University of California
Santa Barbara

# Energy-Efficient Architecture and Dataflow Optimization for Spiking Neural Network Accelerators

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Jeong-Jun Lee

Committee in charge:

Professor Peng Li, Chair
Professor Spencer L. Smith
Professor Bongjin Kim
Professor Michael Beyeler

September 2022

The Dissertation of Jeong-Jun Lee is approved.

---

Professor Spencer L. Smith

---

Professor Bongjin Kim

---

Professor Michael Beyeler

---

Professor Peng Li, Committee Chair

August 2022

Energy-Efficient Architecture and Dataflow Optimization for Spiking Neural Network Accelerators



by

Jeong-Jun Lee

To my parents, family and friends for their great support

# Acknowledgements

First and foremost, I would like to express my sincere gratitude to Prof. Peng Li, for his great support and invaluable guidance during the past years. His expertise, vision and insights in the research fields deeply inspired me, which not only motivated me to have broader perspectives and to pursue higher goals, but also taught me some valuable life lessons. It was a great pleasure working with Prof. Li and I could not have imagined having a better advisor for my Ph.D. study.

I would also like to thank my committee members, Prof. Spencer Smith, Prof. Michael Beyeler, and Prof Bongjin Kim for their constructive and insightful feedback on my research. I truly appreciate your valuable suggestions and they are extremely helpful for this dissertation.

Being in our group with great colleagues was more than a great experience. I would like to appreciate Dr. Yu Liu, Dr. Wenrui Zhang, Dr. Changqing Xu, Yu Wang, Renqian Zhang, Umang Garg and Jianhao Chen, who worked with me on some research topics. I would also like to thank Dr. Hanbin Hu, Dr. Myung Seok Shim, Richard Boone, Zheng Ke, Karthik Somayaji, Zihu Wang, and Yuxuan Yin for their support.

Finally, I would like to thank my family for their love and support. My family is the most priceless gift and more than everything to me. Without your dedicated support, this journey would have been much harder.

## Funding Sources and Disclaimer

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC0021319, and the National Science Foundation under Award Numbers 1948201 and 2000851.



This dissertation is also supported by the UCSB ECE Dissertation Fellowship for Summer 2022.

# Curriculum Vitæ

Jeong-Jun Lee

## Education

| | |
|---|---|
| Aug 2022 | Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara. |
| Feb 2018 | M.S in Electrical and Computer Engineering, Seoul National University. |
| Feb 2016 | B.S in Electrical and Computer Engineering, Seoul National University. |

## Publications

Lee, Jeong-Jun, and Peng Li. "Systolic Array Acceleration of Spiking Neural Networks with Application-Independent Split-Time Temporal Coding." 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022. *submitted

Lee, Jeong-Jun, Wenrui Zhang, and Peng Li. "Parallel Time Batching: Systolic-Array Acceleration of Sparse Spiking Neural Computation." In 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 317-330. IEEE, 2022.

Lee, Jeong-Jun, Wenrui Zhang, Yuan Xie, and Peng Li. "SaARSP: An Architecture for Systolic-Array Acceleration of Recurrent Spiking Neural Networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 2022.

Lee, Jeong-Jun, Jianhao Chen, Wenrui Zhang, and Peng Li. "Systolic-Array Spiking Neural Accelerators with Dynamic Heterogeneous Voltage Regulation." In 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1-7. IEEE, 2021.

Lee, Jeong-Jun, and Peng Li. "Reconfigurable dataflow optimization for spatiotemporal spiking neural computation on systolic array accelerators." In 2020 IEEE 38th International Conference on Computer Design (ICCD), pp. 57-64. IEEE, 2020.

Lee, Jeongjun, Renqian Zhang, Wenrui Zhang, Yu Liu, and Peng Li. "Spike-train level direct feedback alignment: sidestepping backpropagation for on-chip training of spiking neural nets." Frontiers in Neuroscience 14 2020: 143.

Hwang, Sungmin*, Jeong-Jun Lee*, Min-Woo Kwon, Myung-Hyun Baek, Taejin Jang, Jeesoo Chang, Jong-Ho Lee, and Byung-Gook Park. "Analog Complementary Metal–Oxide–Semiconductor Integrate-and-Fire Neuron Circuit for Overflow Retaining in Hardware Spiking Neural Networks." Journal of nanoscience and nanotechnology 20, no. 5 (2020): 3117-3122. doi: 10.1166/jnn.2020.17390 *equally contributed

Lee, Jeong-Jun, Jungjin Park, Min-Woo Kwon, Sungmin Hwang, Hyungjin Kim, and Byung-Gook Park. "Integrated neuron circuit for implementing neuromorphic system with synaptic device." Solid-State Electronics 140 (2018): 34-40.

Lee, Jeong-Jun, Min-Woo Kwon, Hyungjin Kim, Sungmin Hwang, and Byung-Gook Park. "Implementation of inhibitory operation in neuromorphic system." In 2017 Silicon Nanoelectronics Workshop (SNW), pp. 113-114. IEEE, 2017.

Wang, Yu, Jeong-Jun Lee, Yu Ding, and Peng Li. "A scalable FPGA engine for parallel acceleration of singular value decomposition." In 2020 21st International Symposium on Quality Electronic Design (ISQED), pp. 370-376. IEEE, 2020.

Baek, Myung-Hyun, Taejin Jang, Hyungjin Kim, Jungjin Park, Min-Woo Kwon, Sungmin Hwang, Suhyeon Kim, Jeong-Jun Lee, and Byung-Gook Park. "Grain boundary induced short-term memory effect in fully depleted thin-polysilicon devices." Japanese Journal of Applied Physics 58, no. 10 (2019): 101004.

Hwang, Sungmin, Hyungjin Kim, Jungjin Park, Min-Woo Kwon, Myung-Hyun Baek, Jeong-Jun Lee, and Byung-Gook Park. "System-level simulation of hardware spiking neural network based on synaptic transistors and I&F neuron circuits." IEEE Electron Device Letters 39, no. 9 (2018): 1441-1444.

Park, Jungjin, Min-Woo Kwon, Hyungjin Kim, Sungmin Hwang, Jeong-Jun Lee, and Byung-Gook Park. "Compact neuromorphic system with four-terminal Si-based synaptic devices for spiking neural networks." IEEE Transactions on Electron Devices 64, no. 5 (2017): 2438-2444.

## Abstract

Energy-Efficient Architecture and Dataflow Optimization for Spiking Neural Network Accelerators

by

Jeong-Jun Lee

Spiking neural networks (SNNs) offer a promising biologically-plausible computing model and lend themselves to ultra-low-power event-driven processing on neuromorphic processors. Compared with the conventional artificial neural networks, SNNs are well-suited for processing complex spatiotemporal data. In this dissertation, we aim to address key difficulties in accelerating SNNs: developing bio-plausible and hardware friendly algorithm, efficient processing of the added temporal dimension, and handling unstructured sparsity emergent in both space and time.

First, training SNNs to reach the same performances of conventional deep artificial neural networks (ANNs), particularly with error backpropagation (BP) algorithms, poses a significant challenge due to inherent complex dynamics and non-differentiable spike activities of spiking neurons. In this dissertation, we present the first study on realizing competitive spike-train level backpropagation (BP) like algorithms to enable on-chip training of SNNs. We propose a novel spike-train level direct feedback alignment (ST-DFA) algorithm, which is much more bio-plausible and hardware friendly than BP. Algorithm and hardware co-optimization and efficient online neural signal computation are explored for on-chip implementation of ST-DFA. On the Xilinx ZC706 FPGA board, the proposed hardware-efficient ST-DFA shows excellent performance vs. overhead trade-offs for real-world speech and image classification applications.

Despite its significance, dataflow optimization of spiking neural accelerator architec-

tures has not been extensively studied. Recognizing the need for efficient processing of complex spatiotemporal data while considering the all-or-none nature of spiking activities, we propose holistic reconfigurable dataflow optimization for systolic array acceleration of spiking convolutional networks (S-CNNs). A novel scheme is introduced for parallel acceleration of computation across multiple time points, which further allows for systemic optimization of variable tiling for a large performance and efficiency gains. Also, we pack multiple time points into a single time window ($TW$) and process the computations induced by active synaptic inputs falling under several $TW$s in parallel, leading to the proposed parallel time batching. It allows weight reuse across multiple time points and enhances the utilization of the systolic array with reduced idling of processing elements, overcoming the irregularity of sparse firing activities. We optimize the granularity of time-domain processing, i.e., the $TW$ size, which significantly impacts the data reuse and utilization.

Lastly, we propose a novel technique and architecture that allow the exploitation of temporal information compression with structured sparsity and parallelism across time, and significantly improves data movement on a systolic array. We split a full range of temporal domain into several time windows (TWs) where a TW packs multiple time points, and encode the temporal information in each TW with Split-Time Temporal coding (STT) by limiting the number of spikes within a TW up to one. STT enables sparsification and structurization of irregular firing activities and dramatically reduces computational overhead while delivering competitive classification accuracy without a huge drop. To further improve the data reuse, we propose an Integration Through Time (ITT) technique that processes integration steps across different TWs in parallel with a systolic array.

In this dissertation, we provide unique and powerful solutions for the efficient acceleration of the spiking models with various datasets.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Neuromorphic Computing Systems

Conventional non-spiking artificial neural network models, or simply ANNs, employ only rate coding where continuous-valued signals resulted from activation functions such as sigmoid and rectified linear unit (ReLU) correspond to average firing rates. On the other hand, spiking neural networks (SNNs) more closely resemble biological neurons, explicitly model all-or-none firing spikes across both space and time, and can leverage a rich family of rate and temporal codes for complex spatiotemporal information processing.

Recent studies reported competitive performances for various image and speech tasks with biologically inspired [1, 2] and backpropagation based [3, 4] SNN training methods. With great potentials in ultra-low power event-driven learning leveraging the spatiotemproal dynamics of SNNs [5], neuromorphic processors have gathered significant interest in both academia and industry, resulting in well-known neuromorphic chips including IBM's TrueNorth [6] and Intel's Loihi [7].

## 1.2 Spiking Neural Network Algorithms

Despite the recent progresses in SNNs and neuromorphic processor designs, fully leveraging the theoretical computing advantages of SNNs over traditional artificial neural networks (ANNs) [5] to achieve the state-of-the-art performance for real-world applications remains challenging. One chief difficulty here lies in training of SNNs in terms of achievable performance and computational complexity.

Inspired by the success of error backpropagation (BP) and its variants such as stochastic gradient decent in training conventional ANNs [8], various SNN BP methods have emerged, aiming at attaining the same level of performance [9, 10, 11, 4]. The major challenges in BP training of SNNs stem from the non-differentiablity of spike events and temporal dynamics that prevent straightforward derivative computation. SpikeProp [9] is the first BP algorithm to train SNNs by BP. However, SpikeProp is limited to single-spike training for learning simple functions like XOR. [10] proposes a BP algorithm which differentiates neuron's membrane potential instead of discrete output spikes. [11] improves [10] by capturing temporal effects with backpropapogation through time (BPTT) [12]. However, the error gradient is still computed by differentiating the membrane potential, leading to inconsistency w.r.t the rate-coded loss function. More recently, [4] proposes a hybrid macro/micro level backpropagation (HM2-BP) algorithm for training multi-layer

SNNs, which addresses the aforementioned issues. HM2-BP precisely captures the temporal behavior of the SNN at the microscopic level and directly computes the gradient of the rate-coded loss function w.r.t tunable parameters. As a result, HM2-BP demonstrates the state-of-the art learning performances on widely adopted SNN benchmarks such as MNIST [13] and Neuromorphic-MNIST (N-MNIST) [14], outperforming all other existing BP algorithms based on the leaky integrate-and-fire model.

While achieving excellent results, the aforementioned SNN BP algorithms are hampered by several limitations. The error signal is propagated backward layer by layer through weights symmetric to the feed-forward weights. This is considered not biologically-plausible. Furthermore, BP algorithms involve complex layer-by-layer backward computations, which is expensive to implement on-chip and introduces high training latency. For instance, while HM2-BP improves the scalability of BPTT [11] by operating on the spike-train level, i.e. application of BP does not discretize time, it still involves complex computations and its latency in the backward phase is proportional to network depth.

## 1.3 Spiking Neural Network Accelerators

Spiking neural networks (SNNs) more closely resemble biological neurons, explicitly model all-or-none firing spikes across both space and time, and can leverage a rich family of rate and temporal codes for complex spatiotemporal information processing. Recent studies reported competitive performances for various image and speech tasks with biologically inspired [1, 2] and backpropagation based [3, 4] SNN training methods. With great potentials in ultra-low power event-driven learning leveraging the spatiotemproal dynamics of SNNs [5], neuromorphic processors have gathered significant interest in both academia and industry, resulting in well-known neuromorphic chips including IBM's TrueNorth [6] and Intel's Loihi [7].

Nevertheless, hardware acceleration of spike-based models is complicated by temporal computation and sparse spiking activities in both space and time, two new challenges that are absent in accelerators of non-spiking networks such as DNNs. The added temporal dimension is fundamental to SNNs but introduces difficulties in managing compute and data movement. Furthermore, biological brains and engineered SNN models often exhibit a great deal of firing activity sparsity across both space and time, manifesting their promising efficiency. The sparse spiking activities of a well-trained SNN may vary from neurons to neurons, and from time points to time points. To fully explore the benefits of SNNs, one must address the challenges brought by irregular patterns of spatial and temporal sparsity.

Compared to the large body of work on DNN accelerators, e.g., [15, 16, 17, 18, 19], much less research has been devoted to SNN hardware accelerator architectures [20, 21, 22, 23, 24]. The two best-known industrial neuromorphic chips, IBM's TrueNorth [6] and Intel's Loihi [7], are based on a many-core architecture, comprising neuro-synaptic cores with an asynchronous mesh for core-to-core communication. Each neuromorphic core emulates a certain number of spiking neurons in a time-sequential manner. While both architectures target large-scale spiking neural computations with low power, there exist two primary disadvantages in these two designs: 1) lack of parallelism in each core: the computations associated with different spiking neurons are executed sequentially, one neuron at a time, and from time points to time points; and 2) assumption of large core memory: as opposed to many practical cases, it is assumed that all weights of the network are fully stored on-chip, and hence efficient dataflows maximizing the reuse of weight data are not targeted. These issues limit the achievable throughput and/or do not well support SNN acceleration on resource-constrained hardware like ones for edge computing.

The recent SNN architecture SpinalFlow explores a novel compressed, time-stamped,

and sorted spike input/output representation [20]. The main drawback of SpinalFlow is that it only targets the class of temporally-coded spiking neuronal models in which each neuron fires at most once, which is a highly restrictive type and has limited accuracy for challenging learning tasks [25, 26]. While the smart exploration of such extreme temporal sparsity leads to large latency and energy efficiency benefits, SpinalFlow is not applicable to broader classes of SNNs employing rate and other types of temporal codes or a combination of thereof for high-accuracy decision making. Since the maximum firing count for each neuron is one, the structured sparse firing activities are handled as chronologically sorted inputs with a dearth of parallel acceleration through time.

## 1.4 Outline

The summary of the rest of this dissertation is as follows: In Chapter 2, we provide a brief background on unique characteristics of SNNs, basic operations in spiking neuron models, datasets and systolic arrays. In Chapter 3, we perform algorithm-hardware co-optimization and demonstrate the first on-chip hardware realization of Spike-Train level Direct Feedback Alignment (ST-DFA) for SNNs with significantly improved accelerator performance compared to conventional Back-Propagation (BP). In Chapter 4, we propose holistic reconfigurable dataflow optimization for systolic array acceleration of spiking convolutional networks (S-CNNs), and introduce SNN dataflow simulator to support systemic design space exploration. In Chapter 5, we introduce a novel technique for parallel acceleration in both space and time based on simultaneous processing of multiple time batches with a temporal granularity defined by the Time Window (TW) size, along with Spatiotemporally-non-overlapping Spiking Activity Packing (StSAP). In Chapter 6, we explore the proposed parallel acceleration technique to decouple the processing of feedforward synaptic connections from that of recurrent connections, for efficient acceler-

ation of complex spatiotemporal dynamics arising in R-SNNs. In Chapter 7, we propose a novel, application independent solution called Split-Time Temporal (STT) coding that allows the exploitation of temporal information compression with structured sparsity and parallelism across time, which is further supported by Integration-Through-Time (ITT). Chapter 8 summarizes this dissertation and includes discussions on potential future work.

# Chapter 2

# Background

## 2.1 Unique Characteristics of SNNs

Compared with non-spiking ANNs, the most distinctive features of SNNs are temporal data processing and data representation. All data types in ANNs, e.g., input/output feature maps (IFmap/OFmap) and filters data in widely adopted convolutional neural networks (CNNs), are multi-bit. The most commonly used data representations in non-spiking ANN hardware accelerators are based on 8- to 16-bit precision [15, 19, 27, 28], which may be further compressed using techniques such as weight quantization [29, 30]. On the other hand, input/output activations of a spiking layer are binary due to the all-or-none characteristics of spiking neural firing characteristics, which can be more compactly stored than multi-bit partial sum data. This disparity in data representations can be explored in dataflow optimization [21, 20].

While integration and activation steps in ANNs exclude temporal information, a spiking neuron integrates its inputs over time, as shown in Fig. 2.1(b). The spatiotemporal information processing in SNNs empower various models and applications [3, 31, 32]. However, the added temporal dimension causes intertwined spatiotemporal interactions,

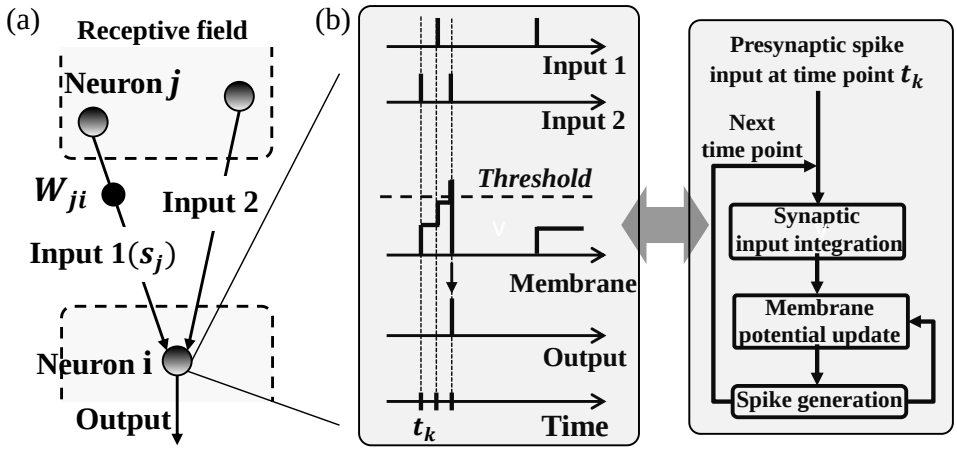


Figure 2.1: (a): Layer operation in SNNs. (b): Main steps of spatiotemporal operation in a spiking neuron.

rendering SNN hardware accelerators to confront complex data movement/computation, which we address in later sections.

## 2.2 Spiking Neural Network Operations

### 2.2.1 Spiking Neurons

Common to literally all spiking neural models, operations in one spiking neuron comprise three main steps at each time point $t_i$, where time point is a minimum unit of time in SNNs: 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 Fig. 2.1 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

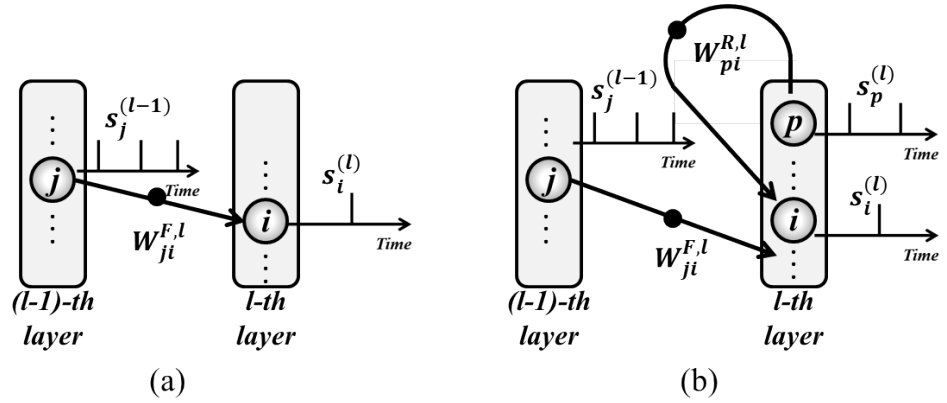


Figure 2.2: (a): Schematic for feedforward layer operation in SNN. (b): Schematic for recurrent layer operation in R-SNN.

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 Fig. 2.1(b). The same process repeats for all time points involved in a given spike-based task.

### 2.2.2 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 dissertation considers wide range of network types, including general the most general (deep) multi-layer feedforward (fully-connected) spiking neural network (SNN), convolutional spiking neural networks (S-CNN or spiking-CNN), and recurrent spiking neural network (R-SNN) architecture, which comprises multiple fully-connected or convolutional or recurrent layers or a combination of thereof.

### Feedforward spiking layers

Conventional and the most natural approach for temporal data processing is to perform operations time point by time point in a sequential manner, for all time points in time stride. In feedforward spiking layers, operations in a single spiking neuron consist of three steps at each time point $t_k$:

**Step 1:** Synaptic input integration at $t_k$:

$$\vec{p}^{Post}[t_k] = \mathbf{W}_{Post,Pre} \times \vec{s}^{Pre}[t_k] \tag{2.1}$$

**Step 2:** Membrane potential update at $t_k$:

$$\vec{v}^{Post}[t_k] = \vec{v}^{Post}[t_{k-1}] + \vec{p}^{Post}[t_k] - V_{leak}^{Post} \tag{2.2}$$

**Step 3:** Conditional spike output generation at $t_k$:

$$\vec{s}^{Post}[t_k] = \boldsymbol{f}(\vec{v}^{Post}[t_k]) \tag{2.3}$$

$$\boldsymbol{f}(v_i^{Post}[t_k]) = \begin{cases} 1, & \text{if } v_i^{Post}[t_k] \geq V_{th}^{Post} \rightarrow v_i^{Post}[t_k] = 0 \\ 0 & \text{else} \rightarrow v_i^{Post}[t_k] = v_i^{Post}[t_k] \end{cases} \tag{2.4}$$

where the *Post* and *Pre* denote the pre-synaptic layer and the post-synaptic layer, and $i$ represents the neuron indices in the post-synaptic layer. $\vec{p}^{Post}[t_k]$, $\vec{v}^{Post}[t_k]$, and $\vec{s}^{Post}[t_k]$ are vectors, representing the integrated partial sum of the spike inputs from the pre-synaptic layer, membrane potential and spike output of the neurons in the post-synaptic layer at time $t_k$, respectively. $\mathbf{W}_{Post,Pre}$ is the matrix of the feedforward synaptic weights between pre- and post-synaptic layers, $V_{th}$ and $V_{leak}$ are the firing threshold and leaky

parameter in post-synaptic layer, respectively. $\boldsymbol{f}$ is a non-linear, all-or-non activation function with a given $V_{th}$. In the above steps, the synaptic input integration (**Step 1**) incurs matrix-vector multiplication and takes place at each time point, comprising the dominant complexity of SNN acceleration.

Importantly, the above steps are repeated at each time point, across all time points in time stride. The above steps present fundamental operations in any feedforward layers including fully-connected and convolutional layers.

### Recurrent Spiking Layers

Processing neural computations of a recurrent layer in SNNs follow the same three steps in the feedforward layer with additional synaptic inputs. In a recurrent layer, lateral recurrent inputs are also considered in addition to the feedforward input integration (**Step 1**) in (2.1):

**Step 1*:** Feedforward synaptic input integration at $t_k$:

$$\vec{p}_F^{Post}[t_k] = \mathbf{W}_{Post,Pre} \times \vec{s}^{Pre}[t_k] \tag{2.5}$$

$$\vec{p}_R^{Post}[t_k] = \mathbf{W}_{Post,Post} \times \vec{s}^{Post}[t_{k-1}] \tag{2.6}$$

$$\vec{p}^{Post}[t_k] = \vec{p}_F^{Post}[t_k] + \vec{p}_R^{Post}[t_k] \tag{2.7}$$

where $\vec{p}_F^{Post}[t_k]$, $\vec{p}_R^{Post}[t_k]$ and $\vec{p}^{Post}[t_k]$ are vectors, representing the partial sum of the feedforward input integration, recurrent input integration, and fully-integrated partial sum in the post-synaptic layer at time $t_k$, respectively. $\mathbf{W}_{Post,Post}$ is the matrix of the

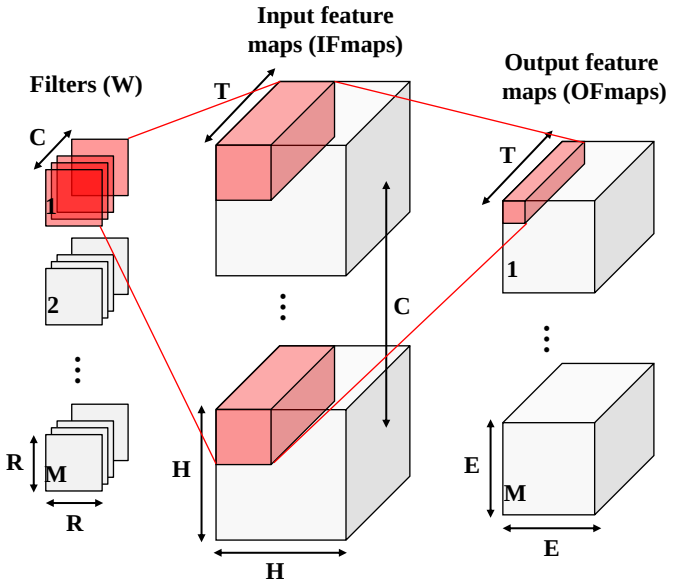


Figure 2.3: Computation of convolution layer in S-CNNs.

Table 2.1: Shape parameters of a CONV layer in S-CNNs

| Shape Parameter | Description |
|---|---|
| H / H | ifmap width / height |
| E / E | ofmap width / height |
| R / R | filter width / height |
| C | # of ifmap/filter channels |
| M | # of ofmap channels |
| T | # of time steps |

recurrent synaptic weights of the post-synaptic (recurrent) layer.

### Convolutional Spiking Layers

The fundamental operations of a single spiking neuron in spiking-CNNs (S-CNNs) follow the aforementioned three main steps (2.1)∼(2.4) where the computation involves multiple filters:

**At a given time-point $t_k$,**

**Step 1:** Integration of receptive field synaptic inputs at $t_k$:

$$\mathbf{P}[m][x][y][t_k] = \sum_{c=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{R-1} \mathbf{W}[m][c][i][j] \times \mathbf{I}[c][Ux+i][Uy+j][t_k] \tag{2.8}$$

**Step 2:** Membrane potential update:

$$\mathbf{V}[m][x][y][t_k] = \mathbf{V}[m][x][y][t_{k-1}] + \mathbf{P}[m][x][y][t_k] \tag{2.9}$$

**Step 3:** Conditional spike output generation:

$$\mathbf{O}[m][x][y][t_k] = \begin{cases} 1, & \text{if } \mathbf{V}[m][x][y][t_k] \geq V_{th}^{O} : \mathbf{V}[m][x][y][t_k] = 0 \\ 0 & \text{else} : \mathbf{V}[m][x][y][t_k] = \mathbf{V}[m][x][y][t_k] \end{cases} \tag{2.10}$$

$0 \leq x,y < E$, $E=(H-R+U)/U$, $0 \leq c < C$, $0 \leq m < M$, $0 \leq t < T$

**Move onto the next time-point** ($t_{k+1}$), and repeat the above three steps.

where $\mathbf{P}$, $\mathbf{V}$, $\mathbf{O}$, $\mathbf{I}$ and $\mathbf{W}$ are the matrices of the partial sums (Psums), membrane potentials, output feature maps (OFmaps), input feature maps (IFmaps) and filters, respectively. $\mathbf{P}[m][x][y][t_k]$ is the partial sum of the neuron at position $(x, y)$ and in output channel $m$ of the OFmap at time $t_k$. Other matrices are defined similarly. U is a given stride size, T is the number of processing time steps, and all the other shape parameters are listed and illustrated in Table 2.1 and Fig. 2.3. (2.8)$\sim$(2.10) correspond to each of the three steps discussed in (2.1)$\sim$(2.4).

