tradeoff. TWSO is generally applicable to both non-spiking RNNs and spiking RNNs.

TWSO Offers Application Flexibility. Typically, a decoupling scheme has a key limitation since it does not provide benefit beyond the first layer due to temporal dependency. This is because as the RNN is unrolled over all time points, both the feedforward and recurrent inputs must be accumulated to generate a recurrent layer's final activations, which are part of the inputs to the next layer, before the next layer can be processed. However, TWSO subdivides all time points into multiple time windows and performs decoupled accumulation of feedforward/recurrent inputs with a granularity specified by time window size, allowing overlapping the processing of different recurrent layers across different time windows. For example, upon completing processing time window i for recurrent layer k, the activities of layer k in time window (i + 1) and those of layer (k + 1) in time window i can be processed concurrently. Therefore, with TWSO, decoupling can be applied to multiple recurrent layers concurrently.

TWSO Is Specifically Beneficial for Spiking Models. Since the partial sums of each layer are multi-bit while its outputs are only binary, the benefit of increased weight data reuse resulted from decoupling may be more easily offset by the increased storage requirement for multi-bit Psums. Optimizing the granularity of decoupling becomes even more critical for R-SNNs as addressed by TWSO.

Also, TWSO alleviates bottleneck due to data bandwidth by allowing weight reuse across chosen time window size as opposed to iterative weight access, and thus the time window size is not enforced due to the memory bandwidth compared to conventional approaches.

As we demonstrate in our experimental studies in Section 5, TWSO can significantly improve the overall performance.

4 EVALUATION METHODOLOGY

We introduce an analytic architecture simulator to assess the latency, memory access, and energy dissipation of the proposed SaARSP architecture and compare it with a baseline. The simulator takes the user-specified accelerator specification and network structure such as the number of PEs, systolic array configuration, size of the cache at each level, stationary scheme, and SNN network structure including the numbers of layers and neurons and feedforward and recurrent connectivity factors, as summarized in Table 1. We compare different accelerator architectures using a large number of feedforward and recurrent SNNs that are synthetic or adapted from the neuromorphic computing community.

4.1 Modeling of Systolic Array and Memory

4.1.1 Systolic Array. The developed simulator can consider any type of systolic arrays, including 1-D and rectangular 2-D systolic arrays. Each PE in the systolic array consists of mainly two

Table 1. A High-Level Overview of the User-Specified Inputs to the Simulator

Input parameter	Description
Array configuration	Specifications of systolic array: array height/width, number of PEs, size of scratchpad, memory per PE, etc.
Memory configuration	Specification of memory hierarchy: Cache sizes for each partitioned storage (input, weight, output) at each level.
Time window size	Range of support: Min: 2D systolic array width Max: All time points of the task
Network structure	Numbers of layers and neurons per layer, and connectivity factors between layers.
Stationary scheme	Choice of different stationary schemes: Output stationary/Weight stationary

components: (1) accumulate (AC) unit and (2) a small scratchpad memory. Unlike in the conventional non-spiking neural networks that employ **multiply-and-accumulate (MAC)** units, each SNN layer takes binary spikes as inputs and outputs binary spike outputs, and hence simpler AC units can be used instead. For the same reason, the accumulation of the weight values to the neural membrane potential only happens when the corresponding input spike is nonzero. For each accelerated SNN layer, the input and weight data is fed from the left and top edges of the array and then propagates to adjacent PEs within the same row or column through a unidirectional link. Throughout this article, we assume that multi-bit weight data propagates from left to right, and binary input data propagates from top to bottom.

4.1.2 Memory Hierarchy. Neural network accelerators, including SNN accelerators, are memory intensive. We adopt three levels of memory hierarchy, which is standard practice: off-chip RAM (DRAM), on-chip L2/L1 caches, and small scratchpad memory in each PE. All on-chip memories are modeled as double buffers, which is also a standard practice used in many previous works [30, 46]. With simultaneous reads/writes, double buffering hides the access latency to higher-level caches. Similar to many other analytic models, we partition each level of memory to separately store input spikes, weight matrices, and generated output spikes or partial sums (Psums), respectively [46].

4.2 Performance Modeling

With given inputs described above, the simulator generates addresses for each type of data with respect to the inputs/outputs required to be fed on the systolic array. The simulator estimates memory access and latency based on cycle-accurate read/write traces for each cache level. As the most widely used method, we follow the estimation method presented in [30, 46].

4.2.1 Memory Access. For an inputted spiking neural network, the simulator generates a sequence of array computations to be completed. Following the produced sequence of systolic array computation, the simulator determines the required data. If the data needed for a certain array computation is absent in local scratchpad memories, memory accesses to the higher-level L1/L2 cache will be issued.

4.2.2 Latency. The systolic array processes the sequence of array computation with the data in a working buffer while the loading buffer in the L1 cache continuously fetches the data needed for the next systolic array computation. Fetching of the required data may result in stalls of the array. Therefore, the accelerator's latency per iteration is determined by the worst latency resulted from array computation and data access.