## 2.3 Datasets

We adopt various types of datasets for the evalutation including images/speeches, neuromorphic images/speeches, and neuromorphic videos.

### MNIST

The MNIST handwritten digit dataset [13] contains 60k training and 10k testing samples, each of which is a $28 \times 28$ grayscale image. Each pixel value of the MNIST image is converted into a spike train using Poisson sampling and the probability of spike generation is proportional to the pixel intensity. Thus, inputs are encoded into a 2D $784 \times N_t$ matrix where $N_t$ is the simulation time steps. Especially, due to the limited hardware resources available on the Xilinx Zynq ZC706 board, we crop each image to include only the $14 \times 14$ pixels around the center for FPGA evaluation in chapter 3. For all other experiments, each pixel value of an MNIST image is converted into a real-valued input current.

### Fashion-MNIST

The Fashion-MNIST dataset [33] is a MNIST-like, but much more difficult, clothing classification dataset which contains 28x28 grey-scale images of clothing items. Fashion-MNIST dataset contains 60k training examples and 10k testing examples with 10 classes.

### CIFAR10-DVS

The CIFAR-10 dataset [34] consists of 60,000 $32 \times 32$ color images in 10 classes, with 6,000 images per class. The CIFAR10-DVS [35] dataset is a neuromorphic version of CIFAR10, using a dynamic vision sensor (DVS) camera, an event-stream dataset for object classification containing 10k examples with an event-based sensor, whose resolution is $128 \times 128$ pixels.

### TI46-Alpha

TI46 speech corpus [36] contains spoken English alphabets/digits audios from 16 speakers. In this dissertation, we only use alphabets of the full TI46 speech corpus. For the rest of this dissertation, we call this as TI46-Alpha or Ti46, where TI46-Alpha consists of 4142 and 6628 spoken English examples in 26 classes for training and testing, respectively. The continuous temporal speech waveforms are preprocessed by Lyon's ear model [37] with the sample rate of 12.5 kHz where each sample is encoded into 78 channels.

### N-MNIST

Similar to CIFAR10-DVS, the N-MNIST dataset [14] is a neuromorphic version of the MNIST dataset using the dynamic vision sensor (DVS) [38] in front of static digit images on a computer monitor. Changes of pixel intensity at each location are encoded as spike trains where image size is $34 \times 34$ rather than $28 \times 28$ due to the relative shifts of each image. As in [39], we reduce the time resolution into a few hundred time steps since original dataset requires 300000 time steps.

### N-TIDIGITS

TIDIGITS [40] is a speech dataset where each sample has 64 input channels, processed by CochleaAMS1b sensor and takes about $0.9s$. The N-TIDIGITS [41] is a neuromorphic version of TIDIGITS, where connected-digit sequences are not considered and we only consider single-digit samples with 2,475 samples for training and 2.475 samples for testing. Similar to N-MNIST, we reduce the temporal resolution by compressing 1 $us$ to 3 $ms$, forming 300 time steps.

**DVS-Gesture**

The DVS-Gesture dataset [42] consists of event streams of hand/arm gestures. Using the dynamic vision sensor (DVS) camera, the input frame is $128 \times 128$ pixels with two channels where we follow the same preprocessing procedures in [43]. While each action (sample) lasts for about 6 $s$, we only consider the first 1.5 $s$ of action video as in [39] and compress the temporal resolution to 5 $ms$ which produced 300 time steps for each sample.

## 2.4 Systolic Array

Systolic array architectures offer efficient parallel processing with high spatiotemporal locality and compute density. In many prior works, a tightly coupled 2-D systolic array has been adopted for CNN accelerations with clear advantages [44, 19, 45, 18]. Systolic arrays propagate data horizontally and vertically, i.e., from left to right and from top to bottom through all processing elements (PEs) in a globally synchronized manner, hence naturally exploit high locality and compute density. For example, a unidirectional link is utilized in the vertical data propagation to allow each PE to receive the input from its upstream neighbor, perform the computation and store the results, and continue to pass the data to its downstream neighbor. Furthermore, data are fed from edges of the array to provide sufficient data distribution bandwidth. The above properties result in a streamlined accelerator platform for managing data fetching without requiring complicated inter-PE communication. With the advantages in terms of complexity, distribution bandwidth, locality, and compute density, systolic arrays are adopted for efficient acceleration of spiking computation in chapter $4 \sim 7$, which well-aligns with parallel-acceleration techniques introduced in this dissertation.

# Chapter 3

# Spiking Neural Processor with Direct Feedback Alignment

In this chapter, we present the first study on realizing competitive spike-train level back-propagation (BP) like algorithms to enable on-chip training of SNNs. We propose a novel spike-train level direct feedback alignment (ST-DFA) algorithm, which is much more bio-plausible and hardware friendly than BP. Algorithm and hardware co-optimization and efficient online neural signal computation are explored for on-chip implementation of ST-DFA. On the Xilinx ZC706 FPGA board, the proposed hardware-efficient ST-DFA shows excellent performance vs. overhead tradeoffs for real-world speech and image classification applications. SNN neural processors with on-chip ST-DFA training show competitive classification accuracy of 96.27% for the MNIST dataset with 4× input resolution reduction and 84.88% for the challenging 16-speaker TI46 speech corpus, respectively. Compared to the hardware implementation of the state-of-the-art BP algorithm HM2-BP, the design of the proposed ST-DFA reduces functional resources by 76.7% and backward training latency by 31.6% while gracefully trading off classification performance.

This work aims to answer the following questions: 1) Can biologically plausible mech-

anisms be developed to sidestep complex BP algorithms while delivering competitive performance? 2) Can such mechanisms be leveraged for efficient on-chip training of multi-layer SNNs?

## 3.1 Direct Feedback Alignment (DFA)

### 3.1.1 Direct Feedback Alignment

Backpropagation (BP) has been widely applied to train neural networks. It is based upon computing a global error at the output layer and then propagating the error signal to hidden neurons layer by layer. During this process, the errors of a preceding layer are multiplied with a weight matrix that is completely symmetric to the one for the feed-forward connections. This fact is not considered biologically plausible. A recent discover called Feedback Alignment (FA) [46] demonstrates that the weights used for propagating the error layer by layer need not be symmetric to the weights used for forward propagation to achieve good performance. The feedback weight matrix can be randomly generated and then stay unchanged since the network can learn how to make feedback useful through training. [47] applies FA for training SNNs.

A more disruptive approach called Direct Feedback Alignment (DFA) is proposed in DNNs [48]. DFA is compared with BP in Fig. 3.1. Unlike propagating the error back layer by layer in BP and FA, DFA feeds back the error through fixed random feedback connections directly from the output layer to each hidden layer, eliminating the need for layer-by-layer error backpropogation or feedback. DFA is considered more biologically plausible because the error is generated almost completely local with no long backpropagation/feed back train and symmetric weights are not required. [48] shows that for conventional multi-layer ANNs like DNNs, the use of DFA can achieve competitive

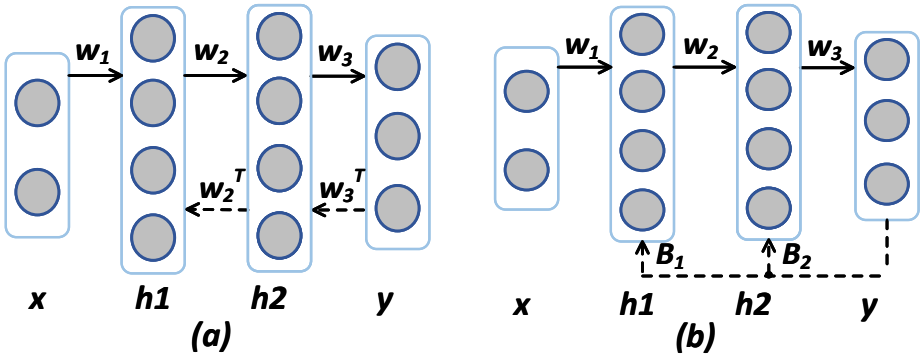


Figure 3.1: (a) Backpropagation (BP) vs. (b) direct feedback alignment (DFA). Solid arrows indicate feedforward paths and dashed arrows indicate feedback paths. The feedback matrices $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ need not be symmetric to $\boldsymbol{W}_2$ or $\boldsymbol{W}_3$.

results with insignificant performance drops when compared with the state-of-the-art BP methods.

In this paper, we extend the DFA for conventional ANNs [48] for SNNs. To the best of our knowledge, this is the first work applying DFA to SNNs. Furthermore, our DFA approach, dubbed ST-DFA, operates on the spike-train level, hence offering improved scalability in both space (network depth) and time.

### 3.1.2 Spike-train Level Post-synaptic Potential

Before describing the proposed ST-DFA in Section 3.2, we present the concept of Spike-train Level Post-synaptic Potential (S-PSP) that is behind the spike-train level computation of ST-DFA.

The widely adopted leaky integrate-and-fire (LIF) model for spiking neurons is given by [49]:

$$\tau_m \frac{u_i(t)}{dt} = -u_i(t) + R\ \alpha_i(t), \tag{3.1}$$

with

$$\tau_s \frac{\alpha_i(t)}{dt} = -\alpha_i(t) + \sum_j w_{ij} \sum_{t_j^{(f)}} D\left(t - t_j^{(f)}\right), \tag{3.2}$$

where $u_i(t)$ is the membrane potential of the neuron $i$, $\alpha_i(t)$ is the first order synaptic model with time constant $\tau_s$, and $\tau_m$ is the time constant of membrane potential with value $\tau_m = RC$. $R$ and $C$ are the effective leaky resistance and effective membra ne capacitance. $w_{ij}$ is the weight of the synapse from the pre-synaptic neuron $j$ to the neuron $i$. $t_j^{(f)}$ denotes a particular firing time of the neuron $j$. $D(t)$ is the Dirac delta function. R is set to 1 since it can be absorbed into synaptic weights.

Integrating (3.1) and (3.2) gives the spike response model (SRM) [4]:

$$u_i(t) = \sum_j w_{ij} \sum_{t_j^{(f)}} \epsilon\left(t - \hat{t}_i^{(f)}, t - t_j^{(f)}\right), \tag{3.3}$$

where $\hat{t}_i^{(f)}$ denotes the last firing time of the neuron $i$. $\epsilon(s,t)$ specifies the normalized time course of the *post-synaptic potential* evoked by a single firing spike of the pre-synaptic neuron:

$$\epsilon(s,t) = \frac{1}{C} \int_0^s \exp\left(-\frac{t'}{\tau_m}\right) \alpha_i\left(t - t'\right) \, \mathrm{d}t'. \tag{3.4}$$

Integrating (3.4) gives:

$$\epsilon(s,t) = \frac{e^{(-\max(t-s,0)/\tau_s)}}{1 - \frac{\tau_s}{\tau_m}} \quad \left[e^{\left(-\frac{\min(s,t)}{\tau_m}\right)} - e^{\left(-\frac{\min(s,t)}{\tau_s}\right)}\right] H(s)H(t), \tag{3.5}$$

where $H(t)$ is the Heaviside step function.

The sum of the (normalized) post-synaptic potential of the neuron $i$ evaluated right before all the neuron $i$'s firing times evoked by the spike train of the pre-synaptic neuron $j$ defines the (normalized) **spike-train level post-synaptic potential** (**S-PSP**) $e_{i|j}$,

which is given by:

$$e_{i|j} = \sum_{t_i^{(f)}} \sum_{t_j^{(f)}} \epsilon(t_i^{(f)} - \hat{t}_i^{(f)}, t_i^{(f)} - t_j^{(f)}). \tag{3.6}$$

S-PSP specifies the aggregated effect of the spike train of the pre-synaptic neuron $j$ on the membrane potential of the post-synaptic neuron $i$, providing a basis for relating firing counts to spike events.

Summing the weighted S-PSPs from all pre-synaptic neurons of the neuron $i$ gives the **total post-synaptic potential** (**T-PSP**) $a_i$, which is directly correlated to the neuron $i$'s firing count $o_i$ through the firing threshold voltage $\nu$:

$$a_i = \sum_j w_{ij}\ e_{i|j}. \qquad o_i = g(a_i) \approx \frac{a_i}{\nu} \tag{3.7}$$

## 3.2 Spike-Train Level DFA (ST-DFA)

### 3.2.1 Proposed ST-DFA Algorithm

For a conventional (non-spiking) ANN, the squared error for one training example can be defined at the output layer by:

$$E = \frac{1}{2}||\boldsymbol{o} - \boldsymbol{y}||_2^2, \tag{3.8}$$

where $\boldsymbol{y}$ and $\boldsymbol{o}$ are vectors specifying the desired output (label) and the actual output, respectively. The output $o_i$ of each neuron $i$ is determined by the activation function $\phi_i$: $o_i = \phi_i(\sum_j w_{ij}x_j)$, where $x_j$ is the input value from the presynaptic neuron $j$ and $w_{ij}$ is the weight between the neurons $j$ and $i$.

The well-known BP algorithm for an ANN [50], which is ubiquitously used in deep

learning, is:

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}^k} = \eta \delta_i^k \phi_j^{k-1}$$
$$\delta_i^k = \begin{cases} o_i - y_i & \text{for output layer,} \\ \phi_i'^{k+1} \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li}^{k+1} & \text{for hidden layers,} \end{cases} \tag{3.9}$$

where $\eta$ is the learning rate, $\delta_i^k$ the error for the $i_{th}$ neuron of the $k_{th}$ layer, $r^k$ the number of neurons in the $k_{th}$ layer.

It has been demonstrated recently that training SNNs using BP with respect to a rate-coded loss function has produced highly competitive performances [10, 11, 4]. Rate-coded loss functions are also adopted for our ST-DFA. Different from BP, the proposed ST-DFA algorithm for SSNs computes each error $\delta$ by direct feedback from the output layer on the spike-train level, giving to the following update rule:

$$\boxed{\begin{aligned} \Delta w_{ij} &= \eta \frac{\partial E}{\partial w_{ij}} = \eta \delta_i^k e_{i|j}^k, \\ \delta_i^k &= \begin{cases} \frac{o_i^o - y_i^o}{\nu} & \text{for output layer,} \\ \sum_{l=1}^{r^o} \delta_l^o b_{li}^k & \text{for hidden layers,} \end{cases} \end{aligned}} \tag{3.10}$$

where $\eta$ is the learning rate, $\delta_i^k$ the error of the neuron $i$ in the $k_{th}$ hidden layer, $e_{i|j}^k$ the S-PSP from the neuron $i$ to neuron $j$, $o_i^o$ the actual firing count of neuron $i$ in the output layer, $y_i^o$ the desired firing count for the neuron $i$, $\nu$ the firing threshold, $r^o$ the number of neurons in the output layer, $\delta_l^o$ the error of the neuron $l$ in the output layer, and $b_{li}^k$ the value of the fixed random feedback.

The last equation of (3.10) is based on the concept of DFA. As in Fig. 3.2, with ST-DFA, the output layer is fully connected to each hidden layer through a different matrix which is called the random feedback matrix $\boldsymbol{B}$. The weights (values) in these matrices

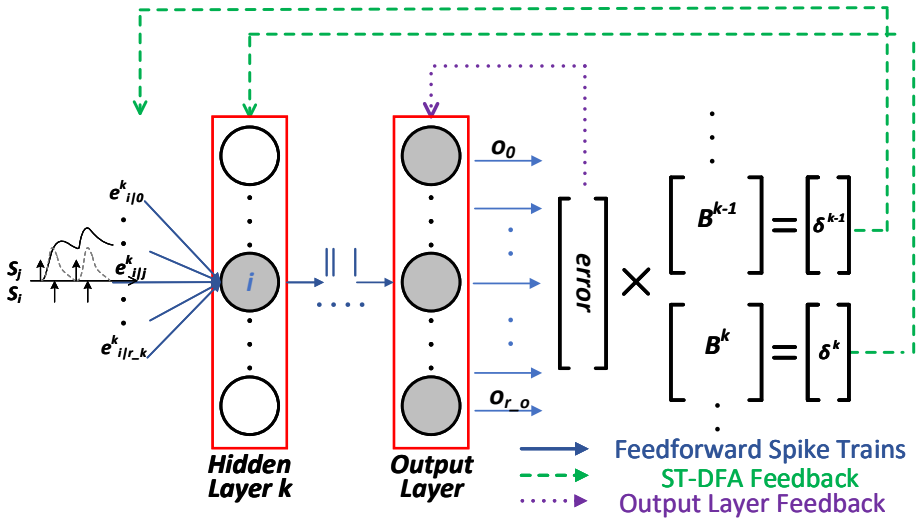


Figure 3.2: The proposed spike-train level DFA (ST-DFA).

are randomly generated and then stay fixed. The error vector $\boldsymbol{\delta}^k$ of the hidden layer $k$ is directly obtained from the error vector of the output layer $\boldsymbol{\delta}^o$ and the random feedback matrix $\boldsymbol{B}^k$ as: $\boldsymbol{\delta}^k = \boldsymbol{B}^k \times \boldsymbol{\delta}^o$. The detailed derivation of ST-DFA is introduced next.

### 3.2.2 Derivation of ST-DFA

Similar to (3.8) and using (3.7), we define the rate-coded loss function as:

$$E = \frac{1}{2}||\boldsymbol{o} - \boldsymbol{y}||_2^2 = \frac{1}{2}||\frac{\boldsymbol{a}}{\nu} - \boldsymbol{y}||_2^2, \tag{3.11}$$

where $\boldsymbol{y}$, $\boldsymbol{o}$ and $\boldsymbol{a}$ are vectors specifying the desired firing counts (label), the actual firing counts, and the T-PSP of the output neurons, respectively. Differentiating the loss function with respect to each trainable weight $w_{ij}$ leads to:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{ij}} = \delta_i^k \frac{\partial a_i^k}{\partial w_{ij}}, \tag{3.12}$$

where $a_i^k$ is the T-PSP of the neuron $i$ in the $k_{th}$ layer.

It is instrumental to note that each S-PSP $e_{i|j}$ depends on both rate and temporal information of the pre/post spike trains, i.e. $e_{i|j}$ depends on the pre/post-synaptic firing counts $o_i$ and $o_j$ and pre/post-synaptic firing times $\mathbf{t}_j^{(f)}$ and $\mathbf{t}_i^{(f)}$:

$$e_{i|j} = f(o_j, o_i, \mathbf{t}_j^{(f)}, \mathbf{t}_i^{(f)}). \tag{3.13}$$

For the $i_{th}$ output neuron, $\delta_i^o$ can be obtained from (3.12) and (3.7):

$$\delta_i^o = \frac{\partial E}{\partial a_i^o} = (o_i - y_i)\ \frac{\partial o_i}{\partial a_i} = \frac{o_i - y_i}{\nu}. \tag{3.14}$$

For each $i_{th}$ neuron in the hidden layer $k$, $\delta_i^k$ is derived from the chain rule based on (3.7):

$$\begin{aligned} \delta_i^k = \frac{\partial E}{\partial a_i^k} &= \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}}\ \frac{\partial a_l^{k+1}}{\partial a_i^k} = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1}\ \frac{\partial a_l^{k+1}}{\partial a_i^k} \\ &= \sum_{l=1}^{r^{k+1}} \delta_l^{k+1}\ w_{li}^{k+1}\ \frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}. \end{aligned} \tag{3.15}$$

The first key development in ST-DFA is that the way in which the error $\delta_i^k$ is calculated in each hidden layer changes from

$\sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li}^{k+1}\ \frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}$ to $\sum_{l=1}^{r^o} \delta_l^o d_{li}^k\ \frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}$, where $d_{li}^k$ is the direct feedback alignment from the output neuron $l$ to the hidden layer neuron $i$. $d_{li}^k$ is a randomized and fixed value. In this process, we replace the $w_{li}^{k+1}$ from $(k+1)_{th}$ layer to $k_{th}$ layer in (3.15) by $d_{li}^k$, leading to:

$$\delta_i^k = \delta_l^o d_{li}^k\ \frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}. \tag{3.16}$$

As such, the error $\boldsymbol{\delta}^k$ of each hidden neuron is directly determined by the output layer error vector $\boldsymbol{\delta}^o$ rather than by the error vector $\boldsymbol{\delta}^{k+1}$ of the next layer.

Moreover, we have the following key observation. In (3.16), since $d_{li}^k$ is randomly generated, $\frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}$ can be absorbed into $d_{li}^k$ to further simplify ST-DFA. Denote the new DFA parameter absorbing $\frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}$ by $b_{li}^k = d_{li}^k \frac{\partial e_{l|i}^{k+1}}{\partial a_i^k}$, the simplified error computation becomes:

$$\delta_i^k = \begin{cases} \frac{o_i^o - y_i^o}{\nu} & \text{for output layer,} \\ \sum_{l=1}^{r^o} \delta_l^o b_{li}^k & \text{for hidden layers,} \end{cases} \tag{3.17}$$

where $b_{li}^k$ is one entry of the random feedback matrix $\boldsymbol{B}$ in Fig. 3.2.

Thus, ST-DFA reduces the computational complexity by not only avoiding layer-by-layer propagation but also the additional simplification via the use of $b_{li}^k$.

### 3.2.3 Simplification for Hardware Friendliness

The last term on the right-hand side of (3.12) differentiates the total post-synaptic potential (T-PSP) $a_i^k$. Considering (3.7), it can be written as:

$$\begin{aligned} \frac{\partial a_i^k}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \left( \sum_{j=1}^{r^{k-1}} w_{ij} \; e_{i|j}^k \right) = e_{i|j}^k + \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{i|l}^k}{\partial o_i^k} \; \frac{\partial o_i^k}{\partial w_{ij}} \\ &= e_{i|j}^k + \frac{e_{i|j}^k}{\nu} \; \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{i|l}^k}{\partial o_i^k}. \end{aligned} \tag{3.18}$$

The exact evaluation of the above expression requires multiple additions, multiplications, and divisions, introducing high hardware overhead and additional latency.

The first term $e_{i|j}^k$ on the right-hand side of (3.18) can be interpreted as the direct influence exerted on the T-PSP $a_i^k$ by changing the synaptic weight $w_{ij}$ as seen from (3.7). The second term $\frac{e_{i|j}^k}{\nu} \; \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{i|l}^k}{\partial o_i^k}$ comes from the fact that changing the weight $w_{ij}$ leads to variation in the post-synaptic spike train. Thus, the S-PSP $e_{i|l}^k$ to the neuron $i$ also varies as it depends on the firing times of the post-synaptic neuron. Nevertheless, we have

observed that the first term dominates the second term. By dropping the second term, we reach the final hardware-friendly ST-DFA algorithm of (3.10), which also maintains good performance.

In comparison, the spike-train level BP algorithm HM2-BP is [4]:

$$\begin{aligned} \Delta w_{ij} &= \eta \delta_i^k e_{i|j}^k \left(1 + \frac{1}{\nu} \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{i|l}^k}{\partial o_i^k}\right), \\ \delta_i^k &= \begin{cases} \frac{o_i^k - y_i^k}{\nu} & \text{for output layer,} \\ \frac{1}{\nu} \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li} \frac{\partial e_{l|i}^{k+1}}{\partial o_i^k} & \text{for hidden layers.} \end{cases} \end{aligned} \tag{3.19}$$

While HM2-BP delivers the state-of-the-art performance, it would be very costly to implement on hardware if ever feasible.

In all, compared to HM2-BP in (3.19), ST-DFA in (3.10) is much more hardware friendly. With ST-DFA, direct error feedback to each hidden layer is accomplished without layer-by-layer back propagation while HM2 requires high-resolution multiplications with the transpose of the forward weights and other expensive operations layer by layer. In the next section, we efficiently realize the ST-DFA algorithm on digital hardware.

## 3.3 SNN Accelerators with ST-DFA On-chip Training

### 3.3.1 Architecture

Fig. 3.3 shows the architecture of the proposed multi-layer feed-forward spiking neural processors with the proposed ST-DFA on-chip training. Only two hidden layers are shown for illustration purpose. Architecturally, the processor is comprised of an input

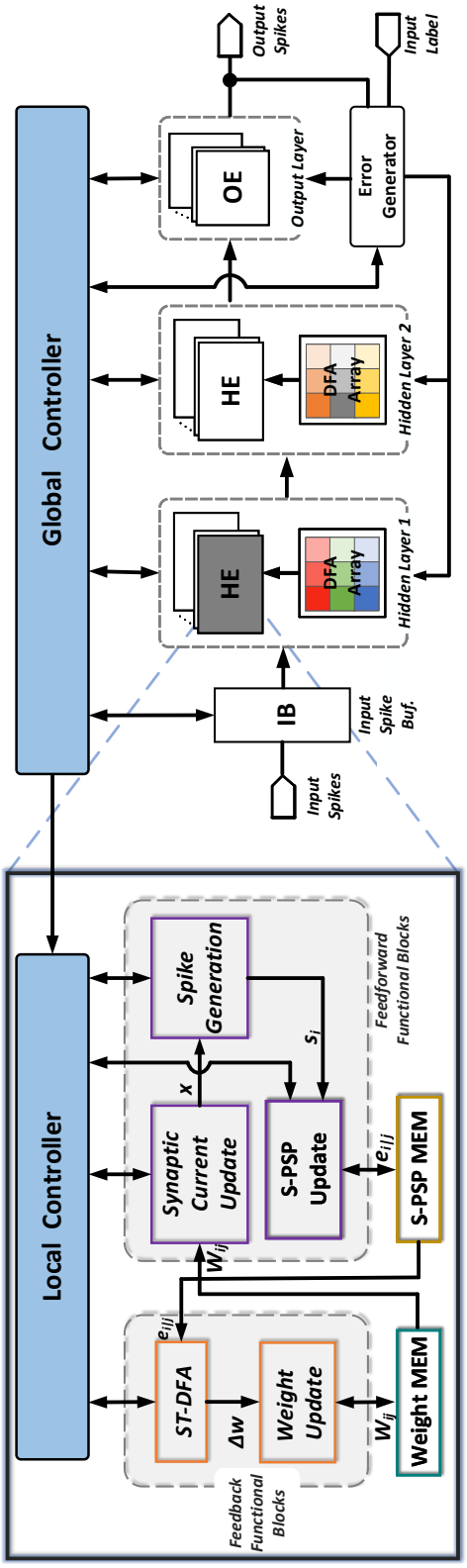


Figure 3.3: Proposed architecture of multi-layer SNNs with onchip ST-DFA training. HE represents a digital hidden neuron element; and OE represents a digital output neuron element.

spike buffer feeding multiple hidden layers composed of hidden neuron elements (HEs). The last hidden layer connects to the output layer which consists of a set of output neuron elements (OEs). A modular design approach is taken where each spiking neuron is implemented in the form of HE or OE. As such, a proper number of HEs and OEs can be instantiated to form a multi-layer SNN with arbitrary depth and width.

Both inference and training are supported. Training over an input example splits into two phases: forward pass and backward pass. The computation of S-PSPs required for ST-DFA training are computed in an online manner in the forward pass of training. The remaining computations of the forward pass are identical to those performed in inference. To support ST-DFA training, the error generator utilizes an array of subtractors to compute the difference between the actual OE output spike counts with expected ones (label). At each hidden layer, this output-layer error vector is multiplied with the associated ST-DFA random feedback matrix inside each layer to allow weight updates performed by each neuron.

### 3.3.2 On-chip Training

For each training example, the forward and backward passes of the training are controlled by a global controller (FSM) as shown in Fig. 3.3. Neurons at the same layer process information in parallel to exploit the inherent parallelism of the hardware SNN processor architecture. In the forward pass and at each biological time step, layers are activated by the global controller one at a time from the input to the output. After output spikes are generated for the current time step, the global controller pushes the training forward to the next time step. This process repeats until the current training example has been entirely learned by the network. Then, the backward pass starts, in which the first step is to calculate the output error $\delta_l^o$ in (3.10). After that, all hidden

layers start to perform ST-DFA for weight updating at the same time. The weight update latency of each hidden layer may be different due to the differences in the number of input synaptic connections (i.e. the preceding layer width). After all hidden layers finish ST-DFA weight updates, the training process moves onto the next training example.

### 3.3.3 Neuron Unit Design

Each HE or OE contains several functional blocks categorized into feed-forward functional blocks and feedback functional blocks as shown in Fig. 3.3. OEs are identical to HEs except that no ST-DFA module is included since the error $\delta_i^k$ defined for output neurons is computed by the Error Generator module. Each neuron unit contains two memory modules that store the synaptic weights and all its spike-train level post-synaptic potentials (S-PSPs), respectively. We implement the weight memory with block RAM (BRAM) and the S-PSP memory with a 2-D array of flip flops (FFs) on the FPGA. A neuron-level local controller (FSM) controls the detailed inference/training steps. The local controller also communicates with the global controller for synchronizing processes between different layers and inference/training stages.

In the forward pass of training, first, the synaptic current $x$ through each synapse is calculated, followed by the spike-train level post-synaptic potential (S-PSP) update for the same synapse. The synaptic current update and the S-PSP update modules shown in Fig. 3.3 are shared by all input synapses. Hence, processing of all synapses are done in series. After all synaptic responses are generated, the spike generation module calculates the neuron's membrane potential and makes the firing decision based on the leaky integrate-and-fire (LIF) spiking neuron model. In the backward pass of training, the ST-DFA module implements the proposed on-chip ST-DFA training algorithm, the output of which is then fed to the weight update module. Finally, the corresponding synaptic

weight is updated and stored back to the weight memory. Similar to the feedforward blocks, the feedback functional modules are also shared among all input synapses.

### 3.3.4 Efficient On-chip S-PSP Calculation

One important component in the proposed ST-DFA algorithm is the spike-train level post-synaptic potential (S-PSP), $e_{i|j}$, in (3.10). As demonstrated in (3.6), by definition, $e_{i|j}$ is the effect of all firing events of the pre-synaptic neuron $j$ on the post-synaptic neuron $i$. However, direct implementation of (3.6) on hardware is very costly; all firing events of the pre- and postsynaptic neurons need to be stored and excessive multiplication, division and exponentiation operations are involved, incurring much logic complexity and memory usage.

Instead, we propose an online S-PSP calculation approach with dramatically reduced hardware overhead. Rather than recording all firing events of the two neurons and computing $e_{i|j}$ at once in the backward pass, in the forward pass we accumulate and update $e_{i|j}$ at the arrival of each firing event and store the updated $e_{i|j}$ in the S-PSP memory of each neuron element.

Inspecting (3.3) and (3.6) reveals that $e_{i|j}$ is the normalized (by weight) of the contribution from the postsynaptic neuron $j$ to the aggregated membrane potential of the postsynaptic neuron $i$. While the aggregated postsynaptic membrane potential is effectively tracked by the LIF model, each individual contribution $e_{i|j}$ to it can be accumulated

exactly using the following equations:

$$\begin{aligned}
\tau_s \frac{p_{i|j}(t)}{dt} &= -p_{i|j}(t) + \sum_{t_j^{(f)}} D(t - t_j^{(f)}), \\
\tau_m \frac{q_{i|j}(t)}{dt} &= -q_{i|j}(t) + p_{i|j}(t), \\
e_{i|j}(t) &= \sum_{t_i^{(f)}} q_{i|j}(t_i^{(f)}),
\end{aligned} \tag{3.20}$$

where $p_{i|j}(t)$ is the (normalized) synaptic input from the neuron $j$ to neuron $i$, which is part of (3.2), and $q_{i|j}(t)$ is interpreted as the (normalized) postsynaptic membrane voltage contribution from the neuron $j$ to neuron $i$, which shall be reset to zero when the neuron $i$ fires at a particular firing time $t_i^{(f)}$.

The hardware realization of (3.20) is based on discretizing it using the first-order Euler method with a fixed stepsize:

$$\begin{aligned}
q_{i|j}[t+1] &= (1 - \frac{1}{\tau_m}) q_{i|j}[t] + p_{i|j}[t+1] \\
p_{i|j}[t+1] &= (1 - \frac{1}{\tau_s}) p_{i|j}[t] + \frac{1}{\tau_s} \sum_{t_j^{(f)}} D_n(t - t_j^{(f)}) \\
&\begin{cases}
e_{i|j}[t+1] \mathrel{+}= q_{i|j}[t+1] & \\
q_{i|j}[t+1] = 0 & \quad if\ t+1 = t_i^{(f)},
\end{cases}
\end{aligned} \tag{3.21}$$

where $D_n(\cdot)$ is the unit sample function and we have abused the notation by using $t$ and $t+1$ to indicate a discrete time step and the step after that.

(3.21) allows $e_{i|j}$ to be accumulated in an online manner with great hardware efficiency and its implementation is shown in Fig. 3.4. At each time step, we first update the value of $p_{i|j}$, followed by the updates of $q_{i|j}$ and $e_{i|j}$, controlled by the FSM states of the local controller shown in Fig. 3.3. The shaded blocks in Fig. 3.4 are registers used to store the

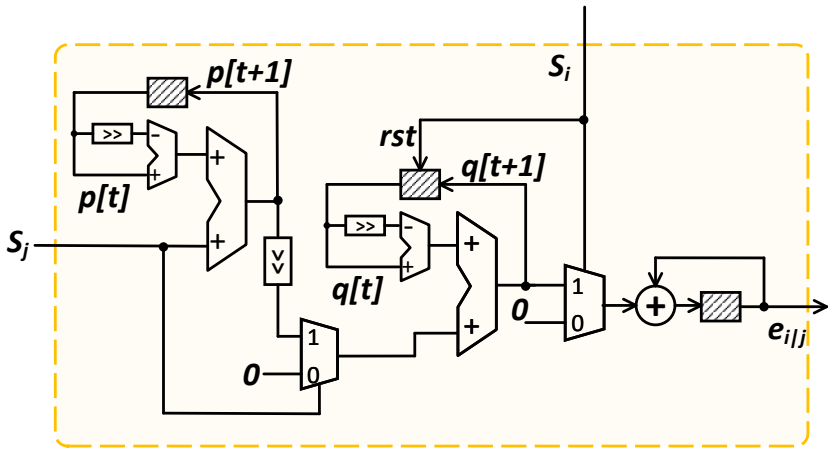


Figure 3.4: On-line S-PSP calculation onchip.

current-time variable values. We set both decay constants $\tau_s$ and $\tau_m$ to be a power of 2 such that multiplications/divisions are realized efficiently using shift operations. The updated $e_{i|j}$ is stored in the S-PSP memory and retrieved by the ST-DFA module during the backward training pass.

### 3.3.5 Efficient On-chip ST-DFA Implementation

Fig. 3.5 depicts the ST-DFA module in hidden neurons shown in Fig. 3.3. As in (3.10), for each hidden neuron $i$, the inner product between the error vector $\delta_l^o$ from the output layer and the $i$-th column of the random feedback matrix $B$ of the corresponding layer is computed. The inner product is then multiplied with $e_{i|j}$ to produce the weight update value $\Delta w_{ij}$ for the $j$-th input synapse. All these inner products for different synapses are computed in series and would result in large hardware and power overheads. Furthermore, if each entry of the feedback matrix is set to be a high-bit resolution random number, high memory usage is required for storage.

To mitigate the above design complexity, we propose a hardware-friendly realization of ST-DFA, named ST-DFA-2. ST-DFA-2 is based on the key observation from extensive algorithmic experiments that the feedback matrix $B$ need not be generated in a true

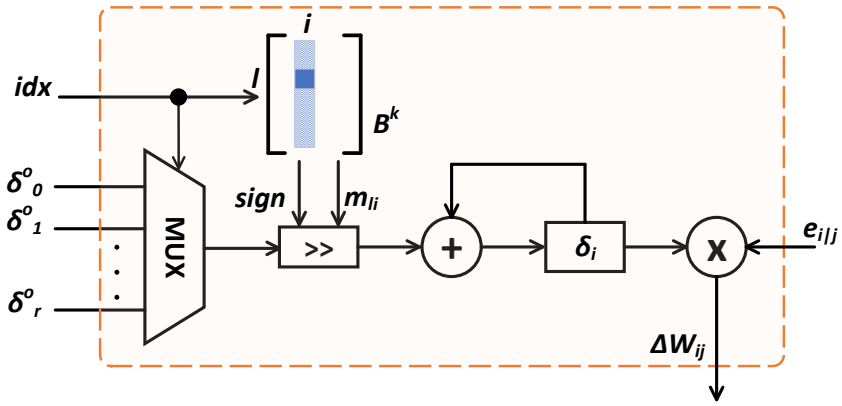


Figure 3.5: On-chip ST-DFA weight update computation.

random manner; setting each entry $b_{li}$ of $B$ to one of a small set of fixed numbers at random is sufficient for achieving good training performance. Furthermore, the set of fixed numbers can be optimized for hardware efficiency. For this, we construct this set by making each number a signed power of 2 with low-bit resolution such that the multiplications in (3.10) can be implemented by shift operations and storage for $B$ is kept at minimal.

Fig. 3.5 illustrates the computation of each weight update. The corresponding inner product is computed by accumulating the element-wise products. The $idx$ signal selects a particular element in the error vector $\delta^o_l$ and its shift amount $m_{il}$, which is set by the corresponding $b_{li}$ in the $B$ matrix according to $|b_{li}| = 2^{m_{il}}$ . If $b_{li}$ is negative, the shift result is converted to its compliment before added to $\delta_i$. Finally, the resulting $\delta_i$ is multiplied with the S-PSP $e_{i|j}$ to get the weight update value $\Delta w_{ij}$ for the current synapse.

## 3.4 Experiments and Results

### 3.4.1 Experimental Settings and Benchmarks

Performance evaluation is divided into two parts:

1) The first section devotes to evaluate the performance of proposed ST-DFA and ST-DFA-2 compared to HM2-BP. The classification performances are evaluated by software simulation of the digital computations with the actual bit resolutions implemented on FPGA. Major SNN variables, for example synaptic weight $w$, S-PSP $e_{i|j}$ and membrane potential $v$, are in the fixed-point representation. Each $w$ is a signed 17-bit variable with 12-bit fractional. 11 bits are used for each unsigned variable $e_{i|j}$ with 6-bit fractional and 9 bits are used for each signed variable $v$ with 3-bit fractional.

2) In the following section, We measure the performance vs. hardware overhead trade-offs of the proposed on-chip ST-DFA training on several feed-forward SNN neural processors. Using multiple SNNs models with varying depths and widths, we demonstrate the performance of both software and hardware (on-board) simulation. Compared to hardware implementation of HM2-BP, proposed ST-DFA significantly reduces hardware overhead which proves hardware-friendliness. FPGA prototypes of SNN neural accelerators are designed on the Xilinx ZC706 platform for performance evaluation, design overhead, and power/energy analysis.

Three datasets are employed for evaluation: MNIST[13], N-MNIST, or the neuromorphic version of MNIST [14], and the 16-speaker English letter subset of the TI46 speech corpus [36]. The MNIST handwritten digit dataset [13] contains 60k training and 10k testing examples, each of which is a $28 \times 28$ grayscale image. Each pixel value of the MNIST image is converted into a spike train using Poisson sampling and the probability of spike generation is proportional to the pixel intensity. Due to the limited hardware resources available on the Xilinx Zynq ZC706 board, we crop each image to include only

the $14 \times 14$ pixels around the center for FPGA evaluation.

The N-MNIST dataset [14] is a neuromorphic version of MNIST. The static digit images of MNIST are converted into spike trains using a dynamic vision sensor (DVS) [38] moving on a pan-tilt unit. The image is resized to $34 \times 34$ since the relative shift of images during the saccade process is required. Two kinds of spike events, ON and OFF, are recorded since the intensity can either increase or decrease. Thus, each N-MNIST image has $34 \times 34 \times 2 = 2312$ spike sequences lasting for about $300ms$. We reduce the time resolution of the N-MNIST images by 500x to speed up the processing.

The TI46 Speech corpus [36] contains spoken English letters from 16 speaker. There are 4,142 and 6,628 spoken English letters for training and testing, respectively. The continuous temporal speech waveforms are first preprocessed by the Lyon's ear model [37] and then encoded into 78 spike trains using the BSA algorithm [51].

Among these datasets, MNIST and TI46 are tested on both software and hardware while N-MNIST is only tested on software simulation due to that the available FPGA resources are not sufficient to support the large number of spike trains. Moreover, to thoroughly assess the classification performance and hardware benefits of our proposed spike-train level direct feedback alignment (ST-DFA), we build multiple SNNs with different network depths and widths.

### 3.4.2 Classification Accuracies

The proposed spike-train level direct feedback alignment (ST-DFA) algorithm is inspired by the spike-train level backpropagation HM2-BP algorithm. In [4], HM2-BP is compared with other state-of-the-art spiking or non-spiking BP methods such as spike-based BP [10], STBP [11], temporal coding BP [52] and non-spiking BP [53] on MNIST and N-MNIST. Apart from its high efficiency due to the spike-train level processing,

Table 3.1: Inference accuracy comparison of HM2BP, ST-DFA and ST-DFA-2. All SNNs are fully connected networks with a single hidden layer of 800 neurons. MNIST: 28x28 input resolution; N-MNIST: 2,312 input spike trains; 16-speaker TI46: 78 input spike trains.

| Dataset | Learning rule & Network structure | Accuracy |
|---|---|---|
| MNIST | HM2-BP: 784-800-10 | 98.93% |
| MNIST | ST-DFA: 784-800-10 | 98.64% |
| MNIST | ST-DFA-2: 784-800-10 | 98.74% |
| N-MNIST | HM2-BP: 2312-800-10 | 98.88% |
| N-MNIST | ST-DFA: 2312-800-10 | 98.47% |
| N-MNIST | ST-DFA-2: 2312-800-10 | 98.59% |
| TI46 | HM2-BP: 78-800-26 | 89.92% |
| TI46 | ST-DFA: 78-800-26 | 87.00% |
| TI46 | ST-DFA-2: 78-800-26 | 87.31% |

HM2-BP outperforms or is on a par with all these recently developed algorithms. For example, with a single hidden layer of 800 neurons, HM2-BP can achieve 98.93% accuracy on MNIST while [53] gets up to 98.30%. HM2-BP obtains 98.88% accuracy on N-MNIST compared with 97.80% by [52]. Moreover, HM2-BP delivers competitive performance on challenging benchmarks such as the 16-speaker spoken English letters of TI46 Speech corpus [36] and 47-class image recognition dataset Extended MNIST (EMNIST) [54].

As presented in Section 3.2, ST-DFA propagates the errors $\delta$ from the output layer to each hidden layer directly without layer by layer error backpropagation through symmetric weights matrices. In Section 3.3.5, we further optimize ST-DFA by setting each entry of the random feedback matrix $\boldsymbol{B}$ to a power of 2, leading to the hardware-friendly ST-DFA-2 algorithm. In this work, feedback matrix entries are randomly chosen from the set $\{-4, -2, -1, 0, 1, 2, 4\}$ for ST-DFA-2.

Table 3.1 compares the inference accuracies of HM2-BP, ST-DFA, and ST-DFA-2 on MNIST, N-MNIST, and TI46. Compared to HM2-BP, ST-DFA and ST-DFA-2 still

maintain rather competitive performance while the low computational cost and hardware-friendliness of ST-DFA-2 translate into huge hardware resources and energy overhead savings as shown later. It shall be noted that in comparison with ST-DFA, ST-DFA-2 does not necessarily degrade performance; it can even slightly outperform ST-DFA in practice.

### 3.4.3 FPGA Hardware Evaluations

We build several FPGA SNN accelerators on the targeted Xilinx ZC706 platform, the sizes of which are decided considering the available resources onchip. Table 3.2 shows the resource and energy overhead as well as the software/hardware inference accuracies of these SNN accelerators with on-chip ST-DFA-2.

As shown in the table, the implemented networks have either one or two hidden layer(s), and each hidden layer has 50 or 100 neurons. Numbers of input and output neurons are application-dependent. Training powers are estimated by the Xilinx Power Analyzer based on application-specific workloads. The training latency and training energy are for training a representative input example of the corresponding dataset using one iteration of forward and backward passes. Table 3.2 indicates that the SNNs integrated with ST-DFA-2 in general have efficient FPGA resource utilization as well as low training energy dissipation.

Furthermore, with a trimmed down input size and/or constrained network size, the FPGA SNNs with on-chip ST-DFA-2 can still deliver competitive classification performance in reference to the simulated accuracies achieved at full input size and by larger networks reported in Table 3.1. For instance, the accuracy of MNIST is based on full input resolution which is $28 \times 28$ with a hidden layer of 800 neurons. However, for on-board simulation, we implemented with reduced input resolution and smaller networks

due to the Xilinx ZC706 board resource limitation. We cropped each data of MNIST into 14x14 which causes 4X input resolution reduction and built smaller networks consists of 50 or 100 hidden neurons.

To better illustrate the cost-effectiveness of the proposed ST-DFA algorithm, we also compare the overheads of implementing HM2-BP vs. ST-DFA-2 in a fully-connected SNN FPGA with two hidden layers in Table 3.3. Since the main difference between HM2-BP and ST-DFA is the backward pass algorithm, we designed HM2-BP in hardware based on the weight updating algorithm represented in [4]. Training latency of the backward pass of the corresponding SNN neural processor is also presented in the table. We do not consider forward pass latency and inference latency since they do not differ significantly in the two cases. The results in the table indicate that ST-DFA is much more efficient in terms of hardware implementation on both resource utilization and backward pass latency compared with HM2-BP. The ST-DFA-2 based SNN neural processor saves 18% on LUTs, 76.7% on DSPs and 31.6% on backward phase latency compared with the HM2-BP based SNN.

The large additional hardware overhead and backward latency of HM2-BP mainly come from the layer-by-layer error propagation and the required multiplication operations. Moreover, as the network goes deeper, the backward phase latency grows proportionally in HM2-BP, while in ST-DFA the backward latency will not affect by the network depth since the error processing is concurrently executed in all hidden layers. This property assures the scalability of ST-DFA which is promising for deeper networks. With the proposed ST-DFA algorithm, we have sidestepped the complex backpropagation and enabled cost-effective on-chip training for multi-layer SNNs.

Table 3.2: Overheads and inference performances of the fully-connected SNNs with on-chip ST-DFA-2.

| MNIST (14x14 input resolution) @100 MHz | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Resource Utilization | | | Training Power (mW) | Training Latency (mS) | Training Energy (mJ) | Accuracy (Software Simulation) | Accuracy (On-board Simulation) |
| | LUTs | FFs | DSPs | | | | | |
| **196-50-10** | 33484 | 6836 | 60 | 113 | 3.998 | 0.452 | 95.48% | 94.34% |
| **196-50-50-10** | 62989 | 12516 | 110 | 125 | 4.836 | 0.604 | 95.87% | 94.51% |
| **196-100-10** | 73027 | 12329 | 110 | 224 | 4.802 | 1.076 | 96.86% | 95.72% |
| **196-100-100-10** | 126482 | 23331 | 210 | 275 | 6.445 | 1.772 | 97.23% | 96.27% |
| **TI46 (16-speaker Spoken English Letters) @100 MHz** | | | | | | | | |
| | Resource Utilization | | | Training Power (mW) | Training Latency (mS) | Training Energy (mJ) | Accuracy (Software Simulation) | Accuracy (On-board Simulation) |
| | LUTs | FFs | DSPs | | | | | |
| **78-50-26** | 38220 | 8826 | 76 | 73 | 3.688 | 0.269 | 73.34% | 71.63% |
| **78-50-50-26** | 74709 | 14641 | 126 | 87 | 5.123 | 0.445 | 76.45% | 74.95% |
| **78-100-26** | 64280 | 14096 | 126 | 113 | 5.089 | 0.575 | 77.64% | 75.19% |
| **78-100-100-26** | 145452 | 30546 | 226 | 185 | 7.929 | 1.467 | 87.40% | 84.88% |

Table 3.3: Overheads of an FPGA SNN with on-chip HM2-BP vs. ST-DFA-2 (Network size:196-100-100-10)

| | LUTs | FFs | DSPs | Backward Phase Latency (uS) | Normalized LUTs | Normalized FFs | Normalized DSPs | Normalized B-P Latency |
|---|---|---|---|---|---|---|---|---|
| **HM2-BP** | 154477 | 23462 | 900 | 17.560 | 122% | 101% | 429% | 146% |
| **ST-DFA** | 126482 | 23331 | 210 | 12.010 | 100% | 100% | 100% | 100% |

## 3.5 Summary and Discussions

While Direct Feedback Alignment (DFA) has been attempted, this work presents the first novel approach for implementing hardware spiking neural networks(SNNs) on the FPGA board. This work aims to build efficient on-chip training FPGA SNN neural processor with reduced backward training latency and hardware cost while gracefully trading off classification performance. To be specific, Table 3.2 shows competitive software/hardware inference accuracy despite reduce input resolution and small network size. Comparing the accuracy of ST-DFA and ST-DFA-2, performance using ST-DFA can be slightly better than using ST-DFA-2 and vice versa. This fact shows that the error may vary based on randomly initialized feedback weight matrix, which makes ST-DFA-2 still powerful. Table 3.3 shows the advantages of ST-DFA-2 over HM2-BP in terms of hardware resource utilization through the DFA algorithm and the efficient design of the hardware design units. As shown in Table 3.1 and Table 3.2, this result proves the practicality of the DFA algorithm and the feasibility of implementing the ST-DFA algorithm for on-chip training of the SNN processor.

Implementing training of SNNs using a BP algorithm suffers from high computing complexity and thus high resource utilization, while a hardware-friendly, non-BP algorithm, such as STDP, suffers from achieving good accuracies. We argue that our approach avoids high computation complexity by extending a BP-like algorithm (i.e. DFA) for SNNs while maintaining the advantage of a well-defined BP-like algorithm in terms of accuracy. To the best of our knowledge, this is the first work presenting algorithm-hardware co-optimization and demonstrating the realization of DFA, which is an efficient on-chip training algorithm, for SNNs.

The main focuses of this paper have been on extending the DFA concept proposed by [48] to efficient training of SNNs, and significantly reducing the hardware cost by

algorithm-hardware co-optimization while maintaining a competitive accuracy. This paper proposes a novel spike-level direct feedback alignment (ST-DFA) algorithm for training multi-layer spiking neural networks (SNNs) with improved bio-plausibility and scalability over traditional backpropagation algorithms. Moreover, it is demonstrated that the ST-DFA algorithm with its hardware-friendly optimized implementation enables efficient on-chip training of FPGA SNN neural processors while delivering competitive classification performance for practical speech and image recognition tasks.

# Chapter 4

# Dataflow Optimization for Spiking Neural Networks

Hardware acceleration of conventional ANNs, in particular deep neural networks (DNNs), offers a viable solution to deployment of models that are deeper and more powerful. Previous works have proposed dataflows to maximize throughput and energy efficiency of DNN accelerators [15, 55, 56, 27, 57, 58], particularly for the widely adopted deep convolutional neural networks (CNNs) [59, 60, 61]. Also, many previous studies have presented specialized hardware platforms for supporting dense matrix multiplication efficiently with various dataflow.

Clearly, there is also a strong demand for efficient dataflows and microarchitectures for SNN accelerators. Despite its significance, dataflow optimization of SNN accelerator architectures has not been extensively studied [62, 20, 63, 64]. The unique temporal aspect of SNN operations and the all-or-none nature of spiking activities introduce new challenges in dataflow optimization. One common approach is to perform SNN acceleration in a temporally sequential manner while following an optimized non-spiking ANN dataflow, ignoring the complex spatiotemporal tradeoffs [60, 62].

In this chapter, we aim to address the stereotypical approach to accelerate spiking-CNNs by exploring alternative dataflows with proposed dataflow simulator. This work is motivated by a dearth of a knowledge base for developing efficient SNN dataflows despite its crucial impact on SNN accelerator design.

## 4.1 Dataflow Optimization for Spiking CNNs

### 4.1.1 Proposed parallel processing in temporal dimension

As discussed in chapter 2.2 and shown in Fig. 2.1, the computation of a spiking neuron consists of: 1) spike input integration, 2) membrane potential accumulation, and 3) spike output generation, as in many other SNN works [65, 66, 67]. At each time point $t$, all presynaptic inputs to the given postsynaptic neuron are integrated and then added to the postsynaptic membrane potential at the previous time $t-1$ in order to compute the potential at the present time $t$. Finally, if the updated membrane potential exceeds a prescribed firing threshed, a binary output spike is generated, and the membrane potential is reset. The spike input integration step completely dominates the total computational cost due to the typically high degree of presynaptic connectivity. As in Fig. 4.1, our key observation is that the most expensive spike input integration step can be parallelized over multiple time steps for all spiking neurons in the layer under processing given the spike outputs from the preceding layer, which act as the inputs to the current layer. This is because spike input integrations at different time points are independent of each other. Upon the completion of the first step, membrane potential accumulation and spike output generation shall be performed in a temporally ascending order due to the temporal dependency introduced by the evolution of each membrane potential and reset. Our key idea is to explore the added parallelism brought by temporal parallel processing of spike

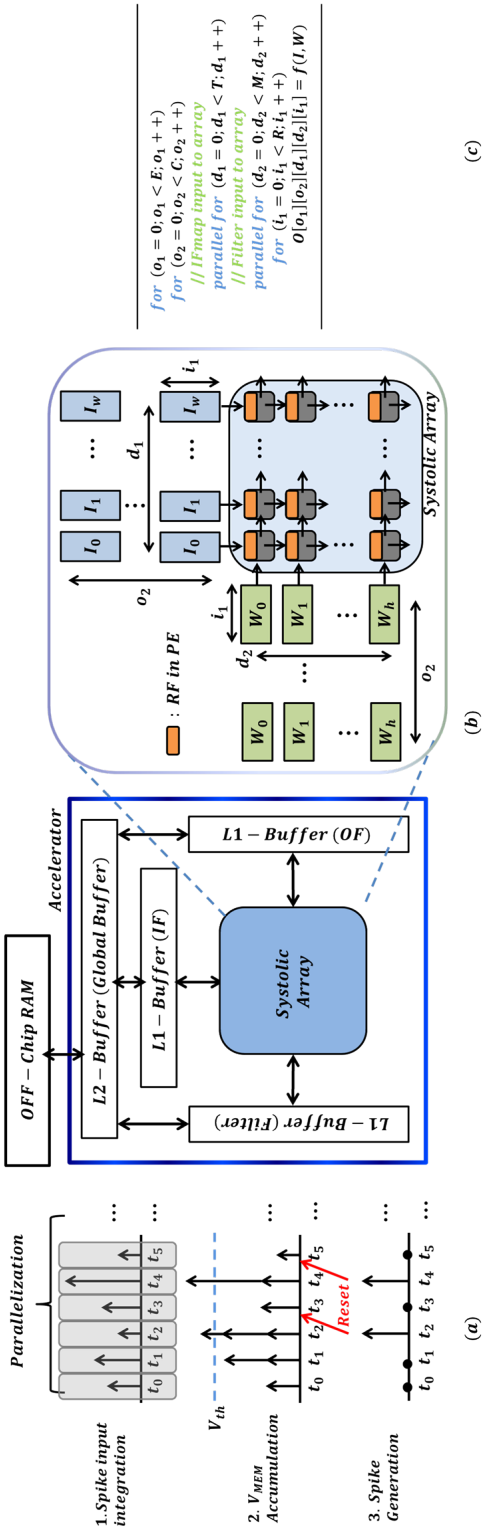


Figure 4.1: Overview of the proposed work: (a) parallelization in temporal dimension, (b) systolic accelerator design, and (c) generalized loop representation of the mapped tiling strategy.

input integration, which further allows inclusion of the temporal dimension as part of the variable tiling scheme. As shown later, systematic variable tiling optimization can be explored to effectively optimize or tradeoff between data movement, throughput, and energy dissipation in spiking based spatiotemporal processing.

### 4.1.2 Dataflow in Spiking CNNs

Dataflows specify data scheduling via variable tiling and directly impact the overall runtime and energy dissipation of a DNN accelerator [15, 55, 56, 27, 58, 61, 68]. However, dataflows for SNNs have not been extensively studied. Mapping the computations of a spiking conv layer involves directly associating two parameters (dimensions) with the 2-D systolic array to naturally exploit spatial-locality and data reuse. These two parameters are represented as d1 and d2 in Fig. 4.1. Ultimately, an optimized tiling strategy over the six-dimensional space of input, weight, output, and temporal data shall be adopted to maximize data reuse, energy efficiency, and throughput.

One common dataflow approach is to perform SNN acceleration in a temporally sequential manner while following an optimized non-spiking ANN dataflow [60, 62]. Clearly, this approach is unable to parallelize computation along the temporal dimension and ignores the complications and opportunities in dealing with complex spatiotemporal processing in SNNs. We propose in-depth dataflow optimizations considering data stationarity, variable tiling, and layer/network dependencies for spiking CNNs (S-CNNs). Also, to address this critical shortcoming of developing dataflow for SNN acceleration, we describe near-optimal dataflows to offer insights to handle the uniqueness of SNNs.