Table 2. Architecture Specifications

Components	SaARSP
PEs	256 (16 × 16)
ALU in PEs	8bit - Adder, Comparator
Global Buffer Size	1MB (250KB/500KB/250KB)
L1 Cache Size	200KB (100KB/200KB/100KB)
Scratchpad Size	32 × 8-bit
DRAM Bandwidth	30GB/sec
Bit precisions	Weight/Membrane Potential - 8bit Input/Output Spike - 1bit (binary)

4.2.3 Energy Dissipation. Energy dissipation of an accelerator is mainly due to (1) array computation and (2) memory access. With a given neural network structure, computation energy is evaluated by multiplying the total number of AC operations with energy per AC operation [25]. The energy dissipation of each read/write memory access is evaluated using CACTI [10] configured for the 32-nm CMOS technology. The total memory access energy dissipation is estimated based on the number of read/write accesses and the energy consumed per access [30, 46].

4.3 SaARSP Array Specification and Baseline Architectures

We specifically consider the systolic array specifications in Table 2 for the proposed SaARSP architecture in our experimental studies in Section 5.

Critically, time windows size (TW) specifies the degree of time-domain parallelism of the SaARSP architecture, i.e., the number of simultaneously processed time points during layer computation as in (6). As discussed in Section 3.3, for a systolic-array with with **H**, we define **K** as the time-iteration factor for more clear illustrative purpose, such that $TW = K \cdot H$. In other words, the parallel integration of feedforward pre-synaptic inputs of a recurrent layer consumes a multiple of K array processing iterations. We evaluate the proposed SaARSP as K varies widely from 1 to 64.

For comparison purposes, we consider a baseline architecture which lacks the proposed decoupled feedforward/recurrent synaptic integration approach (Section 3.1) and hence processes each layer in a time-sequential manner, e.g., time step by time step. As aforementioned in Section 3.2 III-B, our baseline well represent the existing feedforward spiking hardware accelerators, which lacks temporal parallelism and decoupling scheme. For fair comparison, we reconfigure the systolic array into a 1-D array with an equal number of PEs such as the same amount of hardware resources are utilized for parallel compute in space.

4.4 Spiking Neural Network Benchmarks

4.4.1 Characterization of R-SNNs. Compared to the conventional ANN research area, there are much fewer publicly available network models and datasets for spiking neural networks, particularly for the more complex R-SNNs. We use a comprehensive set of feedforward and recurrent SNNs that are either synthetic or adapted from the neuromorphic computing community to demonstrate the performance of the proposed SaARSP architecture in Section 5. Feedforward SNNs are also considered since they are a specific case of R-SNNs and SaARSP can provide a unifying solution to both feedforward and recurrent SNNs.

We recognize two essential metrics that characterize the structure of a given R-SNN layer or network: (1) **topology**, and (2) **connectivity factors**. The most general (deep) multi-layer network architecture is considered where the R-SNN comprises multiple feedforward or recurrent layers with inter-layer feedforward connections. As commonly defined, feedforward layers have no intra-layer (later) connections. On the other hand, recurrent layers do have intra-layer connections that form recurrent loops within the layer. We further consider two recurrent layer topologies as shown

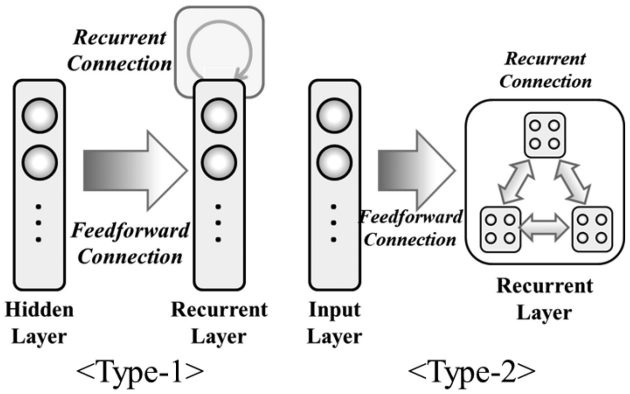

Fig. 6. Two spiking recurrent layer topologies: (a) Type 1 - uniform, and (b) Type 2 - population based.

Table 3. Inference Accuracy of Trained R-SNNs of Type 1 Topology on Common Neurmorphic Image/Speech Recognition Datasets with C_{R-R}=1.0, C_{R-R}=0.2

Dataset	Network structure	Timesteps/ Epochs	Accuracy
MNIST	784-400(H)-400(R)-10	5/100	98.47%
MNIST	784-1000(H)-1000(R)-10	5/100	98.62%
Fashion MNIST	784-400(H)-400(R)-10	5/100	89.86%
Fashion MNIST	784-1000(H)-1000(R)-10	5/100	90.00%
TI Alpha	78-400(H)-400(R)-10	100/200	93.03%
TI Alpha	78-1000(H)-1000(R)-10	100/200	93.72%
N-MNIST	2312-400(H)-400(R)-10	100/100	98.56%
N-MNIST	2312-1000(H)-1000(R)-10	100/100	98.88%
NTIDIGITs	64-400(H)-400(R)-11	300/400	89.05%
NTIDIGITs	64-1000(H)-1000(R)-11	300/400	90.78%

in Figure 6 that were adopted by the neuromorphic computing community and demonstrated state-of-the-art performances: (Type 1) uniform [56], (Type 2) population-based, comprising multiple distinct neural populations [4].