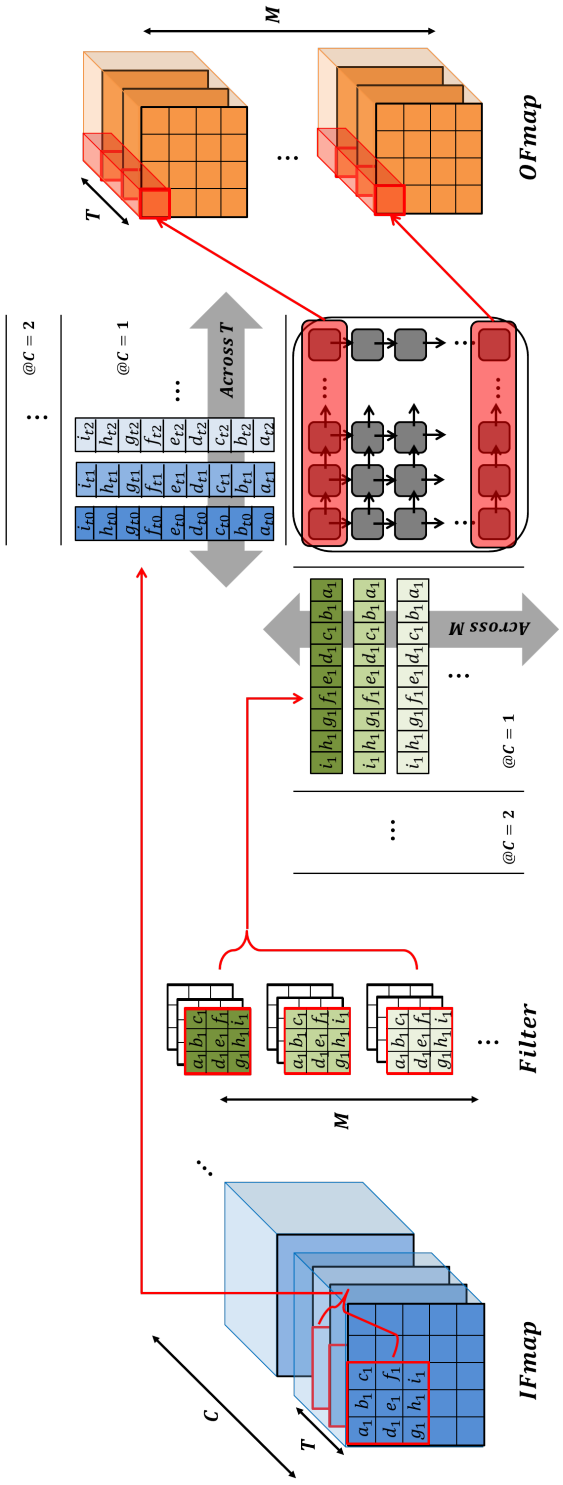


Figure 4.2: Mapping of a *Psum-friendly* output-stationary dataflow onto a systolic array accelerator.

### 4.1.3 Stationary schemes for S-CNNs

Three stationary dataflows, namely, input stationary (IS), weight stationary (WS), and output stationary (OS), have been studied for non-spiking CNN accelerators [15, 55, 56, 27, 58]. In each stationary scheme, one type of the data (i.e., IFmap, filter weight, or Psum data) stays stationary in each processing element (PE) of the array to minimize the movement of the corresponding data. For SNNs, our key observation is that the disparity among different data types must be considered, particularly with respect to the low volumes of input/output spike data for each spiking neuron. The IFmap data (input spikes) are binary and hence require low data bandwidth and are not the main bottleneck in data movement. Similarly, since the final activation of each spiking neuron is binary, the output spike data are not the bottleneck of data movement either. As such, retaining the low-volume IFmap data in the systolic array by choosing the input-stationary dataflow is not effective. The OS and WS stationary flows can minimize the data movement of high-volume multi-bit Psum and filter weight data and shall be specifically focused on for SNN accelerators.

### 4.1.4 Variable Tiling

As discussed in the previous chapter, the proposed parallelization across multiple time points allows for systematic dataflow optimization via a proper choice of variable tiling. Nevertheless, optimization of the variable tiling is a multi-faceted problem and shall be based upon a careful balancing between data reuse/movement, throughput, and energy efficiency involved in processing spatiotemporal data. Under a given hardware resource constraint and resource allocation between the memory and compute units, a systolic array accelerator may be either compute-bound or memory-bound, which is further dependent on the variable tiling in conjunction with the characteristics of the

CNN layer to be processed.

While the dataflow simulator proposed in Section 4.2 supports the evaluation of variable tiling in both the compute-bound and memory-bound regimes, we discuss the impacts of different tiling strategies under memory-bound operations with respect to the high-volume multi-bit filter and Psum data, the two main bottlenecks in data movement. In particular, the positioning of temporal variable $t$ has significant impacts on the tradeoffs between runtime (throughput) and energy dissipation.

**Memory access/energy efficiency**

The energy dissipated in data movement can be orders of magnitude higher than that of the corresponding MAC operation specifically when moving data across the chip boundary [15, 55, 56, 27, 58, 57, 59, 60]. In SNNs, Psums are the most important contributor to data movement.

As the time parameter $t$ goes deeper into the loop nest, it changes more frequently. In other words, there would be more incomplete computations performed over a larger number of time points, producing a higher volume of Psums to be stored. Although the final OFmap data (i.e., spike outputs) is binary, the Psums are multi-bit. On the other hand, placing parameter $t$ in an outer-loop position helps reduce the amount of Psums. This is because, under this case, the computation of spike output of each processed spiking neuron spans over a shortened period of time (clock cycles), leading to faster conversion (summation) of multi-bit Psums to binary spike outputs. This helps reduce data movement and hence improves energy efficiency. We refer to dataflows in which $t$ is placed at an outer-loop position as a *Psum-friendly dataflow*.

### Throughput

Due to the overlap between computation and memory access, the throughput (runtime) of a systolic array accelerator depends on the more dominant delay component of the two [69, 44]. In the memory-bound regime, the filter weight access time dominates the overall runtime of an accelerator.

In output stationary (OS) dataflows, placing parameter $t$ in an inner-loop position of the variable tiling opens up the possibility for parameters related to the filter data to be placed in outer-loop positions, leading to less frequent access to multi-bit filter weight data and more filter data reuse. Ultimately, doing so leads to a greater throughput or speedup of the accelerator. We refer to such dataflows as *filter-friendly* dataflows. Conversely, placing parameter $t$ in an outer-loop position forces the parameters related to the filter data to be placed in inner-loop positions. This may degrade runtime as it requires more frequent filter data transfers, hindering the data loading before starting the systolic array computation. Alternatively, throughput can be improved by adopting a weight stationary (WS) dataflow as it immediately maximizes the filter data reuse. However, this is typically at the cost of higher energy dissipation due to the reduced Psum data reuse.

The above observations suggest that a careful trade-off between throughput and data reuse (i.e., energy dissipation) must be made while optimizing the dataflow for which the positioning the time parameter $t$ in variable tiling is one of the key considerations. One of the proposed tiling strategy, which enables the parallel processing of the temporal dimension, is shown in Fig. 4.2, where data in a spatial 2D space such as filter weight data associated with the R parameter are vectorized. The dataflow in Fig. 4.2 is an example of Psum-friendly dataflow since all Psums generated in the array are for one specific output position in the OFmap. At the same time, data processing associated with parameters

(T, M) is parallelized in a simultaneous row-wise and column-wise manner. In general, this dataflow is expected to have advantages in terms of energy efficiency but results in greater runtime due to frequent filter data transfers.

### 4.1.5 Layer-dependent dataflow reconfiguration

In a deep S-CNN, early conv layers tend to have larger lateral IFmap and OFmap dimensions and a fewer number of channels compared with late layers. For example in VGG-16, H=224 / R=3 / C=3 / M=64 for the first conv layer CONV1, and H=14 / R=3 / C=512 / M=512 for the 11-th conv layer CONV11. If the binary spike output for each spiking neuron cannot be computed quickly, all its Psums must be stored over an extended period of time, producing more Psum data movement. To prevent this from happening, one may place the IFmap channel dimension $C$ and filter spatial dimension $R$ at the inner-most loop positions. However, this may not be the optimal strategy since it can lead to worsened filter data reuse and PE utilization. This Psum data movement problem gets more pronounced for large OFmaps for which there is a tendency for storing Psum data for a larger number of spike outputs.

As a result, early layers are deemed more Psum intensive, pushing the tiling optimization more towards mitigating the impact of Psum data movement. Conversely, the processing of later conv layers with more filters is more severely bottlenecked by the need to access to a larger amount of filter data. Clearly, adopting a fixed variable tiling across different layers or SNNs will lead to non-optimal results. Reconfiguring variable tiling during runtime in a layer-dependent manner may further improve the overall accelerator performance. Comprehensive dataflow optimization considering all these intertwined factors will be supported by the proposed SNN dataflow simulator described next.

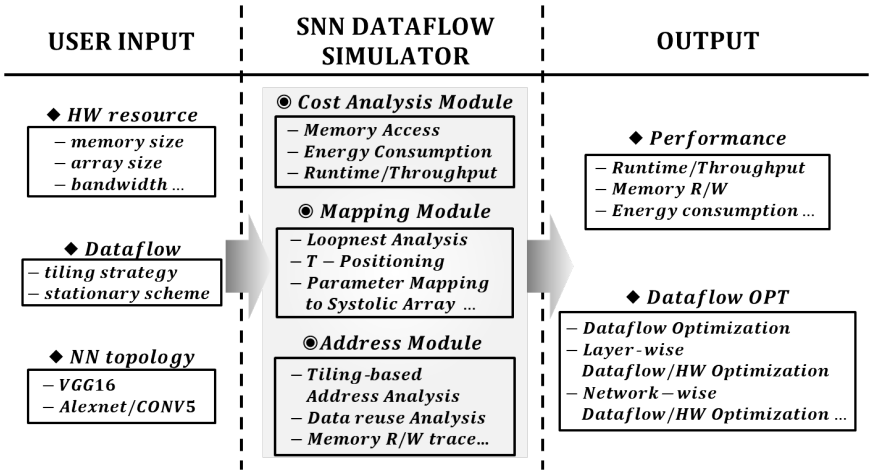


Figure 4.3: An overview of SNN dataflow simulator framework.

## 4.2 SNN Dataflow Simulator

To support dataflow optimization over a large design space while considering complex tradeoffs in the six-dimensional space of input, output, weight, and temporal data, we introduce an analytic simulator for SNN dataflows. As shown in Fig. 4.3, the simulator takes user-specified hardware configuration, including technology node and area utilization for compute units and memory, variable tiling, and targeted S-CNN as inputs. It produces a detailed analysis of runtime (throughput), memory access/data movement, and energy dissipation of the systolic-array accelerator. For the later chapters, we extend this simulator to evaluate various types of layers with the actual spiking activities for more accurate and realistic simulation.

### 4.2.1 Modeling of systolic array and memory

#### Array configurations

The developed simulator adopts a systolic array as a central compute substrate. As discussed in section 2.4, the array comprises tiled processing elements (PEs) with uni-

directional links. In this work, we use different shapes of the array with the given area constraint and compare the results to see how the different memory and array sizes have impact on the overall performance. In the later chapters, we use a fixed number of PEs for a fair comparison, which is similar number of PEs that have been adopted in other works [15, 20], and mainly focus on comparing the efficacy of the proposed techniques. Especially, the most commonly used square-shaped systolic arrays are considered. Each systolic array comprises a set of processing elements (PEs), each containing an accumulate (AC) unit and a scratchpad memory (a register file). Note that since the spike inputs are binary, the more complex MAC units are not needed.

**Memory configurations**

The memory hierarchy design is critical to the overall performance and energy consumption of spiking neural computation due to the SNN's memory-intensive nature. As a standard practice, three levels of the memory hierarchy is assumed, as shown in Fig. 4.1: an external memory (DRAM), on-chip L2/L1 buffers, and a small scratchpad memory in each PE. Similar to many other analytic models [44, 69, 17], each level of memory is double-buffered to hide latency and partitioned to separately store each type of data (IFmaps, OFmaps and Psums) for the array computation.

**Compute and memory tradeoffs**

Under a total accelerator chip area constraint, the area of the compute units vs. that of the memory can be varied. For example, the array size may be set to 8x8, 16x16, 24x24, 32x32, 40x40, or 48x48, which leaves the bulk of the remaining chip area to memory. In the later chapters, we fix the configurations of compute unit and memory to mainly focus on a fair comparison between the proposed technique and the baseline.

### 4.2.2 Performance modeling

With a given tiling strategy, the simulator generates unique addresses for each data type with respect to the inputs/outputs for the array, which follows the dataflow simulator proposed for non-spiking society [44]. The simulator produces read/write traces with the generated addresses for each-level of memory to evaluate memory access and latency, following the estimation methods in many previous works [44, 69, 21].

#### Runtime/throughput

During each systolic array processing iteration, the worst-case delay between computation/data access determines the per iteration runtime. The worst-case delay is accumulated over all iterations in the systolic array to estimate the total runtime of the accelerator. The runtime of one array iteration is modeled by the array computation cycle number which is equal to the sum of the array height/width and length of the input spike train since the data is fed from the left and right edges of the array and propagate via uni-directional links to the right/bottom end of the array. We model the data access delay based on the bandwidth and read/write traces at each level of memory.

**Memory access** - For a given network configuration, the simulator generates a trace of data scheduling based on the mapping order. With the pre-determined sequence of data required from the array, a memory access to higher level caches takes place if a specific data is absent in the current storage. For example, if the L1 buffer requires a specific data which is only presented in the global buffer, it initializes a global buffer read and a L1 buffer write.

**Energy dissipation** - Energy dissipation is evaluated based on the traces of read/writes at each level of memory and the total number of arithmetic operations in PEs based on the standard modeling strategy [15, 69, 21, 44]. Using CACTI [70] configured for 32nm

Table 4.1: Layer-specific comparison of tiling strategies in output stationary (OS) dataflows for the VGG-16 net.

| | **Normalized Performance for VGG16 CONV1 & CONV11 layers** | | | | | | |
|---|---|---|---|---|---|---|---|
| | ***Tiling strategy*** | **Runtime/Energy** | | **EDP**[c] | | **EDP I**[d] | |
| | | ***C1***[a] | ***C11***[b] | ***C1*** | ***C11*** | ***C1*** | ***C11*** |
| ref** | T/M/E/C/R | 367/32.4 | 10K/20K | 119 | 20k | 0.84X | 0.005X |
| ref* | T/R/E/M/C | 100/100 | 100/100 | 100 | 100 | 1X | 1X |
| | E/R/T/M/C | 37.6/0.48 | 62.4/64.6 | 0.18 | 40.3 | 555X | 2.5X |
| | M/T/E/C/R | 359/32.4 | 10K/10K | 117 | 1M | 0.85X | 0.0001X |
| | E/T/C/M/R | 215/7.19 | 50K/50K | 15.5 | 20M | 0.06X | 0.000005X |
| | C/M/E/T/R | 2K/4K | 931/10K | 90K | 100K | 0.001X | 0.001X |
| | M/E/T/C/R | 224/1.20 | 10K/10K | 2.69 | 2M | 37X | 0.00005X |
| | T/M/E/R/C | 178/14.1 | 139/166 | 25.2 | 229 | 3.97X | 0.43X |
| ① E-T | E/C/T/M/R | 30.9/2.88 | 18.6/138 | 0.89 | 25.6 | 112X | 3.9X |
| ② B-T | T/C/E/M/R | 22.9/24 | 14.4/127 | 5.50 | 18.2 | 18X | 6X |
| ③ R-T | C/T/E/M/R | 22.5/35.7 | 13.8/106 | 8.03 | 14.7 | 13X | 7X |

[a]: CONV1. ref*: Existing SNN dataflow [30].
[b]: CONV11. ref**: Non-optimized SNN dataflow.
[c]: Energy-Delay Product. [d]: EDP improvement.

CMOS technology, the energy dissipation is evaluated with the number of accesses based on the read/write traces, multiplied by energy per memory access at each level of memory hierarchy. The computation energy is estimated with the total number of AC operations for the given network multiplied by the energy per AC operation [69].

## 4.3 Experiments and Results

By leveraging the proposed SNN simulator, we demonstrate dataflow optimization under various systolic array configurations and choices of stationary schemes for specific spiking convolutional layers or across one entire spiking CNN network. We focus on the accelerating CONV layers as they account for over 90% of total computation [15, 69, 44]. For demonstration purposes, a 28nm technology node and two total area constraints of 8mm$^2$ and 16mm$^2$ are assumed [15, 69]. We evaluate the performance and efficiency of the systolic array accelerators for accelerating the inference of the pre-trained SNN imple-

mentations of the Alexnet and VGG-16 networks, two deep learning architectures widely adopted for image classification. These spiking CNNs have demonstrated promising classification accuracy [71, 72]. The two spiking network models operate over 200-time steps with time-wise packing of the binary spike inputs/outputs.

### 4.3.1 Layer-specific dataflow optimization

We perform comprehensive dataflow optimization for a specific spiking conv layer by comparing a large number of variable tiling strategies based on output stationary (OS) dataflows. The targeted layers are CONV1 and CONV11 from the VGG-16 net, a representative early and late conv layer, respectively. Dataflow optimization has not been targeted in most of the existing SNN works [62] with a few adopting a non-spiking CNN dataflow in which the computations at different time points are performed sequentially and the scheduling of the computation at each time step follows the chosen dataflow [62, 73]. One of the such SNN dataflows from [73] is denoted by ref* and chosen as a baseline reference.

Table 4.1 reports a comprehensive comparison of the overall accelerator performance across many tiling strategies. The accelerator performance level varies over several orders of magnitude, indicating the key importance of dataflow optimization. For comparison purposes, one non-optimized dataflow denoted by ref** in Table 4.1 is chosen as another reference. It is evident that non-optimal dataflows can result in significantly degraded overall performance.

We identify three near-optimal variable tiling schemes, denoted by ①, ②, and ③ in Table 4.1. We call dataflows ①, ②, and ③ energy-targeted (E-T), energy-runtime balance-targeted (B-T), and runtime-targeted (R-T), respectively. They favor the optimization of energy or runtime or strike a good balance between the two. As presented

Table 4.2: Joint layer-dependent dataflow and accelerator optimization under different optimization-targets for VGG-16.

| **VGG-16** | **CONV Layers (Area = 8mm$^2$)** | | | | | | | | | | | | | **Optimized** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Target** | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 | L12 | L13 | **Array Size** |
| Runtime | ③ | ③ | ③ | ② | ② | ② | ② | ③ | ③ | ③ | ③ | ③ | ③ | 24X24 |
| Energy | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | 32X32 |
| EDP | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ② | ② | ② | 32X32 |

| **VGG-16** | **CONV Layers (Area = 16mm$^2$)** | | | | | | | | | | | | | **Optimized** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Target** | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 | L12 | L13 | **Array Size** |
| Runtime | ③ | ③ | ③ | ③ | ② | ② | ② | ③ | ③ | ③ | ③ | ③ | ③ | 32X32 |
| Energy | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | 40X40 |
| EDP | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | ① | 40X40 |

*①, ②, and ③ represents the E-T (Energy-Targeted) / B-T (energy-runtime Balance-Targeted) / R-T (Runtime-Targeted) dataflow respectively.

Table 4.3: Joint layer-dependent dataflow and accelerator optimization under different optimization-targets for Alexnet.

| **Alexnet** | **CONV Layers (Area = 8mm$^2$)** | | | | | **Optimized** |
|---|---|---|---|---|---|---|
| **Target** | L1 | L2 | L3 | L4 | L5 | **Array Size** |
| Runtime | ① | ② | ③ | ③ | ③ | 24X24 |
| Energy | ② | ① | ① | ① | ② | 32X32 |
| EDP | ② | ③ | ③ | ③ | ③ | 16X16 |

| **Alexnet** | **CONV Layers (Area = 16mm$^2$)** | | | | | **Optimized** |
|---|---|---|---|---|---|---|
| **Target** | L1 | L2 | L3 | L4 | L5 | **Array Size** |
| Runtime | ② | ② | ③ | ③ | ③ | 32X32 |
| Energy | ② | ① | ② | ② | ③ | 40X40 |
| EDP | ② | ③ | ② | ① | ② | 32X32 |

in Fig. 4.2, the advantages of the E-T dataflow result from its Psum-friendly nature in that the Psums for each binary spike output are accumulated over a shortened time span, reducing the energy cost of data movement. On the other hand, R-T dataflow shows the advantages of filter reuse in that the parameters related to filters are placed in outer-loop positions, leading to greater throughput.

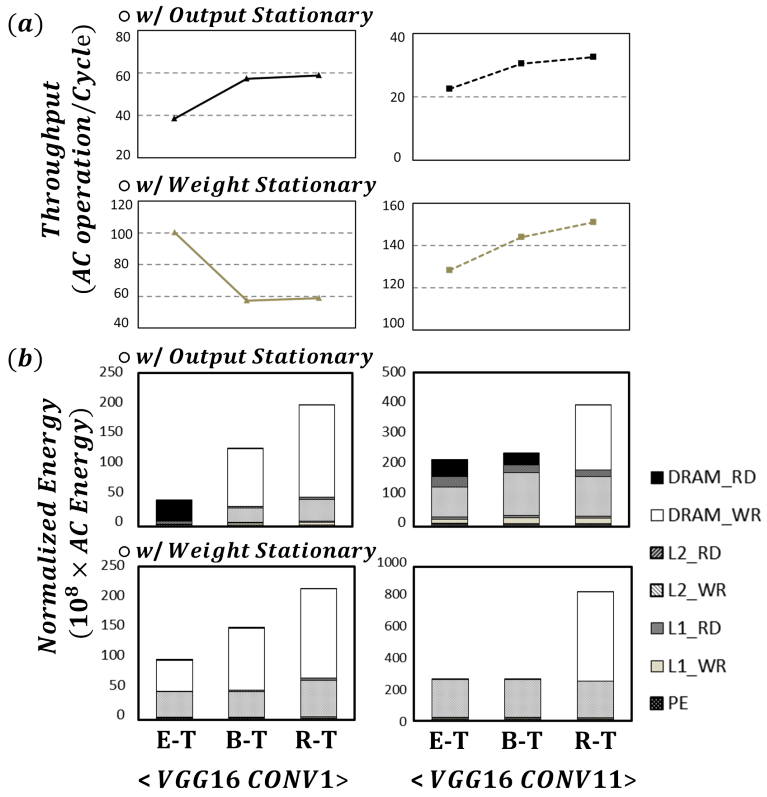


Figure 4.4: (a) The throughput of three dataflows for VGG-16 CONV1 and CONV11 with OS and WS. (b) Energy dissipation of various dataflows for VGG-16 CONV1 and CONV11. AC refers to accumulation operation.

### 4.3.2 Joint optimization of tiling and stationary flows

It is meaningful to investigate how variable tiling and stationarity of the dataflow can be jointly optimized. Fig. 4.4(a) shows the throughputs of the E-T, B-T, and R-T variable tiling when paired with the weight stationary (WS) and output (OS) schemes for the VGG-16 CONV1 and CONV11 layers. It can be seen that WS results in an improved throughput compared with OS due to the maximized reuse of the filter data for both layers, as discussed in Section 4.1.4. Note that while R-T when paired with OS provides better throughput than E-T as presented previously, with WS, E-T happens to offer higher throughput than R-T for CONV1. This suggests that for early layers, WS may exert a greater impact on throughput than variable tiling.

Fig. 4.4(b) reports the breakdown of energy consumption for the three tiling schemes when paired with OS or WS. As discussed before, the Psums data is the most important factor in energy efficiency, which is better handled by OS. As a result, OS improves energy efficiency over WS. Among all the considered tiling and stationarity combinations, paring OS with E-T leads to the best energy efficiency.

### 4.3.3 Joint layer-dependent reconfigurable dataflow and hardware optimization

We demonstrate the additional benefits brought by the proposed layer-dependent configuration of variable tiling during runtime based on OS dataflows. As discussed previously, the three tiling schemes E-T, B-T, and R-T well represent distinct trade-offs that can be made between throughput and energy dissipation. We limit the tiling choice of each layer to these three schemes while our simulator can support wide ranges of design space exploration at a higher computational cost. Furthermore, we explore accelerator hardware optimization by allocating different area budgets for the compute units and on-chip memory under a fixed total chip area constraint. This is done by evaluating the overall performance by changing the systolic array size, which correspondingly alters the amount of on-chip memory under the constant total chip area. Note that optimized accelerator hardware is not runtime reconfigurable except for the variable tiling, which is reconfigured on a layer-by-layer basis.

Table 4.2 and Table 4.3 summarize the results of joint reconfigurable dataflow and hardware optimization based on two total chip area constraints for VGG-16 and Alexnet, respectively. The optimal variable tiling for all layers in the network and the optimized accelerator in terms of its array size are reported. The results are based on maximizing one of the three performance targets across all layers of the network: throughput

($1/runtime$), energy, and energy-delay product (EDP). In general, targeting maximizing total energy efficiency predominantly makes the optimal tiling for all layers to be E-T, with some being B-T, which is well expected. Similarly, targeting minimizing overall runtime makes the optimal tiling R-T for most layers. The trend on the optimization of EDP target is less visible, potentially due to the fact that both the variable tiling and accelerator hardware are simultaneously optimized.

Table 4.4 details the overall runtime, energy dissipation, and EDP resulted from the optimizations performed in Tables Table 4.2 and 4.3. We also report the results of using each of the two reference tiling schemes ref* [73] and ref** for all layers. For completeness of comparison, the results based on a fixed default array size (16X16) are reported to demonstrate the impact of hardware optimization. It is evident that the overall performance improves with the chip area. The proposed joint optimization improves EDP by up to 16.7X and by up to $282,000$X when compared with the variable tiling ref* and ref** without hardware optimization, respectively.

## 4.4 Summary and Discussions

Recognizing the need for developing SNN specific dataflows, we propose holistic reconfigurable dataflow optimization for systolic array acceleration of spiking convolutional networks (S-CNNs). As in many recent DNN accelerator works [16, 19, 74], we leverage systolic array architectures for efficient parallel processing with high spatiotemporal locality and compute density [75, 76].

Our main contributions are: 1) We show that parallel spatiotemporal computation over the six-dimensional space of input, weight, output, and temporal data involves significant complications in balancing between data reuse/storage of different data types and processing element utilization. 2) We investigate how the tiling strategy, in par-

Table 4.4: Normalized runtime, energy dissipation, and EDP under different optimization targets and hardware area constraints for VGG-16 and Alexnet. The SNN dataflow from [30] is denoted by ref*.

| **VGG16** | **AREA = 8mm$^2$** | | | | | **AREA = 16mm$^2$** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Optimization target** | **Hardware optimization** | ***Result*** | | | **EDP Improvement** | **Hardware optimization** | ***Result*** | | | **EDP Improvement** |
| | | Runtime | Energy | EDP | | | Runtime | Energy | EDP | |
| [a]None (ref**) | No (16X16) | 4079 | 3703 | 2x10$^5$ | 0.0005X | No (16X16) | 4024 | 3676 | 2x10$^5$ | 0.0003X |
| [b]None (ref*) | No (16X16) | 100 | 100 | 100 | 1X | No (16X16) | 89.16 | 76.91 | 64.72 | 1X |
| Runtime | No (16X16) | 11.76 | 178.64 | 20.93 | 4.78X | No (16X16) | 11.04 | 139.98 | 16.49 | 3.92X |
| Energy | No (16X16) | 16.84 | 63.27 | 10.25 | 9.76X | No (16X16) | 15.89 | 59.47 | 9.376 | 6.98X |
| Runtime | Yes (24X24) | 9.26 | 174.62 | 16.33 | 6.12X | Yes (32X32) | 6.04 | 152.28 | 8.843 | 7.32X |
| Energy | Yes (32X32) | 21.27 | 39.09 | 7.692 | 13.0X | Yes (40X40) | 16.72 | 23.84 | 3.874 | 16.7X |
| EDP | Yes (32X32) | 20.93 | 39.81 | 7.667 | 13.1X | Yes (40X40) | 16.72 | 23.84 | 3.874 | 16.7X |

| **Alexnet** | **AREA = 8mm$^2$** | | | | | **AREA = 16mm$^2$** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Optimization target** | **Hardware optimization** | ***Result*** | | | **EDP Improvement** | **Hardware optimization** | ***Result*** | | | **EDP Improvement** |
| | | Runtime | Energy | EDP | | | Runtime | Energy | EDP | |
| None (ref**) | No (16X16) | 4839 | 36044 | 3x10$^6$ | 0.00003X | No (16X16) | 4736 | 9600 | 7x10$^5$ | 0.0001X |
| None (ref*) | No (16X16) | 100 | 100 | 100 | 1X | No (16X16) | 87.20 | 93.38 | 75.17 | 1X |
| Runtime | No (16X16) | 19.87 | 67.57 | 11.91 | 8.40X | No (16X16) | 15.76 | 65.59 | 9.521 | 7.90X |
| Energy | No (16X16) | 20.29 | 66.98 | 11.84 | 8.45X | No (16X16) | 15.97 | 64.96 | 9.434 | 7.97X |
| Runtime | Yes (24X24) | 19.85 | 69.88 | 13.05 | 7.66X | Yes (32X32) | 13.73 | 65.30 | 8.137 | 9.24X |
| Energy | Yes (32X32) | 28.44 | 55.95 | 18.70 | 5.35X | Yes (40X40) | 19.65 | 44.30 | 10.01 | 7.51X |
| EDP | Yes (16X16) | 20.29 | 66.98 | 11.84 | 8.45X | Yes (32X32) | 14.50 | 52.03 | 7.009 | 10.7X |

[a] Without dataflow optimization, using single tiling strategy (ref**) for all layers.
[b] Without dataflow optimization, using single tiling strategy (ref*) for all layers.

ticular the positioning of the temporal dimension, significantly impacts data movement, throughput, and energy efficiency. 3) We explore joint layer-dependent dataflow and accelerator hardware optimization to further boost performance and energy efficiency. 4) We develop an SNN dataflow simulator capable of analyzing the throughput and energy dissipation of systolic array accelerators to support systematic design space exploration while considering the inherent spatiotemporal characteristics of spiking neural computation.