Moreover, recurrent spiking layers can be characterized by inter- and intra-layer connectivity, which we specify using two connectivity factors C_{H-R} and C_{R-R}. C_{H-R} specifies the average connection probability of each pair of two neurons between the recurrent layer and the preceding layer. Similarly, C_{R-R} specifies the average connection probability between each pair of two neurons within the recurrent layer. The two connectivity factors are used for both Type 1 and Type 2 recurrent layers, and have a significant impact on the throughput and energy dissipation of hardware acceleration as will be discussed in Section 5.

We trained a number of R-SNNs with Type 1 recurrent layers using backpropagation [57] on a set of widely-adopted spiking neural based image/speech reorganization datasets MNIST [31], Fashion MNIST [40, 55], Neuromorphic MNIST (N-MNIST), TI Alpha (English letter subsect of the TI-46 speech corpus) [33], and NTIDIGITS (the neuromorphic version of the speech datasets TIDIGITS) [3]. Note that the examples in the non-neuromorphic datasets above were converted into a spiking form following the standard practice, such as [34]. In Table 3, first layers in network structures are equal to the number of input spikes for each dataset, where the inputs are all-or-none binary spikes. Table 3 shows the competitive test accuracy of R-SNNs with Type 1 recurrent layers of two different sizes: 400(H)-400(R) and 1,000(H)-1,000(R), where the numbers of spiking neurons in the preceding layer (H) and the recurrent layer (R) are both set to 400 and 1,000, respectively.

Table 4. R-SNN Benchmarks Used in This Work

Tag	Network structure	Avg. connectivity
B1	T1:400(H)-400(R)	C_{H-R} = 1.0 / C_{R-R} = 0.2
B2	T1:400(H)-400(R)	C_{H-R} = 0.7 / C_{R-R} = 0.5
B3	T1:1000(H)-1000(R)	C_{H-R} = 1.0 / C_{R-R} = 0.2
B4	T1:1000(H)-1000(R)	C_{H-R} = 0.7 / C_{R-R} = 0.5
B5	T2:80(H)-(200 + 80 + 120)(R)	C_{H-R} = 0.7 / C_{R-R} = 0.4
B6	T2:80(H)-(200 + 80 + 120)(R)	C_{H-R} = 0.4 / C_{R-R} = 0.7
B7	T2:300(H)-(500 + 200 + 300)(R)	C_{H-R} = 0.7 / C_{R-R} = 0.4
B8	T2:300(H)-(500 + 200 + 300)(R)	C_{H-R} = 0.4 / C_{R-R} = 0.7

Type-2 recurrent layer topology was adopted in the so-called long short-memory spiking neural network (LSNN) model of [4] in which the recurrent layer consists of three distinct leaky integrate-and-fire (LIF) spiking neural populations: (1) excitatory, (2) inhibitory, and (3) adaptive, as shown in Figure 6. It was also shown that this R-SNN architecture can support the powerful Learning-to-Learn (L2L) capability as a spiking compute substrate in [4].

4.4.2 *Adopted SNN Benchmarks.*

Recurrent Layers. For comprehensive evaluation of the proposed accelerator architecture, we adopt eight R-SNN benchmarks with varying connectivity factors as summarized in Table 4. The first four networks **B1**~**B4** employ Type-1 (uniform) topology while **B5**~**B8** are based on Type-2 (population-based) topology. For the former group, each recurrent layer is specified by the numbers of neurons in the preceding layer (H) and within the targeted recurrent layer (R). For the latter group, the number of neurons in each of three populations in the recurrent layer is shown.

Feedforward Layers. Layers in a multi-layer R-SNNs in general can be both feedforward and recurrent. Additionally, several feedforward spiking layers with different inter-layer connectivity factors are considered as special cases of R-SNNs. As the network goes deeper with more layers and more feedforward layers are included in the network, feedforward layers may account for a considerable portion of the total workload. It shall be noted that the proposed time-domain parallel scheme can be also applied to process feedforward connections. In this sense, SaARSP serves as a unifying solution to acceleration of both feedforward and recurrent layers.

5 RESULTS

We perform a comprehensive evaluation of the proposed SaARSP architecture based on both OS and WS dataflows following the setups described in Section 4.

5.1 Acceleration of Feedforward Layers with Output Stationary Dataflow

The proposed SaARSP architecture can accelerate feedforward layers as a simpler special case (Section 4.4.2). Under the output stationary dataflow, Figure 7 shows the normalized latency and energy of processing feedforward layers of 400 spiking neurons with different time window sizes in two different network configurations with 78 and 784 neurons in the preceding layers, respectively. The inter-layer connectivity factor C_{H-R} is 1.0 for all layers (Section 4.4.1). Two recurrent layers with intra-layer connectivity factor C_{R-R} set to 0.3 are also included for comparison purposes. The recurrent layers consume greater latency and energy than the feedforward counterparts due to the additional computation caused by recurrent connections. Nevertheless, processing of feedforward connections contributes a considerable overhead. The latency and energy of the feedforward layers with more connections shown in Figure 7(b) can be significantly reduced by increasing the time window size, i.e., the K value, due to the fact that larger time window sizes offer more weight data reuse opportunities for weight reuse. That is, reusing the same weights for processing larger number of time points mitigates expensive data movement from higher- to lower-level caches.