We demonstrate how the tiling strategy can be reconfigured on a layer-by-layer basis and jointly optimized with the accelerator hardware to achieve large gains in throughput and energy efficiency. Furthermore, an SNN dataflow simulator has been developed to aid systemic design space exploration. The proposed techniques are able to improve EDP of the accelerator by several orders of magnitude and more than one order of magnitude over a non-optimized and existing SNN dataflow, respectively.

# Chapter 5

# Parallel-Time-Computation for Spiking Neural Computation

In this chapter, we aim to develop a systolic-array architecture for general SNN models consisting of densely connected and convolutional spiking layers with the flexibility in employing various rate and temporal codes. We propose two key techniques to enable spike-based computation while efficiently exploring unstructured firing activity sparsity in both space and time. First, the (sparse) firing activity of one (active) synaptic neuron over multiple time points are packed into one time window $TW$. The integration of such sparse firing inputs into the membrane potential of a connected post-synaptic neuron over the given $TW$ is referred to as a *time batch*, which is mapped to a processing element (PE) on the systolic array. This gives rise to the proposed *parallel time batching* (PTB) technique by which multiple time batches are processed simultaneously on the array. On top of PTB, we further propose a *spatiotemporally-non-overlapping spiking activity packing* (StSAP) technique to identify and combine time batches whose spike inputs are non-overlapping either in time or space. This work provides a novel solution to address limitations in stereotypical (time-serial) approaches for energy-efficient dataflow

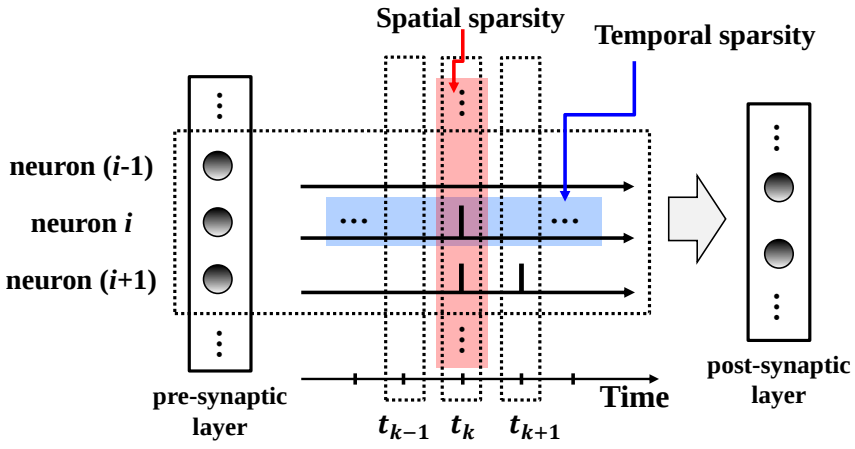


Figure 5.1: Spatial and temporal sparsity emergent in SNNs.

and parallel processing in time-domain towards memory-intensive SNN accelerators.

## 5.1 Challenges of SNN Accelerators

While SNNs are promising brain-inspired models of computation, complex spatial and temporal interactions in data movement and computation hinder their hardware acceleration. Firing sparsity emergent in both spatial and temporal domains provides an opportunity for building efficient SNN accelerators. However, tapping to this opportunity is challenging and requires tackling the unstructured nature of spiking data sparsity from which severe PE under-utilization and energy efficiency degradation may be resulted.

### 5.1.1 Spatial and Temporal Sparsity in SNNs

Unlike in conventional ANNs, information processing in SNNs takes place both spatially across different neurons and temporally through an operational period of multiple time points. Spatiotemporally sparse firing activities often arise in well-trained SNNs. However, such sparsity is usually irregular, as shown for a pair of adjacent presynaptic

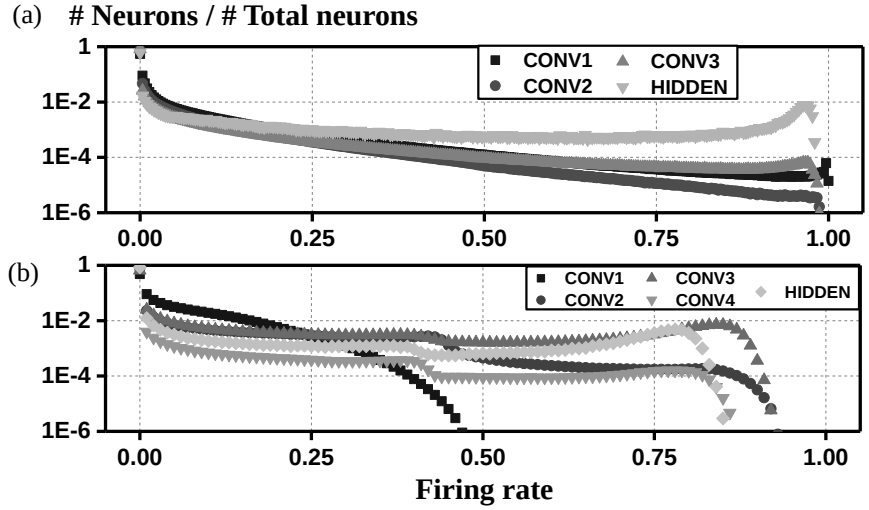


Figure 5.2: Normalized firing rate and distribution of neurons in (a): DVS-Gesture, and (b): CIFAR10-DVS.

and postsynatic layers in Fig. 5.1.

**Spatial sparsity**- At each time point $t_k$, not all neurons in the pre-synaptic layer fire; spatial sparsity can be leveraged to only fetch the data and process the computation associated with active pre-synaptic neurons at a given time point.

**Temporal sparsity**- Different neurons might fire different number of times within the same operational period; temporal sparsity can be exploited to avoid redundant computation and/or data movement at time points when a neuron is silent.

As one example, Fig. 5.2(a) and (b) show the normalized average firing rate distributions of two well-trained SNNs based on the neuromorphic DVS-Gesture [42] and CIFAR10-DVS [35] datasets, respectively. Only 0.0001% of neurons at the CONV3 layer of the DVS-Gesture model produce 150 spikes over 300 time points. Unlike the extreme temporal sparsity assumed in [20], neurons in practical high-performance SNNs may fire more than once. On the other hand, they exhibit a great deal of unstructured sparsity such that neglecting such sparsity as in [21] abandons opportunities for performance improvements.

Table 5.1: Summary of key features in existing and our SNN accelerators.

| | **Applicability**[a] | **Parallel**[b] **processing** | **Sparsity handling** | **Learning**[c] **Performance** |
|---|---|---|---|---|
| Ref* | High | No | Limited | High |
| Ref** | High | No | Yes | High |
| [20] | Low | No | Yes | Low |
| [21] | High | Limited | No | High |
| Ours | High | Yes | Yes | High |

Ref*: Conventional approach: [60, 62, 22]
Ref**: Industrial neuromorphic chips: TrueNorth [6], Loihi [7]
[a]: Applicability for general SNNs [b]: Parallel processing in time domain
[c]: Learning performance (achievable accuracy)

### 5.1.2 Existing SNN Accelerators

While holding a great deal of promise, neuromorphic SNN hardware accelerators have not been extensively studied. **Time-serial processing in SNN accelerators** - The most natural approach for SNN acceleration is to emulate the evolution of neural membrane potentials and firing activities time point by time point in a sequential manner. This has been adopted in several SNN accelerators [60, 62, 22]. We refer to this time-serial processing approach as the *conventional approach* in this paper. In essence, this conventional approach follows the paradigms of non-spiking ANN accelerators for processing at each time step. Time-serial processing can introduce significant inefficiency due to iterative weight data access and low utilization efficiency, as will be discussed in Fig. 5.5. From the memory point of view, the time-sequential process requires alternating access to different weight matrices for different time points.

**Other Existing SNN Accelerators** - [20] proposed an efficient method to accelerate temporally-encoded SNNs. However, [20] only considers a very constrained case of extreme temporal sparsity that prevents its application to more general SNN in which neurons fire more than once and may employ other types of rate and temporal coding for high accuracy. [21] introduced dataflow optimization for SNNs while operating in

the time domain. However, the tiling technique in [21] does not consider firing sparsity, has limited weight reuse, and may lead to a significant PE under-utilization and comprised energy efficiency. Table 5.1 summarizes the key features and limitations of the prior works and our work. Importantly, our work is the first work that proposes parallel acceleration of SNNs while incorporating spatiotemporal sparsity.

## 5.2 Proposed Architecture

The proposed systolic-array SNN accelerator architecture is supported by two novel techniques, namely, *parallel time batching* (PTB) for parallel acceleration in both space and time, and *spatiotemporally-non-overlapping spiking activity packing* (StSAP) to further improve array utilization. Both techniques are geared towards efficient exploitation of unstructured firing sparsity.

### 5.2.1 Overview of the Proposed Architecture

The overall architecture is composed of a tiled array of processing element (PE) with unidirectional links to form a systolic array along with memories for data storage, as illustrated in Fig 5.3(a). As in Fig. 5.3(b), each PE consists of 1) an accumulate (AC) unit, 2) a comparator, 3) a small scratch-pad memory and 4) simple controller logic. While non-spiking accelerators generally adopt multiply-and-accumulate (MAC) units, simpler AC units are employed to accumulate weights under (binary) input spikes. To minimize the data movement overhead of multi-bit partial sums (Psum), one of the main bottlenecks of SNN accelerators [21], the scratchpad in each PE stores the Psums for a given time window (TW). We adopt three levels of the memory hierarchy: 1) an off-chip RAM, 2) a global buffer, and 3) a double buffered L1 cache [69], [44]. The 2-D systolic array exploits spatial and temporal parallelisms for which spike input and weight data

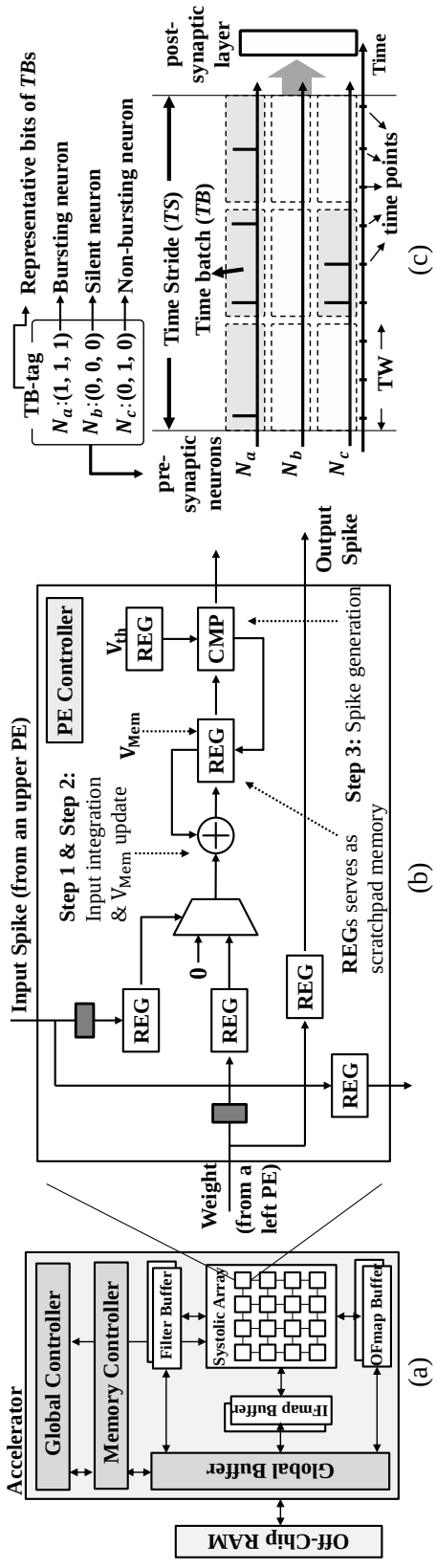


Figure 5.3: (a): Overall architecture. (b): Simplified schematic representation of the processing element (PE) in systolic array. (c): Schematic representation of *time point*, *time batch* ($TB$), $TB$-tag and *time stride* ($TS$).

propagate vertically and horizontally across the array. The membrane potential update and spike output generation for each neuron involves simple local computation. In the rest of the chapter, we focus on synaptic input integration, the dominant complexity of SNN acceleration.

### 5.2.2 Time Batch (TB) and TB-tag

***Time stride*(TS):** Full range of time points over which the SNN operates. TS is split into multiple time windows (TWs).

***Time window*(TW):** One TW packs the firing activity of one (active) synaptic neuron over multiple time points.

***Time batch*(TB):** The integration of such sparse firing inputs into the membrane potential of a connected post-synaptic neuron over the given $TW$, which is mapped to a processing element (PE) on the systolic array. A TB corresponds to the basic unit of workload assignable to a PE.

In Fig. 5.3(c), for example, the pre-synaptic neuron $a$ ($N_a$) generates three TB workloads within the given TS. A TB-tag is associated with TBs to indicate the existence of input spikes in the corresponding time windows: each bits in TB-tag is set to 1 if there is input activity; otherwise it is set to 0. We classify the pre-synaptic neurons into three categories based on their TB-tags as shown in Fig. 5.3(c). If the TB-tags of a neuron are all-zeros, i.e., a neuron does not fire throughout all $TW$s in $TS$, we call this neuron a silent neuron, e.g., Neuron $b$ ($N_b$) in Fig. 5.3(c). We skip silent pre-synaptic neurons to avoid redundant processing. . We call a neuron a bursting neuron if its TB-tags are all-ones, meaning that it fires at all TWs, e.g., Neuron $a$ ($N_a$) in Fig. 5.3(c). All other neurons are defined as non-bursting neurons, e.g., Neuron $c$ ($N_c$) in Fig. 5.3(c).

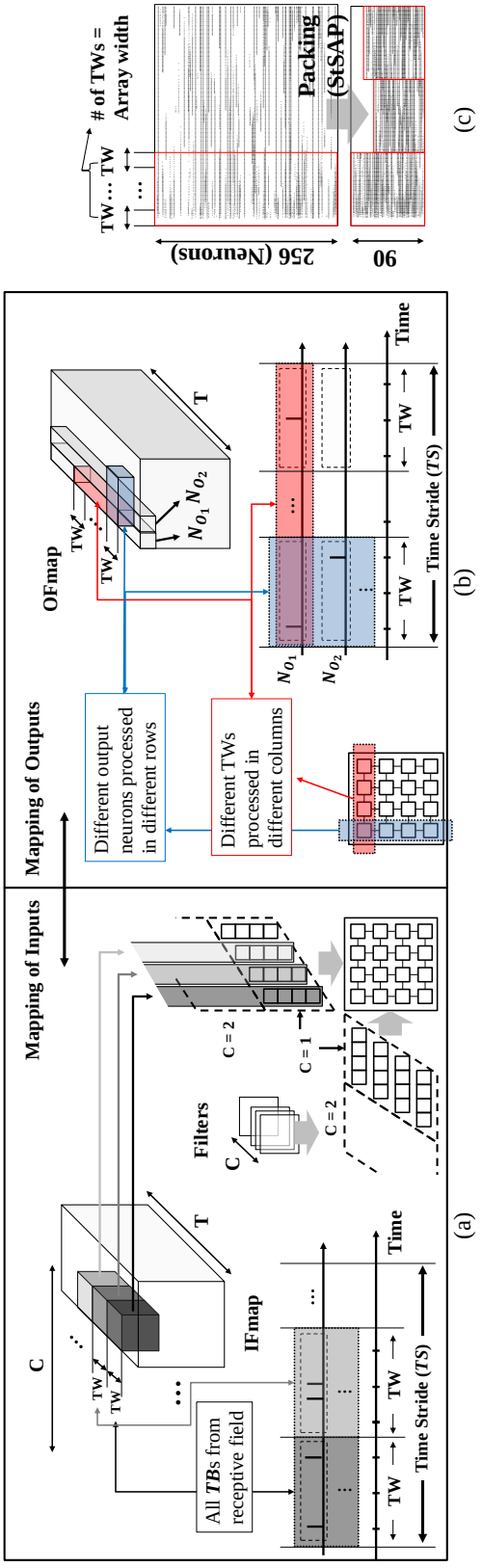


Figure 5.4: Mapping of the (a): inputs and (b): outputs into the systolic array. (c): Example of enhanced spike input density in DVS-Gesture dataset with temporally-non-overlapping spikingactivity packing (StSAP).

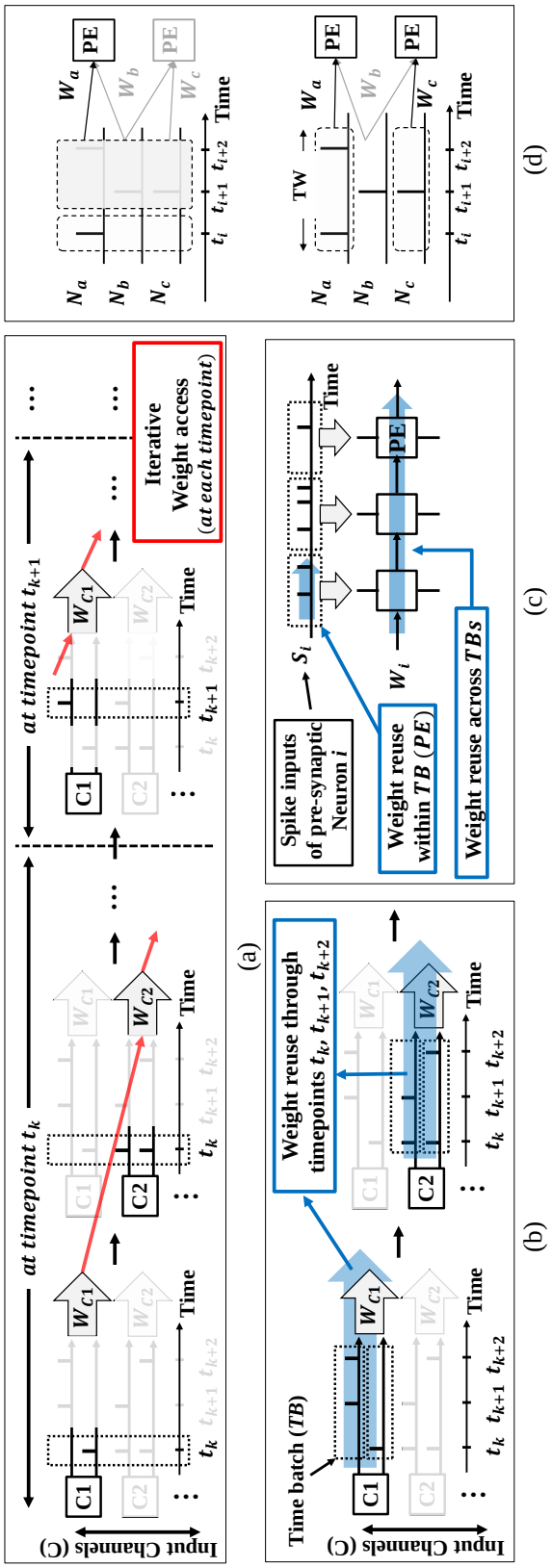


Figure 5.5: Simplified schematic representations of (a): Conventional approach which lacks parallel processing in time domain (executions are performed in time-serial manner). It requires alternating weight access. (b): Proposed approach using parallel time batching (PTB). (c): Weight reuse within, and across TBs. (d): Hiding the absence of spike with TB (grouped spiking activity).

### 5.2.3 Parallel Time Batching (PTB)

Instead of operating in a time-sequential manner as in conventional approach, the proposed architecture accelerates multiple TBs for multiple post-synaptic neurons in different rows, and for different TWs in different columns, in parallel.

#### Mapping Inputs/Outputs

We assign a single PE for processing computations within a given TB of a targeted post-synaptic neuron over the time points in the corresponding TW. Fig. 5.4(b) illustrates how the computations of an OFmap are mapped to the PEs. Each row of the array is utilized to compute output activation of a single post-synaptic neuron for different TWs with multiple time-batched inputs (TBs). PEs in each column process the same $TW$ but for different post-synaptic neurons. Spike inputs into the array are assigned according to the mapping of PEs for post-synaptic neurons, as shown in Fig. 5.4(a). In the array iteration that executes computations for targeted post-synaptic neurons over the TS, the IFmap and filter data of the TBs in range of TS from the corresponding receptive fields are fetched into the array.

Under PTB, the computations of a single PE for a CONV layer can be expressed by modifying (2.1) $\sim$ (2.4) as:

**(For a specific post-synaptic neuron)**

**Step A: For all input neurons in receptive field -**

Integration of synaptic inputs for a given TW, from time point $t_k$ to $t_{k+TW-1}$:

$$\begin{aligned} p_{ji}^{O}[t_k, .., t_{k+TW-1}] &= w_{ji} \times s_j^{RF}[t_k, .., t_{k+TW-1}] \\ p *_i^{O} [t_k, .., t_{k+TW-1}] &= \sum_{j=1}^{M^{RF}} p_{ji}^{O}[t_k, .., t_{k+TW-1}] \end{aligned} \tag{5.1}$$

**Step B:** Membrane potential update & Conditional spike output generation for a given TW, from $t_k$ to $t_{k+TW-1}$ (for $m = 0, 1, \ldots, (TW-1)$).

$$
\begin{aligned}
v_i^O[t_{k+m}] &= p *_i^O [t_{k+m}] + v_i^O[t_{k+m-1}] \\
s_i^O[t_{k+m}] &= \begin{cases} 1, \text{if } v_i^O[t_{k+m}] \geq V_{th}^O : v_i^O[t_{k+m}] = 0 \\ 0 \qquad \text{else :} \qquad v_i^O[t_{k+m}] = v_i^O[t_{k+m}] \end{cases}
\end{aligned} \tag{5.2}
$$

where $p*_i^O$ denotes the integrated partial sum of all spike inputs from receptive field in output neuron $i$. All the other expressions follow the definition described in (2.1) $\sim$ (2.4). PTB groups **Step 1** in (2.1) across multiple time-points with the batch size TW as in (5.1). With mapping of the computations into PEs as described in Fig. 5.4, PTB enables parallel processing in both 1) space - for output neurons at different positions, and 2) time - executing different TWs, for **Step A** in (5.1), the dominant complexity.

**Energy reduction**

To exploit unstructured firing sparsity as shown in Fig. 5.1, PTB minimizes the weight access with two different types of reuse, as shown in Fig. 5.5(b) and (c). First, PTB reduces alternating accesses to different weights, which is however inevitable in the conventional time-serial processing as shown in Fig. 5.5(a). In the latter approach, the array cycles through all required weight data to compete the processing of all post-synaptic neurons at $t_k$. At the next time point $t_{k+1}$, the above process is repeated without allowing weight data sharing between the two time points. Differently, PTB processes each TB while allowing the same weight data associated with the presynaptic neuron to be reused across the multiple time points within the TB, as shown in Fig. 5.5(b). Furthermore, as PEs in the same row of the array performs computations of different TWs for a given post-synaptic neuron; the same weight data is reused across these PEs,

as illustrated in Fig. 5.5(c). In summary, PTB enables weight data reuse within each TB (PE) and across different TBs (PEs).

#### Utilization

PTB alleviates severe under-utilization which originates from the inactive processing elements with a silent receptive field. Since TB packs multiple presynaptic spikes into a TB and assign the entire TB to a PE, it reduces the number of idling PEs due to the larger temporal granularity defined by the time window (TW) size. For example, the only spike in $t_i$ from Neuron $a$ ($N_a$) results in degradation of utilization in the conventional approach while PTB hides the absence of spikes within the packed input spikes in the PB as described in Fig. 5.5(d).

### 5.2.4 Spatiotemporally-non-overlapping Spiking Activity Packing (StSAP)

To further leverage the utilization efficiency, we propose a novel compression scheme, dubbed spatiotemporally-non-overlapping spiking activity packing (StSAP), which packs sparse spike inputs into a denser format. The key idea here is to combine non-overlapping spike inputs based on the TB-tags, which enables simultaneous scheduling/processing of the non-overlapping spiking activities and hence increases the utilization efficiency.

#### Packing strategy

Recognizing the sparsity emergent across different neurons and TWs, we combine TBs of non-bursting neurons. First, we trim out silent presynaptic neurons with all-zero TB-tags without fetching them to the array. By removing silent neurons which do not fire throughout all TWs, we compress the sparse input firing activity spatially. Then,

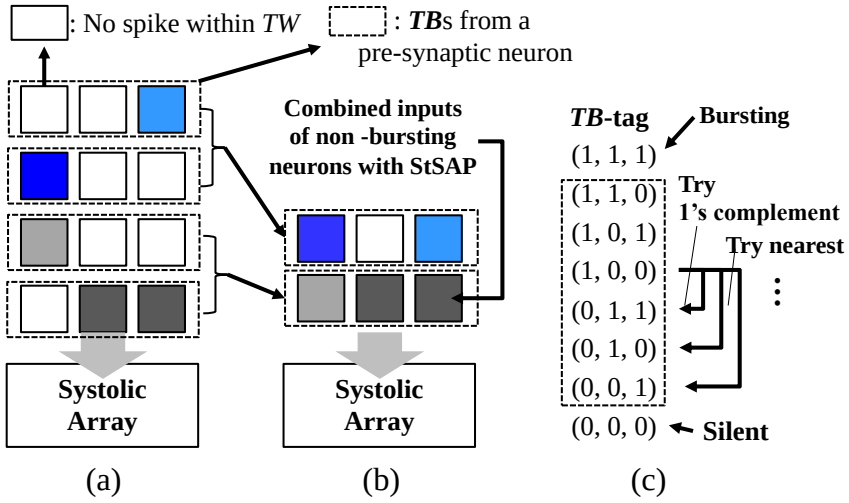


Figure 5.6: Schematic representation of StSAP. Mapping of the spike inputs from non-bursting neurons (a): without StSAP, (b): with StSAP. (c): Greedy policy applied to find nearest 1's complement based on TB-tag.

we use plain spike inputs for the bursting neurons, as bursting neurons generate non-zero TBs across TS. Finally, StSAP is applied to the non-bursting neurons to explore temporal sparsity. For simple and efficient packing, we adopt a greedy combining policy for searching TBs that can be packed together. Starting from a given TB with its TB-tag, StSAP first tries packing with 1's complements and finds the nearest non-overlapping TB-tags if the exact 1's complement does not exist, as shown in Fig. 5.6(c). We limit the number of neurons for packing to two to simplify the packing process.