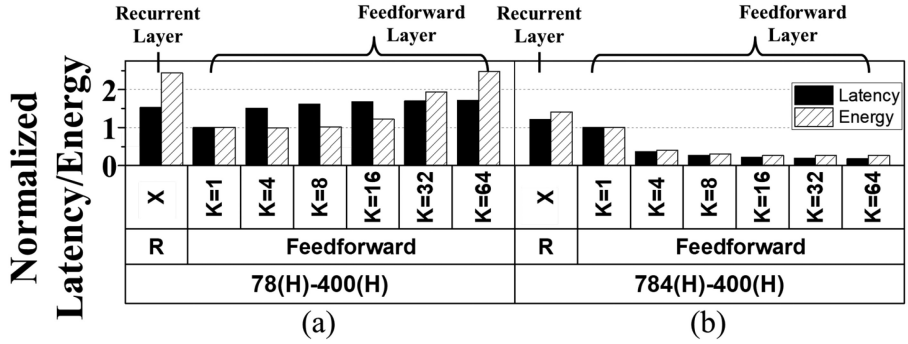

Fig. 7. Normalized latency/energy of two feedforward layers in comparison with recurrent layers, with C_{H-R} = 1.0 and C_{R-R} = 0.3. The values are normalized to those of the feedforward layer with time-iteration factor K = 1.

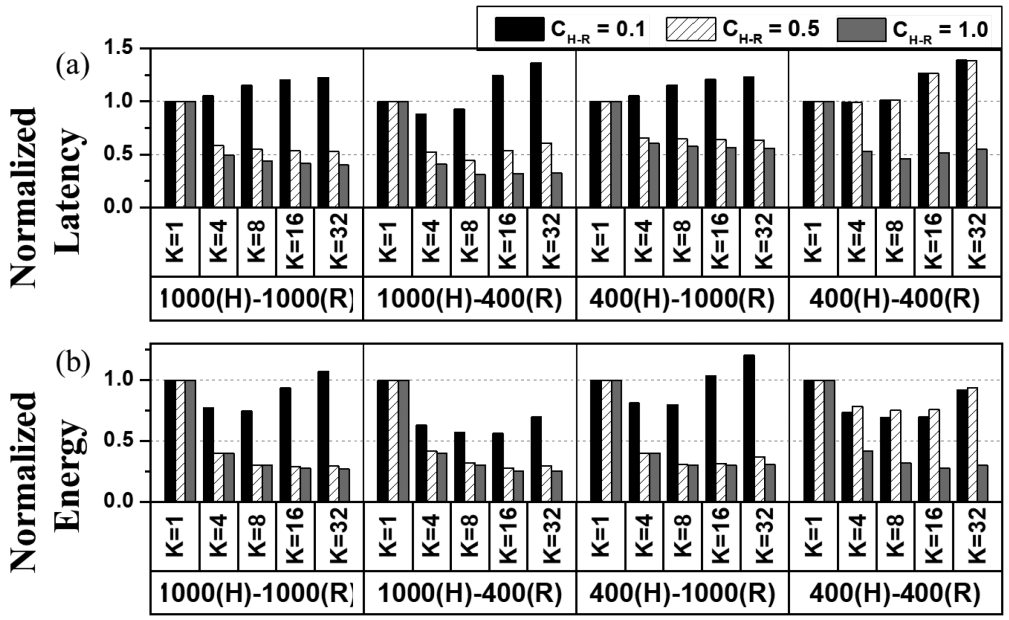

Fig. 8. Normalized (a) latency, and (b) energy dissipation of recurrent layer acceleration under OS dataflow with C_{R-R} = 0.5.

However, it shall be noted that merely increasing time window size may degrade the overall performance, which is reflected in Figure 7(a). These feedforward layers incur reduced weight access overhead due to their fewer connections. On the other hand, employing a greater time window size produces more multi-bit Psum data that must be kept over a larger number of time points, leading to a degradation in overall performance.

5.2 Acceleration of Recurrent Layers with Output Stationary Dataflow

Before presenting a more complete evaluation of recurrent layer acceleration in Section 5.3, we first analyze latency and energy dissipation tradeoffs of accelerating Type-1 recurrent layers with varying time window size under the output stationary dataflow.

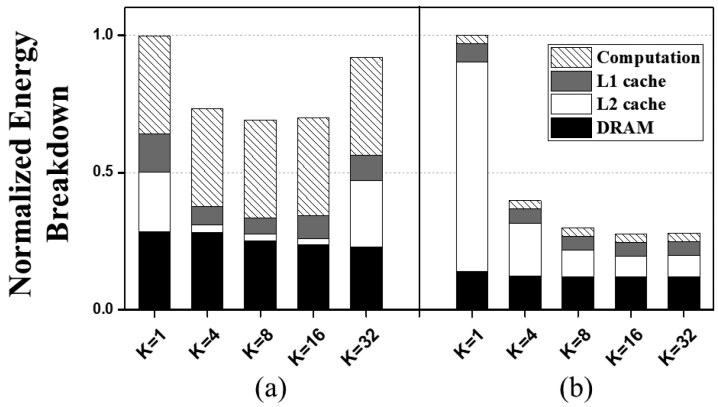

Fig. 9. Normalized energy dissipation of recurrent layer acceleration under output stationary dataflow.

5.2.1 Latency. Figure 8(a) shows the normalized latency under different connectivity and time window sizes settings with the intra-layer connectivity factor C_{R-R} = 0.5. In most of cases, larger time window size (K) values are preferred at the highest inter-layer connectivity factor (C_{R-R} = 1.0). At smaller C_{R-R} values, latency grows with time window size or starts to rise after time window size is increased beyond some point. These observations may be understood based on the role of time window size. Larger time window sizes render more weight data reuse over a greater number of time points. However, after a certain point, latency starts to increase due to a relatively huge amount of generated Psums. However, increasing time window size leads to a greater amount of multi-bit Psums that are created over more time points and must be stored prior to the completion of binary spike output generation (Step 3 in Section 2.2). The increased Psum data movement slows down the processing, particularly under low C_{H-R} values for which the benefit of weight data sharing is less due to the reduced dominance of feedforward connections.

5.2.2 Energy Dissipation. Figure 8(b) shows the normalized energy dissipation of the same set of recurrent layers. Tradeoffs between weight data reuse and Psum data are similar to the ones discussed for latency. Larger window sizes induce more storage and movement of Psums while allowing for more weight data reuse. When the amount of the Psum data exceeds the capacity of lower-level caches, e.g., the scratchpad in PEs and L1 cache, expensive memory access to higher-level caches and DRAM increases. Figure 9 shows the breakdown of energy dissipation of two recurrent layers.

5.3 Comprehensive Evaluation and Optimization of Recurrent Layer Acceleration

We investigate how the proposed decoupled feedforward/recurrent input integration, time window size, and stationary dataflow can be jointly applied/optimized for a given network structure based on the eight R-SNN layer benchmarks (**B1** to **B8**) of two different types of Section 4.4.2. We compare the SaARSP architecture with baseline array, which has the same number of PEs but is unable to explore the proposed time-domain parallelism. The results in Figures 10 and 11 are normalized to this baseline.

5.3.1 Tradeoffs between Weight Data Reuse and Psum Movement. In order to elaborate the impact of TWSO, we sweep the time-iteration factor K as shown in Figure 10. Figure 10 reports the latency and energy of the Type-1 and Type-2 networks with various time window sizes (K values)

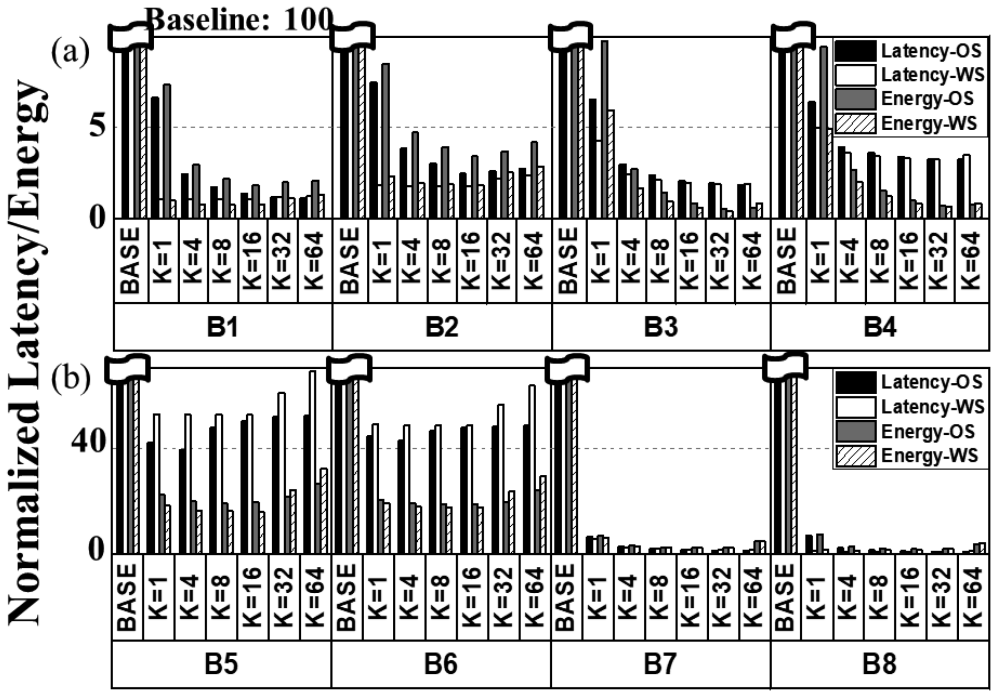

Fig. 10. Latency/energy of (a) Type-1, and (b) Type-2 recurrent networks with OS and WS dataflows normalized to that of the baseline design, which follows conventional approaches.

and the two stationary dataflows. Clear benefits can be brought by larger time window sizes when the size of the pre-synaptic layer and the number of feedforward connections are greater or comparable to their post-synaptic layer/recurrent connection counterparts. In such cases, the benefit of weight reuse is maximized since the feedforward connectivity becomes more dominant. Certainly, the impact of weight data reuse is layer/connectivity dependent. For instance, in the case of layers B2, B4, and B5, increasing time window size may lead to high-volume of multi-bit Psum storage and movement, offsetting the benefit of weight data reuse and degrading the overall performance. This fact signifies the importance of performing the proposed TWSO on a layer-by-layer basis.