### Utilization efficiency

Recognizing the unique sparse nature of SNNs, StSAP manages the spatial and temporal sparsity of the spike inputs in the time-domain based on TB-tags. In Fig. 5.6, for example, four TBs from non-bursting neurons are packed into two with StSAP. Fig. 5.4(c) demonstrates the packing of realistic spiking input data, revealing significantly densified input by the proposed StSAP. Simultaneous scheduling of non-bursting neurons alleviates severe PE under-utilization, and is particularly well-suited for sparse spiking computa-

tions.

Importantly, StSAP directly points to the actual activation values and fundamentally different from the existing packing strategies for DNNs [19, 18, 77, 78, 79, 80, 81]. For example, [18] and [19] proposed packing of sparse columns of a convolutional filter matrix into a denser weight matrix. However, as the time dimension is not incorporated in DNNs, [18] and [19] primarily focused on combining non-zero weights into a denser format. However, StSAP focuses on unique features of sparse spike inputs over time and explores the unstructured sparsity with TB-tags.

## 5.3 Experiments and Results

We evaluate the performance of the proposed architecture focusing on the impact of the proposed PTB and StSAP described in section 5.2 based on the setups described in the following section. The performance of the proposed architecture hinges on optimizing tradeoffs between reuse of multi-bit weight and binary input activation data, storage of multi-bit partial sums, and array utilization, which are also dependent on structures of layers of the spiking neural network to be accelerated. There exist two critical architectural parameters, i.e., time window ($TW$) size and systolic array dimension that impact the above tradeoffs. We first examine how to near-optimally choose array dimension, and then comprehensively evaluate the proposed architecture based on realistic S-CNN networks.

### 5.3.1 Architecture Specifications and Benchmarks

We evaluated the proposed idea with the simulator introduced in chapter 4 where the user specified inputs and architecture specifications are summarized in Table 5.2 and Table 5.3, respectively.

Table 5.2: A high-level overview of the input parameters.

| Input Parameter | Description |
|---|---|
| Array configuration | Array width/height, size of the scratch-pad in each PE |
| Memory configuration | Size of the memory, partitioning of the memory for each type of data, at each level |
| Time Window ($TW$) Size | Ranging from each time-point ($TW$=1) to cover all time-points with given array width |
| Packing | Packing the non-bursting neurons or use plain inputs |
| Network Structure | Number of layers, number of the neurons in each layer, layer type (CONV, FC) |

Table 5.3: Architecture specifications.

| Components | Proposed Architecture |
|---|---|
| Number of PEs | 128 |
| ALU in PEs | Adder, Comparator - 8-bit |
| Global Buffer Size | 54KB |
| L1/Scratchpad Size | 2KB / 96 × 8-bit |
| DRAM Bandwidth | 30GB/sec |
| Bit precisions | Weight/Membrane Potential - 8-bit<br>Input/Output Spike - $TWS$ × 1-bit<br>($TWS$: $TW$ size) |

### 5.3.2 Optimization of Array Dimension

As discussed, without using StSAP, PEs in each row of the systolic array perform computations for the same post-synaptic neuron at different time windows while PEs in each column process different post-synaptic neurons for the same TW. Having more columns by increasing the array width processes each post-synaptic neuron over a longer overall time span, resulting in more multi-bit weight data reuse. When the PE count is fixed, this will lead to a fatter array that processes fewer post-synaptic neurons per array iteration. This has the downside of reduced input activation data reuse across different post-synaptic neurons. Oppositely, skinner arrays encourage input data reuse among different post-synaptic neurons while reducing weight data reuse across multiple time points. $TW$ size plays an important role in the tradeoffs between input/weight data

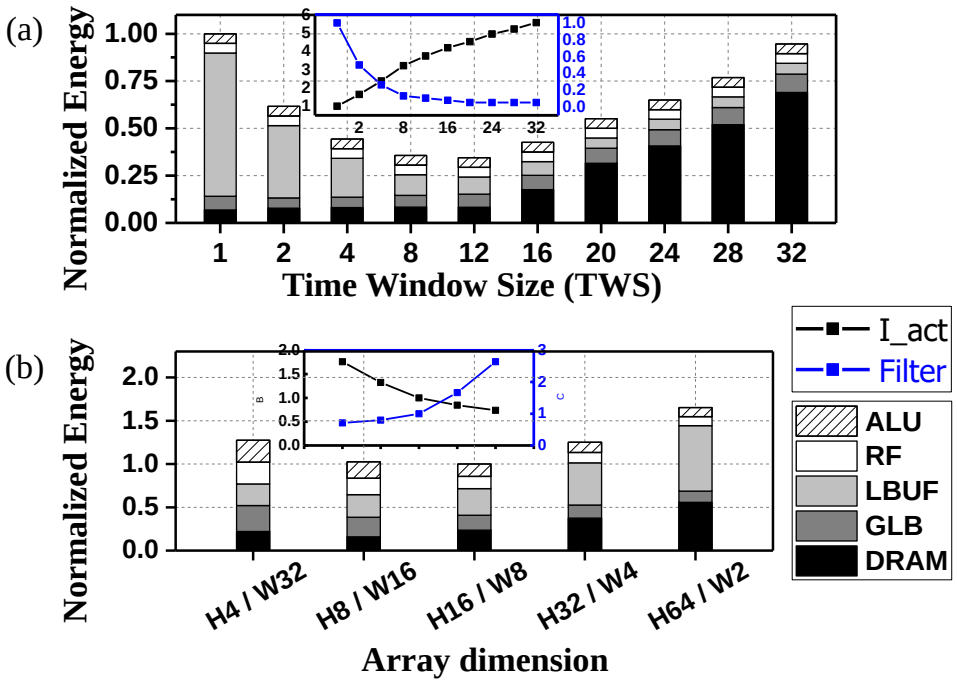


Figure 5.7: Energy dissipation breakdown of CONV2 in DVS-Gesture with different (a): TW size. (b): array size (TW=8).

reuse and Psum data storage, and impacts the optimal array dimension.

### Impact of Time Window Size

PTB improves weight data reuse by grouping multiple time-points into a single TW, maximizing the weight data sharing opportunity within each time window. Movement of binary input activation data tends to a lesser problem compared to that of other multi-bit data types. No data compression is applied to binary input data when it is fed onto the systolic array to avoid the overheads of compression and decompression. To this end, while wider TWs improve the reuse of multi-bit weight data on the array, but there is a tendency of packing an increased number of zero-valued input activations within the TW, incurring higher overheads of data fetching to and input data storage on the array. Also, wide TWs stretch the integration of synaptic inputs over many time points, producing more multi-bit partial sum data that must be stored.

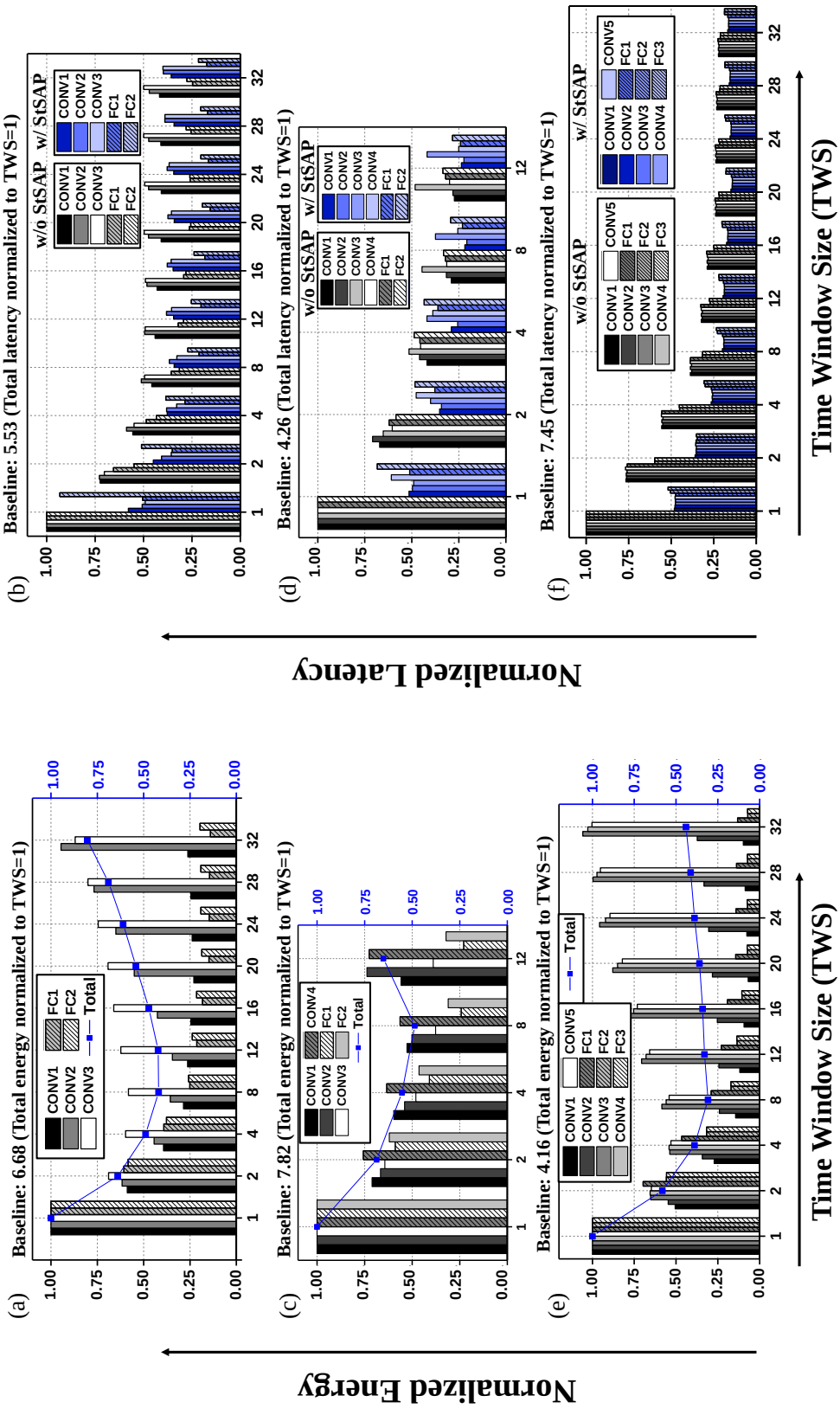


Figure 5.8: Normalized energy dissipation and latency of layers with different TW sizes, with- and without StSAP, in each dataset. **(a),(b): DVS-Gesture, (c),(d): CIFAR10-DVS, and (e),(f): Alexnet**. PTB with non-optimized TW size (TWS=1) improves the total energy dissipation and latency by DVS-Gesture: 6.68X and 5.53X, CIFAR10-DVS: 7.82X and 4.26X, and Alexnet: 4.16X and 7.45X, over the baseline.

We use the CONV2 layer trained for the DVS-Gesture dataset as a representative layer configuration to evaluate the impact of $TW$ size in Fig. 5.7(a), which clearly shows decreased weight access and increased input activation data access at larger $TW$ sizes. Typically, a $TW$ size of 8 is near optimal, which is further discussed in Section 5.3.3.

#### Near-Optimal Array Dimensions

While fixing the $TW$ size to 8, we examine how array dimension impacts the energy dissipation when accelerating the representative CONV2 layer of the DVS-Gesture dataset. Fig. 5.7(b) shows the normalized energy dissipation and the tradeoff between weight (filter) and input activation data access (inset) under different array dimensions when the PE count is fixed at 128. The array dimension of 16×8 is typically a near-optimal choice, which is adopted for the rest of the paper.

### 5.3.3 Comprehensive Evaluation

While fixing the array dimension, we jointly optimize the proposed PTB and StSAP techniques with the other key architectural parameter $TW$ size and compare our architecture with the baseline. We adopt the approaches in [21] as our **baseline**. We also examine the dependencies on SNN layer structures to shed light on how the proposed techniques exploit sparsity of spike data and the granularity of time-domain processing to improve the overall performance.

#### SNN Layer-Dependent Tradeoffs

Reuse of multi-bit weight and binary input activation data, storage of multi-bit partial sums, and array utilization can be traded off by altering the granularity of time batching, i.e., the $TW$ size. The resulting optimal tradeoffs have a strong dependency on the

structure of spiking neural network layers.

In general, fully-connected (FC) layers favor larger $TW$ sizes as the multi-bit weight data have a greater footprint and the overhead of weight data movement tends to dominate. The number of input channels in a convolutional (CONV) layer determines the amount of IFmap data that must be fetched to the array for each time batch. The lateral dimension of the filters determines the sheer amount of weight data that must be fetched. Therefore, CONV layers with many few input channels and large filter sizes benefit from enlarged $TW$ sizes as the overhead of the input activation data movement is more than compensated by the improved weight data reuse. The opposite can be said for CONV layers with many input channels but small sized filters.

**Performance of PTB**

PTB offers significant benefits in terms of latency and energy dissipation across most CONV and FC layers in the three SNN models compared with the baseline as demonstrated in Fig. 5.8. The impact of the TW size, on the other hand, varies from layer to layer as a result of changing tradeoffs between weight (filter) and input (IFmap) data movement.

**Energy dissipation:** Energy dissipation in CONV layers is reduced as the TW size increases to a certain point from which any further increase in the TW size degrades energy efficiency. In typical S-CNNs, early CONV layers and FC layers have large sized filters or a great amount of weight data while the number of input channels tends to be limited. This is in contrast to later CONV layers which are featured by small-sized IFmaps, but very importantly many input channels. As such, FC layers favor large TW sizes across the board, and the same is for early CONV layers, e.g., layer CONV1 in Fig. 5.8 (e). The figure shows the energy dissipation of different layers in the AlexNet model along with the total energy. The benefit from increasing the TW size is even

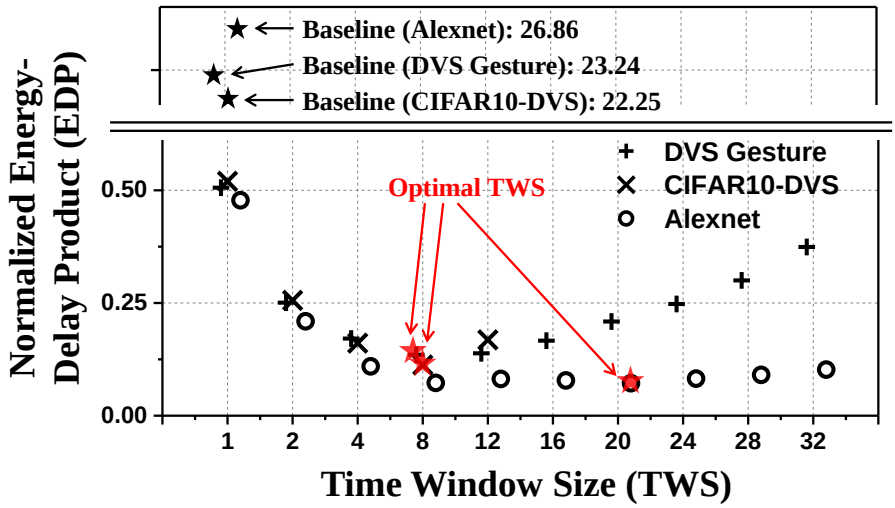


Figure 5.9: Total energy-delay product (EDP) of three different benchmarks. EDP values are normalized to the baseline result, which exclude merging and TW size optimization.

more pronounced for early CONV layers than FC layers. On the other hand, for later CONV layers such as CONV4, energy dissipation is initially reduced by applying a small window size; however, going beyond a TW size of 4 degrades energy efficiency due to the comprised input data movement as shown in Fig. 5.8 (e).

**Latency:** We observe a clear improvement of latency by using PTB in all three networks, as shown in Fig. 5.8. As discussed in Section 5.2, PTB mitigates systolic array under-utilization by packing multiple input activities into time batches, reducing the idling of the PEs. In general, applying a larger TW size further reduces the number of idling PEs and hence latency. However, it is possible to experience a very modest increase of latency for certain layers at large TW sizes. This is caused by the fact that a fewer number of time points are packed into time batches towards to the very end of the operational time period, introducing idling PEs while processing the latest time batches. Array utilization is further improved by the proposed StSAP, discussed next.

### Performance of StSAP

On top of PTB, StSAP offers further array utilization and latency improvements for all most all CONV and FC layers, as shown in Fig. 5.8. PTB improves PE utilization by packing multiple input spikes in a given time batch, which is to be processed on a PE. StSAP takes one step further to analyze the patterns of sparse spiking inputs. A set of time batches that are non-overlapping either in time or space are processed simultaneously on the array, further reducing PE idling and overall latency. It shall be noted that overlaps between time batches may increase with the TW size, which may comprise the benefits of StSAP. Moreover, the choice of TW size impacts the performance of the underlying PTB based on which StSAP is applied. As a result, the overall performance of StSAP is also layer dependent; overly large TW sizes can degrade the benefit of StSAP, and hence the latency.

### EDP evaluation

We use energy-delay product (EDP) to simultaneously consider latency and energy dissipation for evaluation of the overall system performance. We multiply the total energy consumption and the total amount of time for executing (latency) at each layer, and then integrate EDP values of all layers to calculate total EDP. Fig. 5.9 shows the normalized EDP of three different networks with varying TW sizes. The baseline is based on the approach proposed in [21], which exploits limited temporal parallel processing without handling the sparsity. Both DVS-Gesture and CIFAR10-DVS models show a clear optimal choice at TW size of 8. The optimal trade-off point of TW size is larger for the AlexNet model. In AlexNet, the energy dissipation of later CONV layers is minimized by choosing a proper TW size while other layers are benefited continously with increasing TW size, as shown in Fig. 5.8 (e). Since the overheads of later CONV

Table 5.4: Performance comparison of ANNs and SNNs.

| Workload | ANN | SNN |
|---|---|---|
| MNIST | 99.80% [83], 99.04% [29] | 99.62% [3], *99.53% [39] |
| CIFAR10 | 90.49% [84], 93.75% [85] | *91.41% [39], 93.58% [86] |
| DVS-Gesture | 92.01% [85], 93.75% [85] | 95.49% [87], *93.75% [39] |
| CIFAR10-DVS | 31.00% [88], 52.40% [89] | 58.10% [11], *56.86% [39] |

*: This work.

layers constitute to a small portion of the total overhead, the overall optimal TW size of AlexNet is larger than those of the other two models. With the optimized TW sizes, the proposed architecture delivers 172X, 198X and 373X EDP improvement over the baseline for accelerating the DVS-Gesture, CIFAR10-DVS and AlexNet models, respectively. On average, our architecture delivers 248X EDP improvement.

## 5.4 Summary and Discussions

SNNs have shown great potentials and results in both energy efficiency and performance [3, 39, 11]. SNNs can show better efficiency over ANNs when key factors, i.e., data reuse, sparsity handling and repeated operations through time, are efficiently managed as discussed in [82, 20]. We discuss broader impacts and promises of our work.

**Machine learning performance:** As shown in Table 6.3, SNNs achieved comparable performance to ANNs with promising training algorithms and network structures. Especially, with advantages in handling spatiotemporal information, SNNs can achieve better performance over ANNs in neuromorphic datasets. Apart from machine learning performance, the main focus of our work is to develop efficient architectures and techniques for accelerating SNNs.

**Hardware acceleration:** [20] showed outperforming energy efficiency over ANNs [15] with constraint that each neuron fires at most once, exploring extremely high temporal sparsity and low bit resolution. However, extreme sparsity often suffer from low performance. General SNNs employing various rate and temporal codes lack efficient architectures and techniques for hardware acceleration, which is addressed in our work.

For accelerating CIFAR10 dataset, our approach delivers 14.6X and 3.3X improvement over the ANN counterpart [44, 84] for energy dissipation and latency, respectively, as shown in Fig. 5.10(b). We adopted same network structure in [39, 84] and architecture specifications using [44] for fair comparison. We adopted the training algorithm in [39] and applied actual spiking activities of a well-trained network to our pre-determined architecture specifications. Our work addresses the main source of inefficiency in SNNs, i.e., iterative and irregular patterns of data access repeated through time, and presents promising methods to outperform ANNs.

**Scalability of PTB w.r.t. sparsity level:** As shown in Fig. 5.10(b), the benefit of PTB depends on temporal sparsity level: 1) low sparsity (high firing rate) increases PTB benefits, 2) high sparsity (low firing rate) decreases PTB benefits. Still, PTB improves energy efficiency by 28X even for the rare case of 1% firing rate. Importantly, as shown in 5.10(a), firing rate of well-trained networks ranges from 1$\sim$15% in practice for which PTB can significantly improve energy efficiency.

**Generality of PTB:** PTB pre-calculates the synaptic input integration (S-I) step for multiple time-points in parallel prior to the rest step. The S-I step can be performed without knowing the state of the post-synaptic neuron and hence without violating causality. Thus, as shown in Fig. 5.10(c), PTB is applicable across: 1) all typical spiking neuron models (LIF, IF, etc.), 2) all layer structures (fully-connected, convolutional, recurrent, etc.), 3) general SNNs with various layer types, and 4) SNN accelerators of any given array/memory size with flexible choice of TW size. Layerwise fine-grained optimization

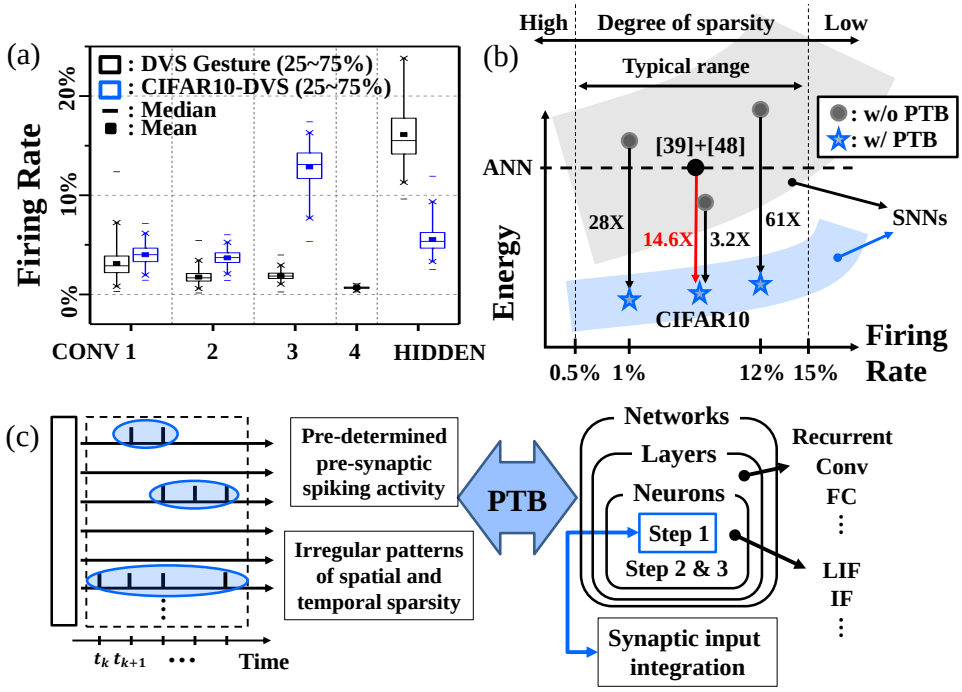


Figure 5.10: (a) Firing rate of well-trained networks. (b) PTB significantly improves energy efficiency across wide range of sparsity-levels. With PTB, SNN showed better result than ANN. (c) PTB supports diverse family of spiking models.

is possible if the optimal TW size is chosen offline, based on the given architecture, sparsity-level and layer type.

In summary, the proposed architecture is built upon a novel *parallel time batching* (PTB) technique and a *spatiotemporally-non-overlapping spiking activity packing* (StSAP) strategy. PTB introduces parallel acceleration of *time windows* (TWs) that incorporates multiple time-points, and significantly improves energy efficiency and under-utilization by reducing iterative data access and idling of processing units. StSAP densifies the grouped input spikes (TBs) by combining non-bursting neurons with greedy policy, which further benefits the utilization efficiency of the array. We also observe that larger TW size does not always provide monotonic improvements, and hence perform a joint optimization of PTB and StSAP with varying TW sizes for different networks. Our experimental results provide insights for parallel acceleration with an optimal choice of

handling the fundamental trade-offs in SNNs. Experimentally, our work improves the energy-delay product (EDP) of the array accelerator for DVS-Gesture, CIFAR10-DVS, and AlexNet by 248X over a baseline, on average. Compared to ANN based accelerator, our approach improves EDP by 47X on the CIFAR10 dataset.

# Chapter 6

# Recurrent Spiking Neural Network Acceleration

## 6.1 Recurrent Spiking Neural Network Accelerators

### 6.1.1 Recurrent Spiking Neural Network

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 back-

propagation techniques[3, 39, 31], achieving state-of-the-art accuracy on commonly used spike-based neuromorphic image and speech recognition datasets such as MNIST [90], Fashion-MNIST [91], N-TIDIGITS [92], TI46 speech corpus [36], and Sequential MNIST and TIMIT [93]. For example, [3] proposed the Spike-Train level R-SNNs Backpropagation (ST-RSBP) algorithm for training deep R-SNNs. By directly computing the gradient of a rate-coded loss function defined at the output layer, this work presents near state-of-the-art accuracy for challenging speech and image datasets, such as TI46 [36] N-TIDIGITS [92] Fashion-MNIST [91], and MNIST [90]. [31] 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.

### 6.1.2 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 [20, 22, 94, 95, 96], particularly array-based accelerators [97, 62, 73, 98, 99] 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 [7, 6, 100]. For example, [97, 62, 73, 98] 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 re-

sides 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 [6, 7, 100]. [6, 7] 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 [6] is based on intra-core crossbar memory and long-range connections through an inter-core network and [7] adopt neuron-to-neuron mesh routing model [101]. [100] 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, [100] 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 [6] and Intel's Loihi with 130M synapses [7], 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 paper allows parallel compute over multiple

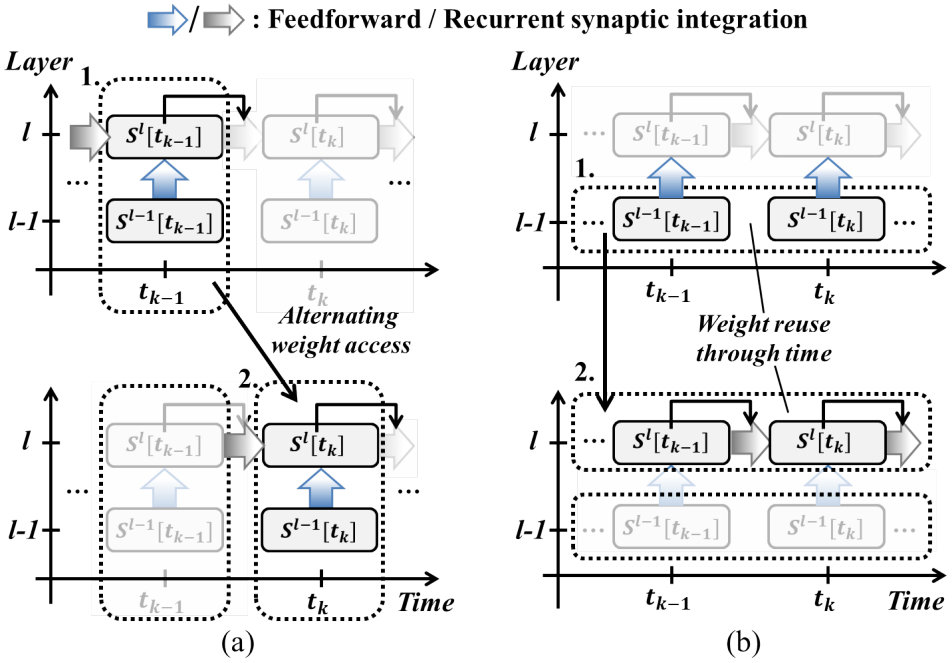


Figure 6.1: Schematic representation of (a): Time-serial processing in conventional SNNs, (b): Decoupling scheme to separate feedforward and recurrent steps. Layer $l$: Recurrent layer.

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 6.2, 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).