For example, we observe clear indications of the impact of Psums in Figure 10. In type-2 networks, the network consists of a relatively smaller pre-synaptic layer size. As shown in Figure 10, benefit from weight reuse is concealed by the increased impact of Psums. Compared to the type-1 networks, with high-volume of weights, Psum is a more dominant factor in type 2 benchmark, making the optimal window size small.

5.3.2 Impact of Stationary Dataflows. Figure 10 present a clear difference between the output stationary (OS) and weight stationary (WS) dataflows, two dataflows suitable for SNNs due to the multi-bit nature of weight and Psum data. In most of the benchmarks, WS shows better results especially when there is no window size optimization. For the benchmarks with considerably large pre-synaptic layers, WS shows better weight reuse and thus better performance overall.

However, the performance of WS tends to be more directly affected by the amount of Psums since Psums are directly stored in higher-level caches without going through scratchpads, as manifested in the results of B5 and B6. For benchmarks with fewer weight parameters, OS produces

Table 5. Detailed Performance Metrics: PE Utilization, Number of Operations and Data Reuse in Each Integration (feedforward/recurrent) Step

Tag	B1	B2	B3	B4	B5	B6	B7	B8	Avg.
PE utilization %	87.7	75.9	89.6	50.7	79.8	78.1	67.6	45.6	**71.9**
#Op ratio %	93.7 /	84.3 /	90.0 /	74.9 /	58.2 /	41.5 /	53.2 /	34.3 /	**66.3 /**
(F/R)*	6.3	15.7	10.0	25.1	41.8	58.5	46.8	65.7	**33.7**
Data reuse (R)**	7.4X	4.9X	1.1X	1X	6.8X	4.3X	4.4X	2.2X	**4.0X**

#Op ratio (F/R)*: Ratio of the number of operations in feedforward and recurrent pass.
Data reuse (R)**: Data reuse improvement compared to the dataflow without TWSO.

better results. Overall, this leads to a decrease in the optimal window size for WS, compared to OS. Nevertheless, time window optimization is still quite beneficial, even in the case of WS.

SaARSP architecture with TWSO achieves PE utilization of 71.9% for eight benchmarks, on average, as shown in Table 5. We observe a significant improvement compared to the PE utilization of 11.3% in the conventional approach since TWSO decreases the additional latency for iterative memory access, minimizing the stall cycle. Compared to DNN systolic array counterpart with 80% PE utilization [26], which does not incorporate recurrence, the result would be a good starting point to build efficient architectures for accelerating recurrent networks to handle the iterative memory access and complex spatiotemporal interactions. Furthermore, we observed that the number of operations in feedforward integration outnumbers the number of operations in recurrent integration, on average, where it especially dominates the computational overhead in case of popularly used Type-1 (uniform) topology. Still, data reuse in recurrent integration step with our TWSO technique delivers 4.0X improvement across benchmarks, on average. Detailed results are summarized in Table 5. The proposed SaARSP architecture and TWSO reduces the latency and energy dissipation of these benchmarks by up to 102X and 161X, respectively, over the baseline.

5.3.3 EDP Evaluation. The energy-delay-product (EDP) offers a balanced assessment of latency and energy dissipation. The normalized EDP of the OS and WS dataflows with and without TWSO for eight different networks is shown in Figure 11. We normalize each of the latency and energy of the baseline design to 100, so that the EDP of baseline design is normalized to 10,000.

Compared with the 1-D array baseline and our 2-D SaARSP architecture without TWSO, optimizing time window size on top of the SaARSP architecture present orders of magnitude of performance improvements across all benchmarks. In particular, the SaARSP architecture with TWSO delivers 11,000X and 58X EDP improvement, respectively, over the 1-D array baseline and SaARSP architecture without time window size optimization in the case of B3. In summary, TWSO significantly improves EDP across different benchmarks by optimizing temporal granularity. On average, decoupling scheme with time window size optimization introduces 4,000X EDP improvement across all eight benchmarks over the 1-D array baseline.

6 CONCLUSION

This work is motivated by the lack of an efficient architecture and dataflow for efficient acceleration of complex spatiotemporal dynamics arising in R-SNNs. To the best of our knowledge, the proposed architecture for **s**ystolic-**a**rray **a**cceleration of **r**ecurrent **sp**iking neural networks, dubbed SaARSP, presents the first systematic study of array-based hardware accelerator architectures for recurrent spiking neural networks.

One major challenge in accelerating R-SNNs stems from the tightly coupled data dependency in both time and space resulted from the recurrent connections. This challenge prevents direct exploration of time-domain parallelism and may severely degrade the overall performance due to poor data reuse patterns.

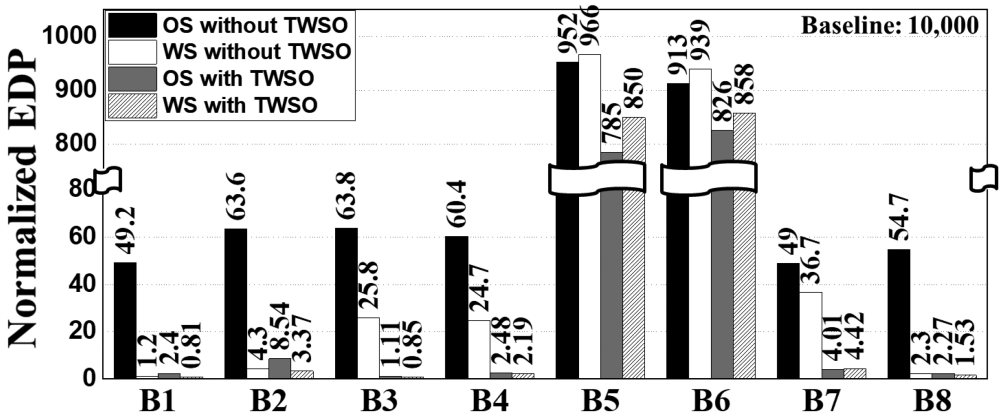

Fig. 11. Normalized EDP in recurrent layer of eight benchmarks with OS and WS. EDP values are normalized to the baseline result using 1-D array. The EDP of OS and WS with and without the time window optimization is shown.

The proposed SaARSP architecture is built upon a decoupling scheme and novel TWSO technique to enable the parallel acceleration of computation across multiple time points. This is achieved by cleverly decoupling the processes of feedforward and recurrent synaptic input integration, two dominant costs in processing recurrent network structures. We further boost the accelerator performance by optimizing the temporal granularity of the proposed decoupling and stationary dataflows in a layer-dependent manner. The SaARSP architecture can be applied to the acceleration of both feedforward and recurrent layers and hence is able to support a broad class of spiking neural network topologies.

Experimentally, the proposed SaARSP architecture and optimization scheme reduce the latency, energy dissipation, and energy-delay product (EDP) of the array accelerator by up to 102X and 161X, and 4,000X on average, respectively, over a conventional baseline for a comprehensive set of benchmark R-SNNs.

REFERENCES

[1] Kaveh Akbarzadeh-Sherbaf, Saeed Safari, and Abdol-Hossein Vahabie. 2020. A digital hardware implementation of spiking neural networks with binary force training. *Neurocomputing* 412 (2020), 129–142.

[2] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon nam, and B. Taba. 2015. Truenorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and DSystems* 34, 10 (2015), 1537–1557.

[3] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. 2018. Feature representations for neuromorphic audio spike streams. *Frontiers in Neuroscience* 12 (2018), 23.

[4] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. 2018. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*. 787–797.

[5] G. Bi and M. Poo. 1998. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 24 (Dec. 1998), 10464–10472.

[6] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. 2019. Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on EmergingTechnologies in Computing Systems (JETC)* 15, 2 (2019), 1–35.

[7] Yongqiang Cao, Yang Chen, and Deepak Khosla. 2015. Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vision* 113, 1 (May 2015), 54–66. https://doi.org/10.1007/s11263-014-0788-3

[8] N. Caporale and Y. Dan. 2008. Spike timing-dependent plasticity: A Hebbian learning rule. *Annual Review of Neuroscience* 31 (2008), 25–46.

[9] Andrew S. Cassidy, Paul Merolla, John V. Arthur, Steve K. Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore M. Wong, Vitaly Feldman, and A. Amir. 2013. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.

[10] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. 2012. Cacti-3DD: Architecture-Level Modeling for 3D Die-Stacked dram main memory. In *Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 33–38.

[11] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah) *(ASPLOS '14)*. Association for Computing Machinery, New York, 269–284. https://doi.org/10.1145/2541940.2541967

[12] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2017. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro* 37, 3 (2017), 12–21.

[13] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal ofSolid-State Circuits* 52, 1 (2016), 127–138.

[14] Kit Cheung, Simon R. Schultz, and Wayne Luk. 2012. A large-scale spiking neural network accelerator for FPGA systems. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 113–120.

[15] Sung-Gun Cho, Edith Beigné, and Zhengya Zhang. 2019. A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm CMOS. In *Proceedings of the 2019 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–4.

[16] Iulia M. Comsa, Thomas Fischbacher, Krzysztof Potempa, Andrea Gesmundo, Luca Versari, and Jyrki Alakuijala. 2020. Temporal coding in spiking neural networks with alpha synaptic function. In *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'20)* (May 2020). IEEE. https://doi.org/10.1109/icassp40776.2020.9053856

[17] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, and Y. Liao. 2018. LOIHI: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.

[18] Adele Diamond. 2013. Executive functions. *Annual Review of Psychology* 64 (2013), 135–168.

[19] Z Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. 2015. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104.

[20] John S. Garofolo. 1993. Timit acoustic phonetic continuous speech corpus. In *Linguistic Data Consortium,* (1993).

[21] Wulfram Gerstner and Werner M. Kistler. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.

[22] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedoings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 696–701.

[23] Shasha Guo, Lei Wang, Shuquan Wang, Yu Deng, Zhijie Yang, Shiming Li, Zhige Xie, and Qiang Dou. 2019. A systolic SNN inference accelerator and its co-optimized software framework. In *Proceedings of the 2019 Great Lakes Symposium on VLSI*. 63–68.

[24] Wenzhe Guo, Hasan Erdem Yantir, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama. 2021. Toward the optimal design and FPGA implementation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[25] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

[26] Nandan Kumar Jha, Shreyas Ravishankar, Sparsh Mittal, Arvind Kaushik, Dipan Mandal, and Mahesh Chandra. 2020. DRACO: Co-optimizing hardware utilization, and performance of DNNs on systolic accelerator. In *Proceedings of the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 574–579.