## 6.2 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.

### 6.2.1 Decoupled feedforward/recurrent synaptic integration

As discussed in Section 6.1, 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 Fig. 6.1(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 (2.5), 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 Fig. 6.1(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

Fig. 6.1(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.

$$\mathbf{f}_i^l[t_k, \cdots, t_{k+TW-1}] = \sum_{j=1}^{M^{l-1}} \mathbf{W}_{ji}^{F,l} \times \mathbf{s}_j^{(l-1)}[t_k, \cdots, t_{k+TW-1}] \tag{6.1}$$

**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 section 6.1 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 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 2.2, 2.4 and 2.6. Nevertheless, it is important to note that despite 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

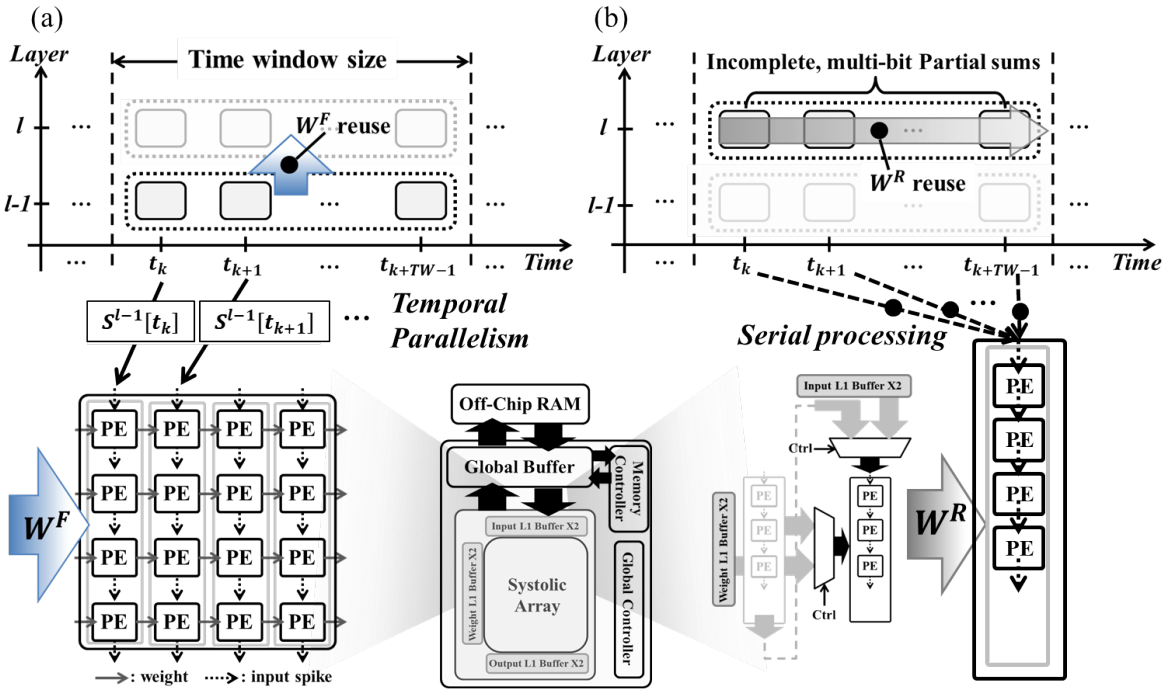


Figure 6.2: Overview of the proposed SaARSP architecture. (a): Array computation for feedforward integration (**Step A**, OS dataflow) (b): Array computation for recurrent integration (**Step B**, OS dataflow)

for both feedforward integration and recurrent integration, and are applicable to both feedforward and recurrent SNNs.

### 6.2.2 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 Fig. 6.1(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 the paper for comparison.

In contrast, Fig. 6.2 shows the overall SaARSP architecture, comprising controllers, caches, and a reconfigurable systolic array. Memory hierarchy design is critical to the overall performance of neural network acceleration due to its memory-intensive nature. As a standard practice, we adopt three levels of memory hierarchy, as shown in Fig. 6.2: 1) DRAM, 2) Global buffer, and 3) double buffered L1 cache [69], [44]. 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.** As discussed in section 2.4, a major benefit in using systolic arrays is parallel computing in a simultaneous row-wise and column-wise manner with local communications [62]. 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, separate 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 [15, 44, 102]. 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, stationary dataflow may result in different impact for spiking models, considering 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.

### 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 temporal parallelism, both across space and time, as illustrated in Fig. 6.2(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 Fig. 6.2(b).

### 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.

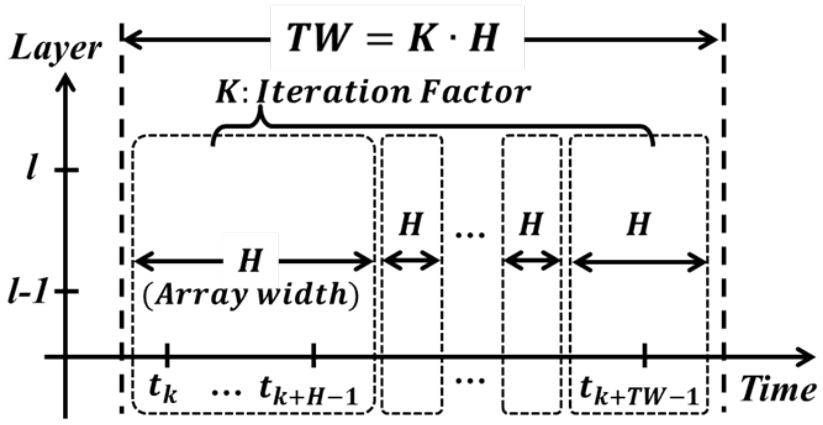


Figure 6.3: Operate the array accelerator with a chosen time window size $TW$ for $K$ array processing iterations.

### 6.2.3 Time-window size optimization (TWSO)

Processing across $TW$ time points within the chosen time window as in (6.1) 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 [103]. Blindly decoupling can exacerbate the above trade-off, and even result in worse performance, as shown later in Fig. 6.5 and Fig. 6.7.

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 [103].

**TWSO address the fundamental trade-off:** 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 Fig. 6.3. 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 Fig. 6.3 and equation 6.1, 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 trade-off by exploring granularity with varying time-window size, and finding optimal point for the trade-off. TWSO is generally applicable to both non-spiking RNNs and spiking RNNs.

**TWSO offers application flexibility:** Typically, 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 6.3, TWSO can significantly improve the overall performance.

## 6.3 Experiments and Results

We perform a comprehensive evaluation of the proposed SaARSP architecture based on both output-stationary (OS) and weight-stationary (WS) dataflows following the setups described in the following section.

### 6.3.1 Configurations and Setups

We use an analytic architecture simulator introduced in section 4.2 to assess the latency, memory access, and energy dissipation of the proposed SaARSP architecture and

Table 6.1: A high-level overview of the user-specified inputs to the simulator.

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

Table 6.2: 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<br>Input/Output Spike - 1bit (binary) |

compare it with a baseline. The simulator takes 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 6.1. We compare different accelerator architectures using a large number of feedforward and recurrent SNNs that are synthetic or adapted from the neuromorphic computing community.

We specifically consider the systolic array specifications in Table 6.2 for the proposed SaARSP architecture in our experimental studies in Section 6.3.

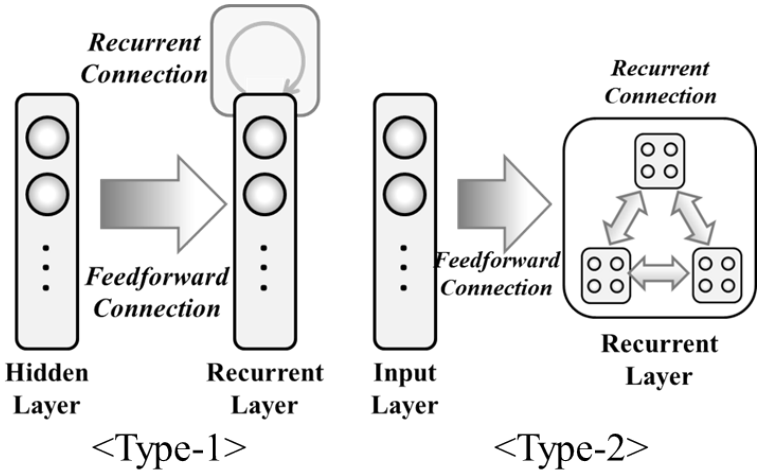


Figure 6.4: Two spiking recurrent layer topologies: (a) Type 1 - uniform, and (b) Type 2 - population based.

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 discussed in Section 6.2.3, 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 6.2.1) and hence processes each layer in a time-sequential manner, e.g. time step by time step. As aforementioned in the previous section, 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.

### 6.3.2 Spiking neural network benchmarks

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 6.3. 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 in Fig. 6.4 that were adopted by the neuromorphic computing community and demonstrated state-of-the-art performances: Type 1) uniform [3], Type 2) population based, comprising multiple distinct neural populations [31].

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 6.3.

We trained a number of R-SNNs with Type 1 recurrent layers using backpropaga-

Table 6.3: 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% |

tion [39] on a set of widely-adopted spiking neural based image/speech reorganization datasets MNIST [90], Fashion MNIST [91] [104], Neuromorphic MNIST (N-MNIST), TI Alpha (English letter subsect of the TI-46 speech corpus) [36], and NTIDIGITS (the neuromorphic version of the speech datasets TIDIGITS) [92]. Note that the examples in the non-neuromorphic datasets above were converted into a spiking form following the standard practice, such as [37]. In Table 6.3, 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 6.3 shows the competitive test accuracy of R-SNNs with Type 1 recurrent layers of two different sizes: 400(H)-400(R) and 1000(H)-1000(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.

Type-2 recurrent layer topology was adopted in the so-called long short-memory spiking neural network (LSNN) model of [31] 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 Fig. 6.4. It was also shown that this R-SNN architecture can support the powerful Learning-to-Learn (L2L) capability as a spiking

Table 6.4: 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) | $C_{H-R} = 0.4$ / $C_{R-R} = 0.7$ |

compute substrate in [31].

### 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 6.4. 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.

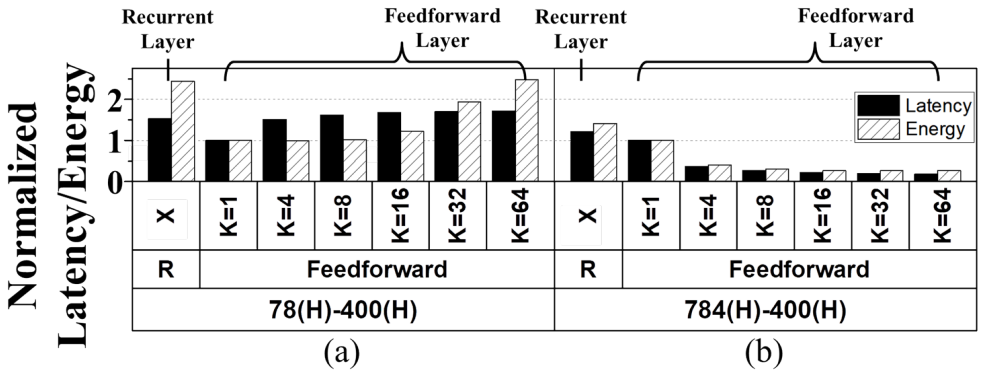


Figure 6.5: Normalized latency/energy of two feedforward layers in comparison with recurrent layers, with $C_{H-R}$=1.0 and $C_{R-R}$=0.3. The values are normalized to those of the feedforward layer with time-iteration factor K=1.

### 6.3.3 Acceleration of feedforward layers with output stationary dataflow

The proposed SaARSP architecture can accelerate feedforward layers as a simpler special case (Section 6.3.2). Under the output stationary dataflow, Fig. 6.5 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 and 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 Fig. 6.5(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

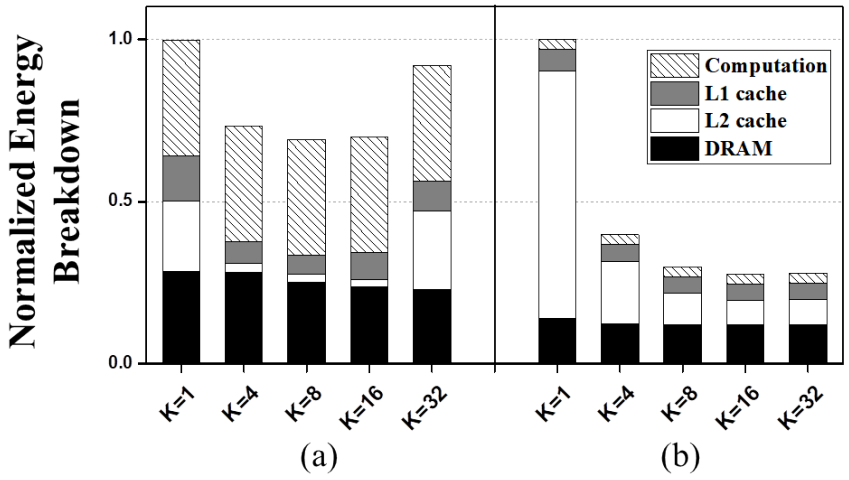


Figure 6.6: Normalized energy dissipation of recurrent layer acceleration under output stationary dataflow.

from higher- to lower-level caches. However, it shall be noted that merely increasing time window size may degrade the overall performance, which is reflected in Fig. 6.5(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.

### 6.3.4 Acceleration of recurrent layers with output stationary dataflow

Before presenting a more complete evaluation of recurrent layer acceleration in Section 6.3.5, we first analyze latency and energy dissipation trade-offs of accelerating Type-1 recurrent layers with varying time window size under the output stationary dataflow.

### Latency

Fig. 6.7(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.

### Energy dissipation

Fig. 6.7(b) shows the normalized energy dissipation of the same set of recurrent layers. Trade-offs 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. Fig. 6.6 shows the breakdown of energy dissipation of two recurrent layers.

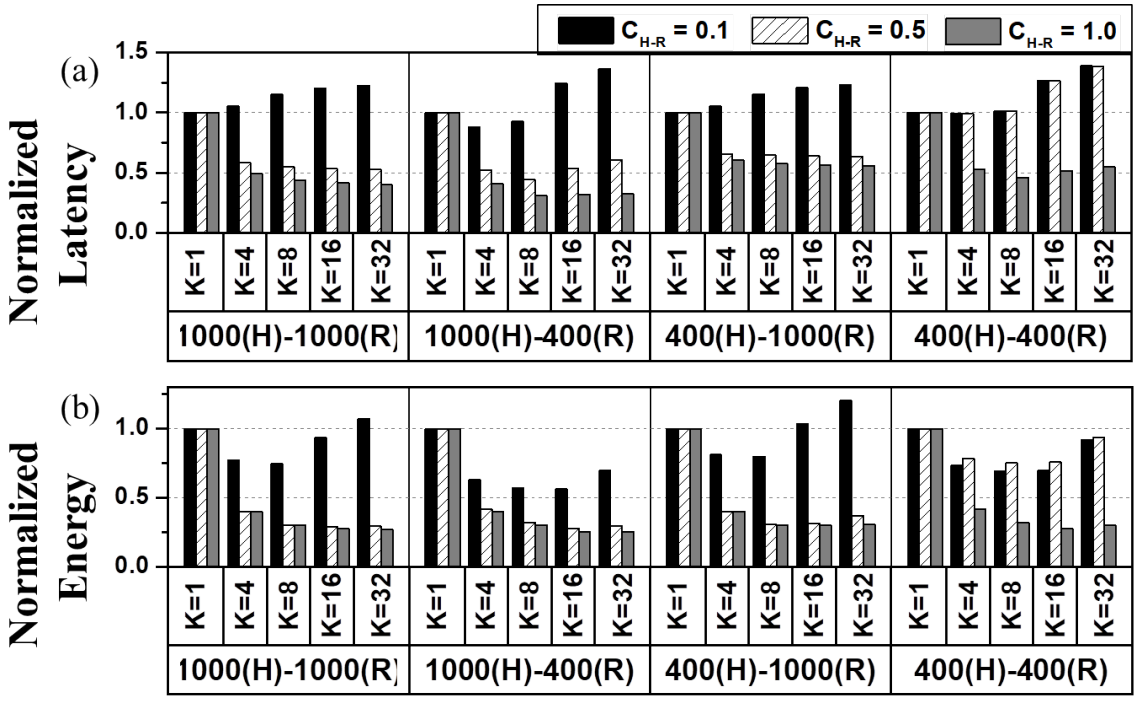


Figure 6.7: Normalized (a) latency, and (b) energy dissipation of recurrent layer acceleration under OS dataflow with $C_{R-R}$=0.5.

### 6.3.5 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 6.3.2. 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 Fig. 6.8 and Fig. 6.9 are normalized to this baseline.

#### Trade-offs 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 Fig. 6.8. Fig. 6.8 reports the latency and energy of the Type-1 and Type-2

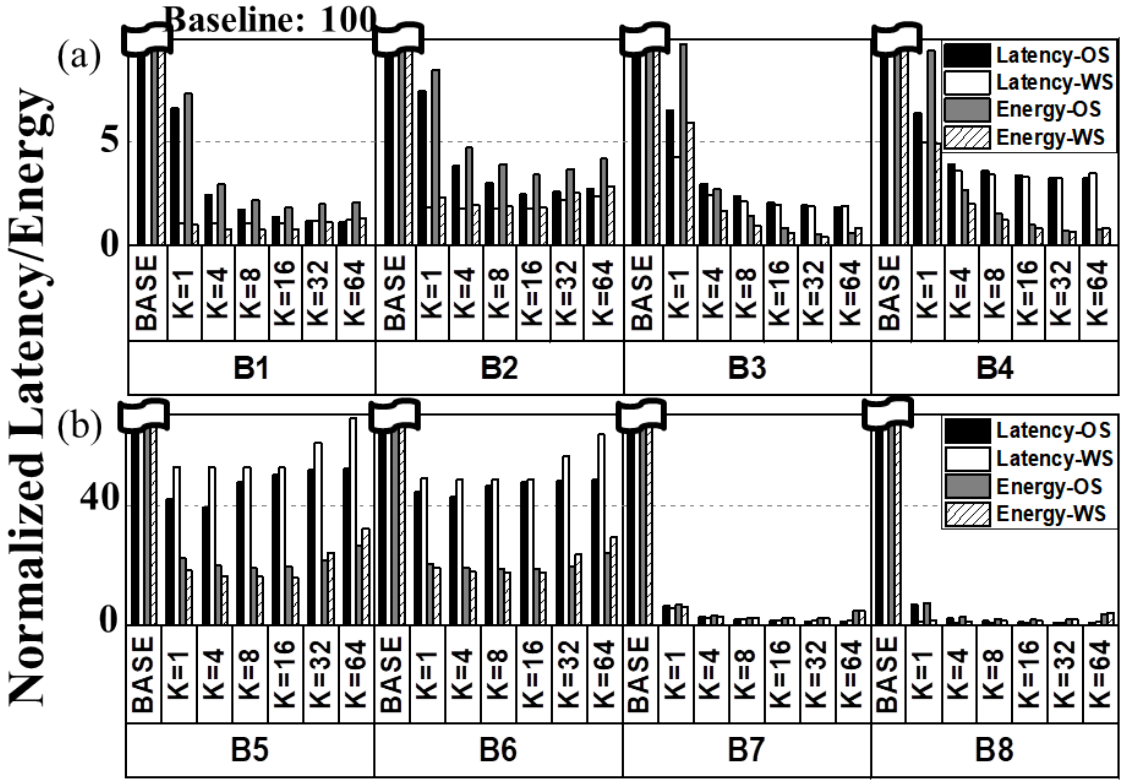


Figure 6.8: Latency/energy of (a) Type-1, and (b) Type-2 recurrent networks with OS and WS dataflows normalized to that of the baseline design, which follows conventional approaches.

networks with various time window sizes ($K$ values) 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 Fig. 6.8. In type-2 networks, the network consists of a relatively smaller pre-synaptic layer size. As shown in Fig. 6.8, 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.

**Impact of stationary dataflows**

Fig. 6.8 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 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 6.5. 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 [105], 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

Table 6.5: 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 % (F/R)*** | 93.7 / 6.3 | 84.3 / 15.7 | 90.0 / 10.0 | 74.9 / 25.1 | 58.2 / 41.8 | 41.5 / 58.5 | 53.2 / 46.8 | 34.3 / 65.7 | **66.3 / 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

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 6.5. 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.

### 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 Fig. 6.9. 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

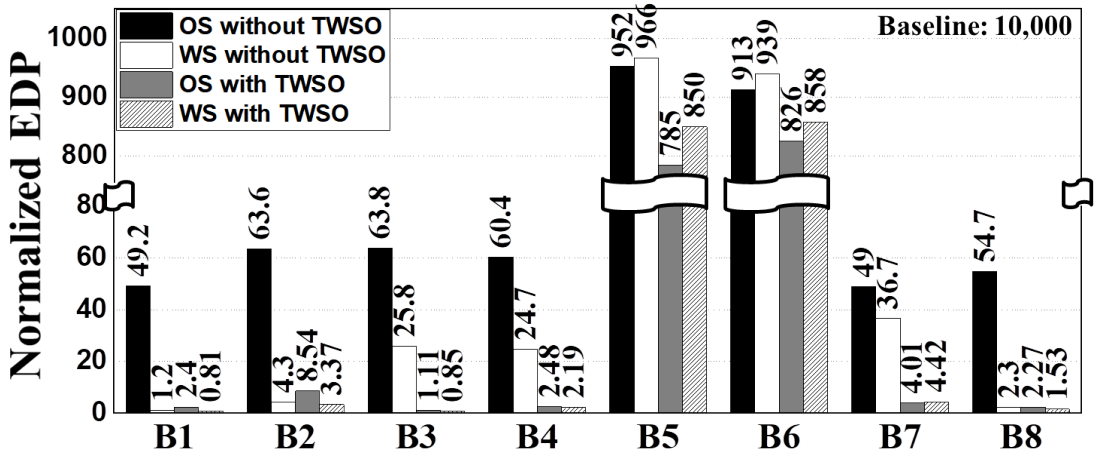


Figure 6.9: Normalized EDP in recurrent layer of eight benchmarks with OS and WS. EDP values are normalized to the baseline result using 1-D array. The EDP of OS and WS with and without the time window optimization is shown.

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.

## 6.4 Summary and Discussions

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.

The proposed SaARSP architecture is built upon a decoupling scheme and novel

time window size optimization (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.

# Chapter 7

# Application-Independent Split-Time-Temporal Coding

In this chapter, we propose a novel technique and architecture that allow the exploitation of temporal information compression with structured sparsity and parallelism across time, and significantly improves data movement on a systolic array. We split a full range of temporal domain into several time windows (TWs) where a TW packs multiple time points, and encode the temporal information in each TW with Split-Time Temporal coding (STT) by limiting the number of spikes within a TW up to one. STT enables sparsification and structurization of irregular firing activities and dramatically reduces computational overhead while delivering competitive classification accuracy without a huge drop. To further improve the data reuse, we propose an Integration Through Time (ITT) technique that processes integration steps across different TWs in parallel with a systolic array. The proposed architecture with STT and ITT offers an application-independent solution for spike-based models across various types of layers and networks.

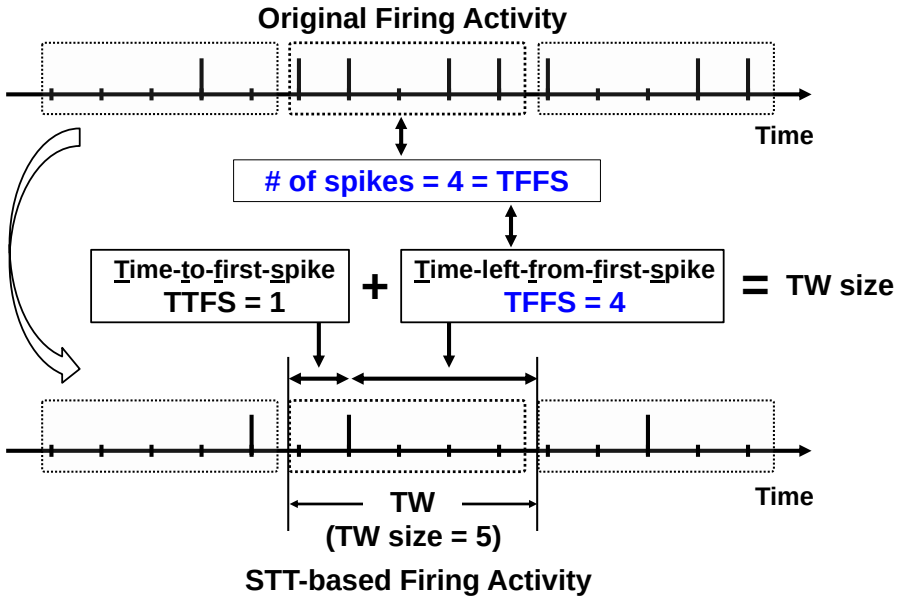


Figure 7.1: Local structurization and sparsification with the proposed STT. Time-left--from-first-spike (TFFS) presents the firing rate of the corresponding TW.

## 7.1 Split-Time Temporal coding (STT)

STT significantly reduces computational overhead by introducing local temporal resolution reduction per TW, well tunable based on TW size, while maintaining global temporal information of the original spikes. While data representation in a single TW bears similarity with conventional temporal coding, the essence of STT is to split and structurize the spiking activities by imposing regularity in terms of the number of spikes per synchronized TW and to enable a tunable tradeoff between machine learning and accelerator performance from a broader perspective. By creating regularized spike trains throughout the network, STT offers multiple benefits including uniform processing time across TWs, reduced computational overhead, and avoiding processing of redundant spikes as in conventional approach.

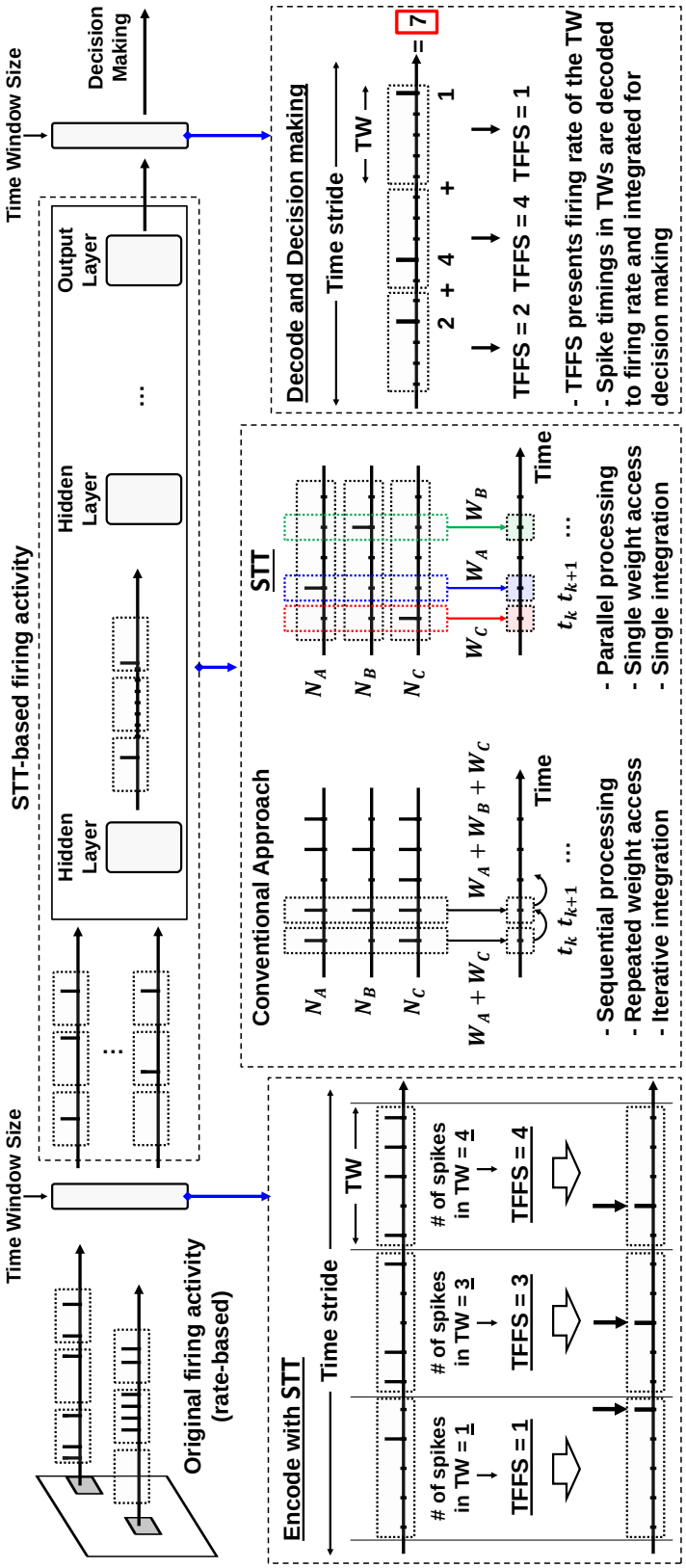


Figure 7.2: Schematic representations of STT-based network operations. (a): STT-encoder at the input layer (b): Comparison between the operations in conventional approaches and the proposed STT-based approach (c): STT-decoder at the output layer

### 7.1.1 Proposed STT

To address the key challenges, we propose a novel technique to locally employ temporal coding for the rate-coded SNNs by dividing the time stride (TS) with a temporal granularity defined by the time window (TW) size, dubbed *Split-Time Temporal coding* (STT). The key idea here is to employ local structurization and sparsification and improve the computational/data movement overhead by reducing the redundancy in rate-coded firing activities on a TW basis, while local rate information is retained by using prefix sum. Importantly, STT is universally applicable for accelerating spiking models, with flexibility in choosing the TW size. The spike timing of a single spike in each TW carries firing rate information with time-left-from-first-spike (TFFS), as shown in Fig. 7.1. All layers in the network operate on locally-temporal codes, based on the proposed STT, with the following rules:

**Rule 1.** We limit the maximum firing count of each neuron in a TW up to one. In all TWs, each neuron is allowed to fire up to once where the only spike represents rate information.