[27] Yingyezhe Jin, Wenrui Zhang, and Peng Li. 2018. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*.

[28] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier. 2018. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* 99 (Mar. 18), 56–67. https://doi.org/10.1016/j.neunet.2017.12.005

[29] Samir Kumar. 2013. Introducing Qualcomm zeroth processors: Brain-inspired computing. *Qualcomm ONQ Blog* (2013), 1–11.

[30] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Agshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 754–768.

[31] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[32] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. 2016. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience* 10 (2016), 508.
[33] Mark Liberman, Robert Amsler, kKen Church, Ed Fox, Carole Hafner, Judy Klavans, Mitch Marcus, Bob Mercer, Jan Pedersen, Paul Roossin, Don Walker, Susan Warwick, and Antonio Zampolli. 1991. TI 46-word IDC93s9. https://catalog.ldc.upenn.edu/docs/ldc93s9/ti46.readme.html.
[34] Richard Lyon. 1982. A computational model of filtering, detection, and compression in the cochlea. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 7 (*ICASSP'82*). IEEE, 1282–1285.
[35] Wolfgang Maass. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* 10, 9 (1997), 1659–1671.
[36] Wolfgang Maass, Thomas Natschläger, and Henry Markram. 2002. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation* 14, 11 (2002), 2531–2560.
[37] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri. 2018. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Transactions on Biomedical Circuits and Systems* 12, 1 (2018), 106–122.
[38] Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Mmmanuel Gaillardon. 2020. Spinalflow: an architecture and dataflow tailored for spiking neural networks. In *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 349–362.
[39] Daniel Neil and Shih-Chii Liu. 2014. Minitaur, An event-driven FPGA -based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 12 (2014), 2621–2628.
[40] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. 2015. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience* 9 (2015), 437.
[41] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs. 2017. Hierarchical address event routing for reconfigurable large-scale neuromorphic systems. *IEEE Transactions on Neural Networks and Learning Systems* 28, 10 (2017), 2408–2422.
[42] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal. 2013. Memory-centric accelerator design for convolutional neural networks. In *Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD)*. 13–19.
[43] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. 2015. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience* 9 (2015), 141. https://doi.org/10.3389/fnins.2015.00141
[44] Ulrich Rueckert. 2020. Update on brain-inspired systems. In *Proceedings of Nano-Chips 2030*. Springer, 387–403.
[45] Saunak Saha, Henry Duwe, and Joseph zZambreno. 2020. Cynapse: A low-power reconfigurable neural inference accelerator for spiking neural networks. *Journal of Signal Processing Systems* 92, 9 (2020), 907–929.
[46] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic CNN accelerator simulator. *arxiv preprint arxiv:1811.02883* (2018).
[47] Sumit Bam Shrestha and Garrick Orchard. 2018. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*. 1412–1421.
[48] Franyell Silfa, Gem Dot, Jose-Maria Arnau, and Antonio Gonzàlez. 2018. E-PUR: An energy-efficient processing unit for recurrent neural networks. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. 1–12.
[49] Pai-Yu Tan, Po-Yao Chuang, Yen-Ting Lin, Cheng-Wen Wu, and Juin-Ming Lu. 2020. A power-efficient binary-weight spiking neural network architecture for real-time object classification. *arxiv preprint arxiv:2003.06310* (2020).
[50] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. 2019. Deep learning in spiking neural networks. *Neural Networks* 111 (2019), 47–63.
[51] Matthijs van Keirsbilck, Alexander Keller, and Xiaodong Yang. 2019. Rethinking full connectivity in recurrent neural networks. *arxiv preprint arxiv:1905.12340* (2019).
[52] Shu-Quan Wang, Lei Wang, Yu Deng, Zhi-Jie Yang, Sha-Sha Guo, Zi-Yang Kang, Yu-Feng Guo, and Wei-Xia Xu. 2020 . SIES: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array. *Journal of Computer Science and Technology* 35 (2020), 475–489.
[53] Yujie wWu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. 2018. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience* 12 (2018), 331.
[54] Qiangfei Xia and Joshua J. Yang. 2019. Memristive crossbar arrays for brain-inspired computing. *Nature Materials* (2019), 309–323.
[55] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arxiv preprint arxiv:1708.07747* (2017).

[56] Wenrui Zhang and Peng Li. 2019. Spike-train level backpropagation for training deep recurrent spiking neural networks. In *Advances in Neural Information Processing Systems*. 7802–7813.

[57] Wenrui Zhang and Peng Li. 2020. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *arxiv preprint arxiv:2002.10085* (2020).

[58] Wenrui Zhang and Peng Li. 2021. Skip-connected self-recurrent spiking neural networks with joint intrinsic parameter and synaptic weight training. *Neural Computation* 33, 7 (2021), 1886–1913.

[59] Y. zhang, Peng Li, Yingyezhe Jin, and Yoonsuck Choe. 2015. A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE Transactions on Neural Networks and Learning systems* 26, 11 (Nov. 2015), 2635–2649.

Received 19 March 2021; revised 6 January 2022; accepted 9 January 2022