**Rule 2.** The spike information within a TW is represented by the timing of a single spike. Especially, at the input layer, the spike information of original firing activity is converted with STT, based on the number of spikes in a TW.

**Rule 3.** At the output layer, STT-based firing activities are decoded to firing rate. The firing rate of each neuron is decided by integrating the firing rate from all TWs, i.e., the sum of all TFFS in the time domain.

As shown in Fig. 7.2(a), we first convert the rate-coded original firing activities into STT-based firing activities at the input layer. For example, as in Fig. 7.1, if the TW size is 5 and the number of spikes in a TW is 4, the time-left-from-first-spike (TFFS) in a corresponding TW is determined by : TFFS = (TW size) - TTFS = 5 - 1 = 4,

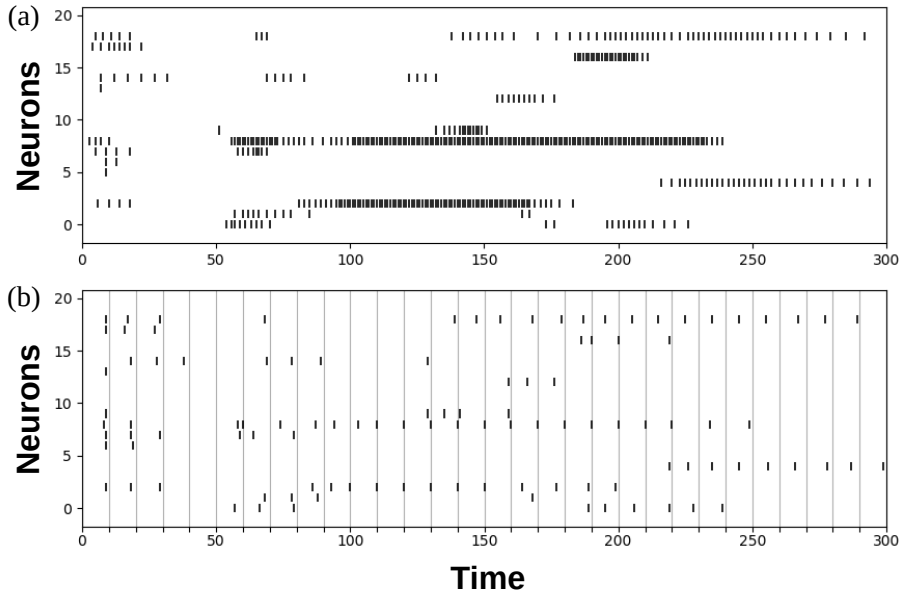


Figure 7.3: Spike raster plot of 20 neurons from the recurrent layer for accelerating NTIDIGITS. (a): Original firing activities without using STT and (b): STT-based firing activities with TW size = 10.

representing the firing rate of the TW. If a neuron does not fire in a specific TW in original firing activities, the STT-based firing activity of the corresponding TW of the neuron also remains silent. In all layers in the network, each neuron follows **Rule 1** and fires at most once for a TW. For each TW throughout the layers, the timing of a single spike represent the spike information of a TW similar to [106]. The earlier the spike, the stronger the stimulus. At the output layer, STT-based firing activities are decoded to firing rate for the decision making. For example, the spike train in Fig. 7.2(c) is decoded by integrating rates across TWs: $\sum$(TFFS) = $\sum$ (TW size)-(TFFS) = 2 + 4 + 1 = 7, following **Rule 3**.

### 7.1.2 STT-based Acceleration

STT-based hardware acceleration significantly simplifies the synaptic input integration step, the dominant computational complexity in spiking neural computations, with

structured, high sparsity as shown in Fig. 7.3. First, STT reduces the repeated weight access across time points to a single weight access per input neuron for a given TW. Since an input neuron fires up to one in a TW, the corresponding weight is used only once for the synaptic input integration. To retain the local and also global information, we use the prefix sum of the STT-based integrated synaptic inputs in a TW while the process is still based on a single spike per TW, following **Rule 2**. As will be shown in Fig. 7.5 and discussed in Section 7.2, the prefix sum of STT-based integrated inputs is equivalent to the Psums using a left-aligned rate code where the firing rate corresponds to TFFS.

Also, STT allows parallel acceleration through time by allocating small memory for each time point of a TW. For example in conventional approaches, the input integration step requires $W_A$ and $W_C$ at time point $t_k$, and $W_A$, $W_B$ and $W_C$ at the next time point $t_{k+1}$, as shown in Fig. 7.2(b). Considering the unstructured firing patterns across different neurons and different time points in actual spiking activities, it requires repeated weight access without data reuse. However, with STT, $W_A$ is integrated to the partial sums (Psums) at $t_{k+1}$, $W_B$ is integrated to Psums at $t_{k+3}$ and $W_C$ is integrated to Psums at $t_k$. Additionally, regularized spike trains, i.e., single spike per TW, enable uniform processing time of TWs. Each weight is used only once in a TW, and multiple TWs are mapped in parallel on a systolic array to maximize the weight reuse across TWs.

### 7.1.3 Machine Learning Performance with STT

Typically, temporally-coded spiking models limit each neuron to fire at most once in the entire time domain. This highly restrictive type of representing spike information can introduce huge benefits in latency and energy efficiency. However, such extreme temporal sparsity does not apply to broader classes of SNNs employing rate or other types of temporal codes or a combination of thereof, and limits achievable accuracy

especially for challenging learning tasks.

On the other hand, rate-coded spiking models can support various types of spatiotemporal dynamics of SNNs. While many recent works based on rate-coded models reported competitive performances on various spatiotemporal tasks with bio-inspired [32, 1] and backpropagation based [106, 39, 4] training methods, iterative weight access due to repeated operations across time and irregular firing patterns complicate hardware acceleration of the spike-based models.

Importantly, STT is universally applicable to any rate-coded model including fully-connected, convolution and recurrent layers for efficient hardware acceleration of a trained network with flexibility in selecting the temporal granularity, i.e., TW size. STT delivers competitive accuracy without any hyper-parameter tuning and significantly reduces computational overhead for synaptic input integration, the dominant complexity of hardware acceleration. STT is fundamentally different from existing temporal coding schemes [106, 72] while the information-carrying feature in a single TW bears a similarity. Results on various networks are discussed in Section 7.3.

## 7.2 Proposed Architecture

We present a systolic array-based SNN accelerator architecture that supports the proposed STT and exploits parallelism in both space and time. The proposed architecture addresses existing inefficiencies via structured sparse firing patterns and parallel computations across TWs based upon the STT.

### 7.2.1 Overview of the Proposed Architecture

Fig. 7.4 shows the overall architecture of the proposed architecture incorporating an STT-encoder for the input layer, an STT-decoder for the output layer, controllers, caches,

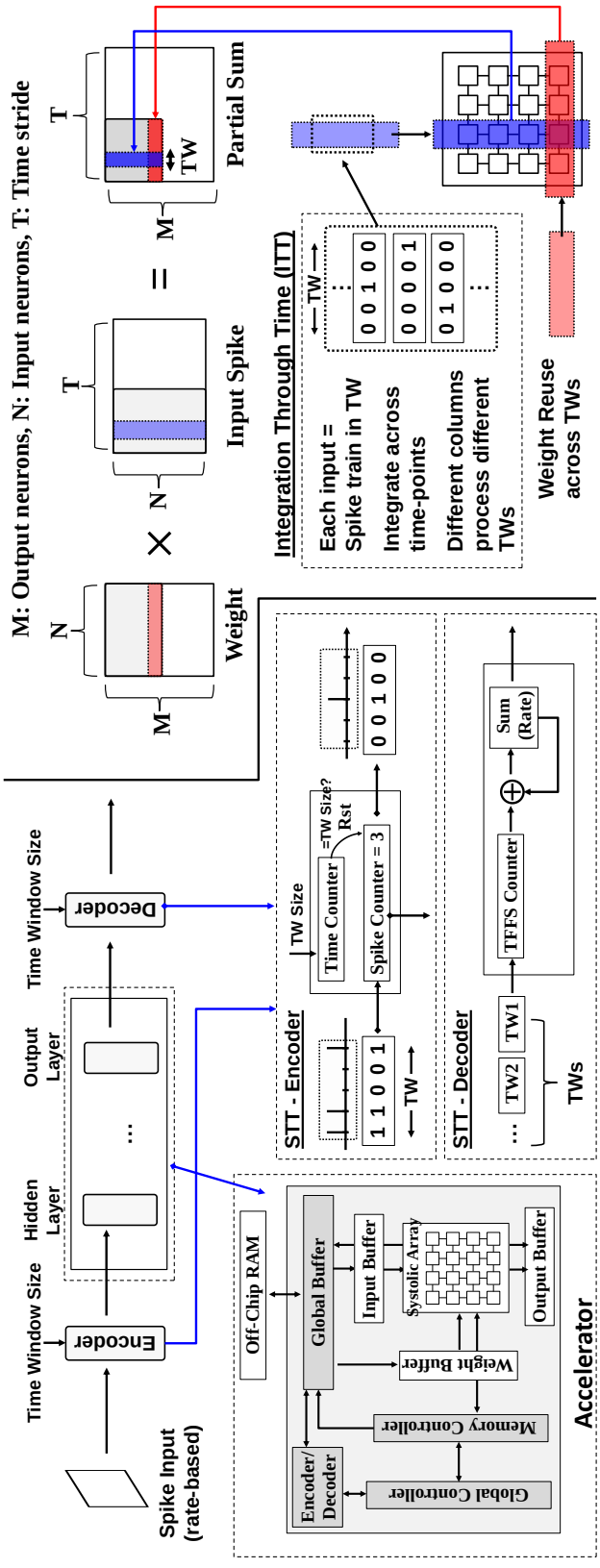


Figure 7.4: (a): Overall architecture of the proposed accelerator (b): STT-encoder and decoder at the input and output layer, respectively (c): Mapping of the inputs and outputs into the systolic array with the proposed ITT

and a systolic array composed of tiled processing elements (PEs) with unidirectional links. As shown in Fig. 7.4(a), the systolic array fetches the required data through three levels of memory hierarchy: 1) off-chip RAM, 2) a global buffer and 3) double-buffered L1 caches. The received spike input and weight data propagates vertically and horizontally with unidirectional links across the 2-D array and is reused through multiple PEs. Each PE is composed of 1) a simple controller, 2) a small scratch-pad memory, 3) accumulate unit (AC), 4) a simple one-hot-to-binary decoder and 5) a comparator. Unlike multiply-and-accumulate (MAC) operations in non-spiking accelerators, simpler AC units are used to accumulate weight values with binary-valued spikes. To fully leverage STT-based acceleration, the synaptic input is properly integrated into a corresponding time point with a simple decoder, and the scratch-pad in each PE stores Psums of all time points in a given TW. In the rest of the paper, we primarily focus on synaptic input integration, the dominant computational complexity.

### 7.2.2 Integration Through-Time (ITT)

As discussed in Section 1.3, conventional approach basically perform the integration step, time-point by time-point, which is a matrix-vector operation as in (2.1). However, time-serial processing hinders data reuse across time-points in many small-medium scale accelerators, resulting in iterative data access. We propose a technique that process integration steps through time which performs integration step as matrix-matrix operation on top of STT, breaking out of a stereotypical approach.

2-D systolic arrays naturally exploit parallelism and data reuse in both vertical and horizontal directions. To fully utilize such advantages, we propose an *Integration Through-Time* (ITT) technique on top of STT, which defines a spiking activity in a TW as a basic unit of workload and maps spiking activities in multiple TWs into the systolic

array, concurrently. As shown in Fig. 7.4(c), ITT assigns entire spike trains in a TW to a single PE and accelerates multiple TWs in different PEs simultaneously. ITT allows for accelerating multiple time points in several TWs in parallel based on the fact that the synaptic input integration step (Step 1) only depends on the spike inputs from the previous layer. Integration of synaptic inputs across multiple time points with ITT can be expressed by modifying (2.1), which is similar to the ones in the previous chapter, but with regulated spiking activities upon STT, as:

**Step 1 - ITT:** Synaptic input integration in $TW_n \sim TW_{n+m}$:

$$\begin{aligned}
&\mathbf{p}^{Post}[TW_n, TW_{n+1}, ..., TW_{n+m}] \\
&= \mathbf{p}^{Post}[(t_{k(n-1)+1}, ..., t_{kn}), ..., (t_{k(n+m-1)+1}, ..., t_{k(n+m)})] \\
&= \mathbf{W}_{Post,Pre} \times \mathbf{s}^{Pre}[TW_n, TW_{n+1}, ..., TW_{n+m}] \\
&= \mathbf{W}_{Post,Pre} \times \mathbf{s}^{Pre}[(t_{k(n-1)+1}, ..., t_{kn}), ..., (..., t_{k(n+m)})]
\end{aligned} \tag{7.1}$$

where $k$ is the size of the TW, $\mathbf{p}^{Post}$ and $\mathbf{s}^{Pre}$ are now matrices, and synaptic input integration is processed across $TW$s. $TW_n$ denotes the $n$-th time window which contains $k$ different time points, i.e., $TW_n = (t_{k(n-1)+1}, ..., t_{kn})$. Remaining steps remains the same as in equation 2.2 $\sim$ 2.4 and all the other expressions follows the definition described in chapter 2.2. Importantly, ITT directly maps the TW to a PE which contains multiple time points which is different from chapter 4. Also, regulated spiking activities with STT enables uniform processing time of TWs which further boost the efficiency.

### 7.2.3 Mapping to Systolic Array

We structurize the irregular sparse firing activities with uniformity across TWs based on STT and accelerate input integration steps of multiple TWs in different output neurons in parallel using ITT. With STT and ITT, our mapping strategy enables parallel

processing in both 1) time: across multiple time point and 2) space: across different output neurons, which significantly improves data movement and processing time.

### Mapping Input and Outputs

The proposed architecture accelerates partitioned matrix-matrix multiplication of the weight and spike input matrices on the systolic array and employs parallelism both across different neurons and different TWs. As shown in Fig. 7.4(c), PEs in a specific row performs the computations for a particular output neuron across different TWs. In each column, PEs process spike inputs of a given TW for different output neurons. Data are only fed from the edges of the systolic array providing high data distribution bandwidth. In each PE, the PE receives spike input and weight from its upper and left neighbors and passes spike input and weight to its lower and right neighbors.

### Energy Reduction

The main bottleneck of the SNN accelerators is the data movement/access overhead of multi-bit weight data which is addressed by the proposed techniques. First, STT minimizes the computational overhead required for dense spiking activities with structured sparsification. STT restricts each neuron to fire at most once in a TW and enables the same weight data associated with a presynaptic neuron to be used only once. In general, applying a larger TW size further reduces the computational and data movement overhead with higher sparsity in spiking activities. Data movement/access is further improved with ITT by the improved weight data reuse. PEs in the same row in the array perform computations of a post-synaptic neuron across different TWs, i.e., the same weight data is reused across PEs in the same row with different spike inputs.

**Utilization Efficiency and Latency**

STT and ITT improve severe under-utilization which originates from iterative data access and the irregularity of sparse firing activities at each time point in the time stride. As discussed in Section 7.1, each neuron fires at most once in a TW with STT, and thus the processing of any TW takes the same amount of time. Uniformity in processing time across TWs and higher sparsity with STT significantly improve latency and utilization efficiency. Iterative access of the required data at each time point can cause stalls of the array, which is the source of inefficiency in addition to computation latency, while ITT reduces memory access to higher-level caches by the improved weight data reuse and less data movement.

While a mapping strategy in [21] share a similarity that different PEs can operate for different time-points, we emphasize that our work is based on the sparse actual spiking activity and multiple time-points packed into one TW, which is fundamentally different from [21].

### 7.2.4 STT-based Layer Acceleration

Each PE accelerates the fundamental operations of a spiking neuron. In recurrent layers, PE operates with a simple additional step to incorporate recurrent synaptic inputs. Note that, hardware resources are reused across different steps.

The operations in a single PE follow the three steps (2.1) $\sim$ (2.4) with an AC unit and a small scratch-pad shared through the steps, as shown in Fig. 7.5(a). In Step 1, the synaptic input integration step, PE determines the address based on the spike timing in a given TW and accumulates the associated weight into a corresponding memory. A single spike in a TW can be interpreted as a one-hot encoded address for the integration. The small scratch-pad memory first stores the integrated synaptic inputs (ISI) of multiple

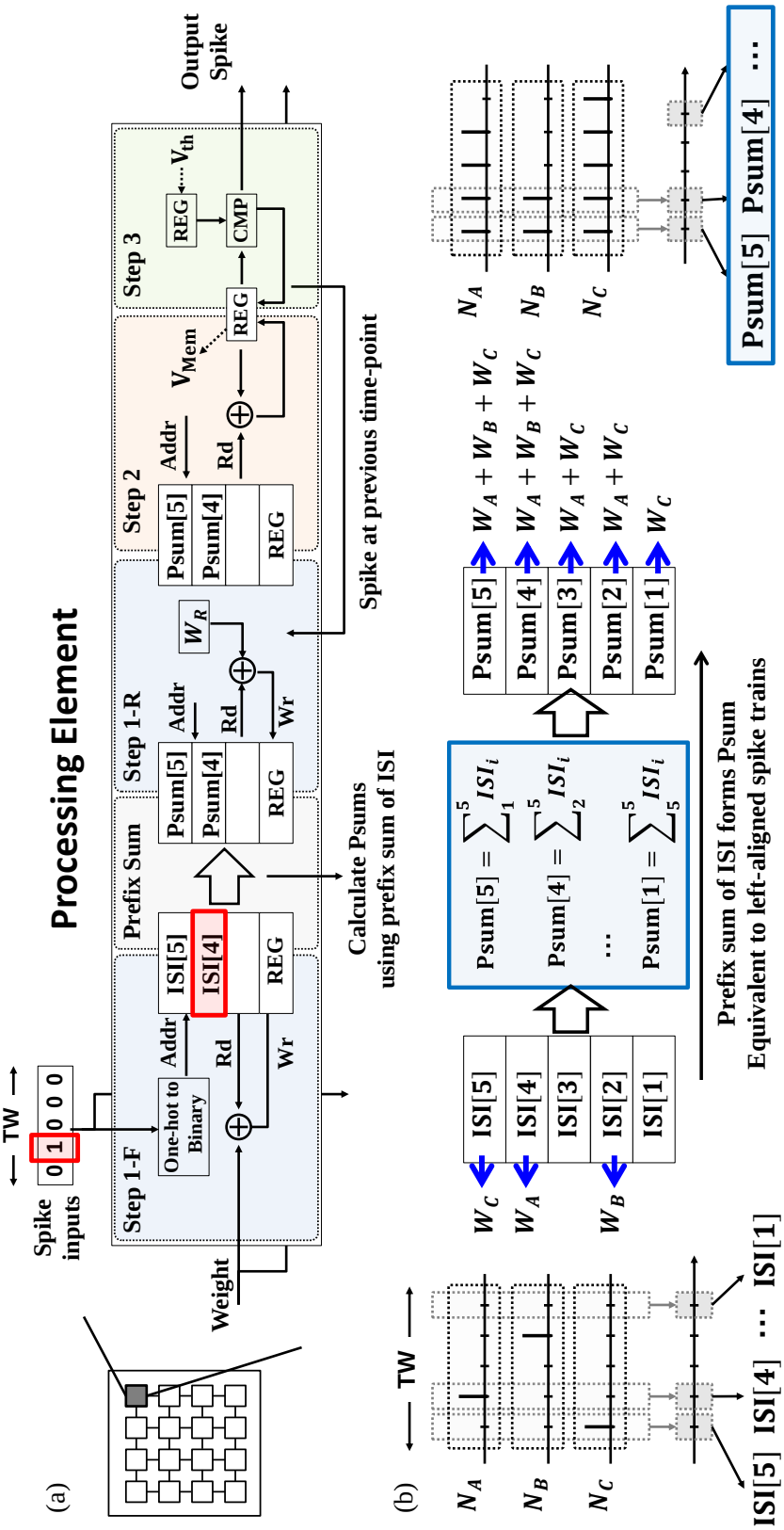


Figure 7.5: Schematic representations of (a): Operations in a PE for accelerating feedforward and recurrent layer (b): Calculating partial sums (Psums) using a prefix sum of the integrated synaptic inputs (ISI)

time points in a given TW. In the above operation, a simple combinational logic, one-hot to binary, converts the spike trains of a TW into an address to the small scratch-pad. As shown in Fig. 7.5(a), for example, if the spike input is 01000 with TW size 5, the associated weight is properly integrated into ISI[TFFS] = ISI[4], which is the integrated synaptic input of the second time point in the TW.

Next, the actual Psum is calculated using the ISI in the previous step. As discussed in Section 7.1 and shown in Fig. 7.5(b), we utilize the prefix sum of ISI which restores the rate and temporal information equivalent to a left-aligned rate code counterpart, while sustaining the advantages of using STT with a single spike. As shown in Fig. 7.5(b), the use of prefix sum yields the same Psum results as using left-aligned, rate codes where the rate equals TFFS. Note that, the number of operations to calculate the prefix sum equals to (TW size - 1) which is negligible compared to input integration steps.

For the rest of the operation, PE processes Step 2 and Step 3 with the integrated Psums, time point by time point, in a sequential manner. At a given time point $t_k$, PE updates the membrane potential with Psum[$t_k$] and the membrane potential of the previous time point $t_{k-1}$. If the updated membrane potential exceeds the pre-defined threshold, the PE generates an output spike and resets the membrane potential.

In case of recurrent layers, the synaptic input integration step is almost the same as in those of feedforward acceleration with additional step for integrating recurrent synaptic inputs, denoted as Step 1-R in Fig. 7.5(a). To simplify the recurrent pass, we adopt self recurrent structure in [107] which only requires a single additional integration operation. The proposed PE is capable of accelerating both feedforward and recurrent layer on top of the proposed techniques. As will discussed in Section 7.3, STT with the use of prefix sum delivers competitive accuracy for various networks and significantly improves the accelerator performance.

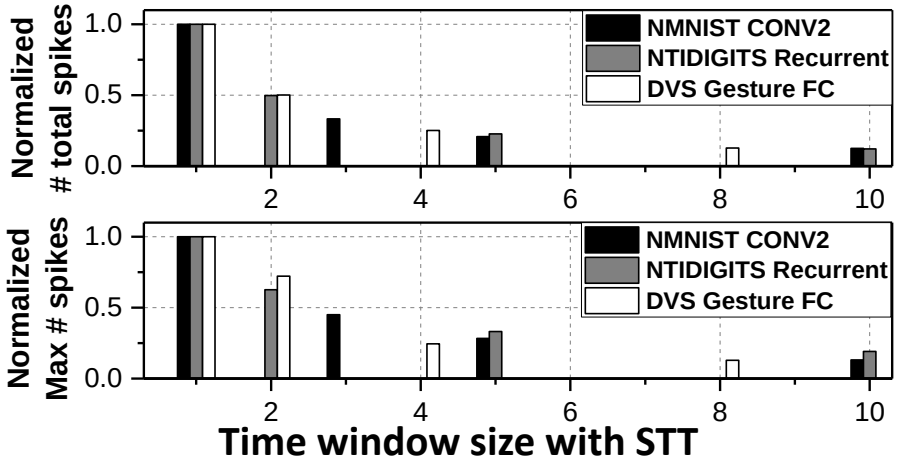


Figure 7.6: Normalized number of total spikes and maximum number of spikes in a neuron with different time window sizes.

## 7.3 Experiments and Results

We perform comprehensive evaluations of the proposed architecture with various layer types, i.e., fully-connected (FC), convolutional (CONV) and recurrent, focusing on the impact of the proposed STT and ITT. We first examine how the data reuse and computational complexity change upon the proposed techniques with critical architectural parameter, i.e., time window (TW) size. Then, we explore joint optimization of machine learning performance and SNN hardware accelerator performance with application-independent split-time temporal coding. We adopt the state-of-the-art training algorithm proposed in [44] as our ML performance baseline, and compare it with our accuracy achieved using the proposed STT over various TW sizes. Since this is the first work of temporal information compression (STT) with time-domain parallel processing (ITT), we set our hardware baseline as the one that has trained with [44] and optimizes data reuse and storage efficiency for each time-point (time-serial approach) without incorporating proposed STT and ITT, as in [108, 22, 109].

Table 7.1: A high-level overview of the user-defined inputs.

| Input | Description |
|---|---|
| Array configuration | Array width/height, size of the scratch-pad in PE |
| Memory configuration | Size of the memory in three levels: off-chip RAM, Global buffer, L1 cache |
| STT | Use STT-based spiking acitivities or plain counterpart along with time window size |
| ITT | Mapping different TWs across columns of the systolic array with given TW size |
| Time Window ($TW$) Size | Ranging from plain inputs ($TW$=1) to the size of a scratch-pad in PE, i.e., $TW$=50 |
| Layer Type | fully-connected, convolutional and recurrent |
| Network Structure | Number of layers, layer types, and number of the neurons in each layer |

Table 7.2: Architecture specifications.

| Components | Proposed Architecture |
|---|---|
| Number of PEs | 128 |
| ALU in PEs | Adder, Comparator - 8-bit |
| Global Buffer Size | 54KB |
| L1/Scratchpad Size | 2KB / 50 × 8-bit |
| DRAM Bandwidth | 30GB/sec |
| Bit precisions | Weight/Membrane Potential - 8-bit<br>Input/Output Spike - $TWS$ × 1-bit<br>($TWS$: $TW$ size) |

### 7.3.1 Configurations and Setups

We use an analytic architecture simulator introduced in section 4.2 to assess the latency, memory access, and energy dissipation of the proposed techniques and compare it with a baseline. The user-specified inputs and architecture specification are summarized in Table 7.1 and Table 7.2.

### 7.3.2 STT: Temporal Information Compression

STT reconstructs the spike information with higher, but structured sparsity by dividing time stride into multiple TWs, squeezing the entire spike information in each TW to

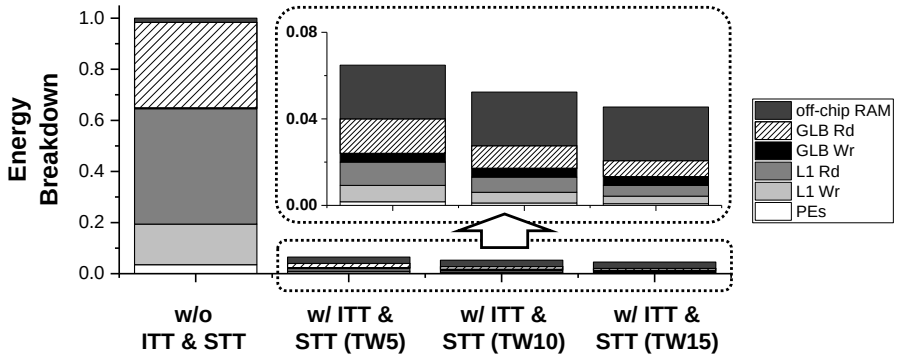


Figure 7.7: Normalized energy dissipation and energy breakdown with and without the proposed techniques.

a timing of the single spike. Spike rates are usually directly related to the performance of the accelerator and a fewer number of spikes not only reduces the number of accumulate operations but also alleviates the overhead of data movement. As introduced in Section 7.1, STT applies to all layer types including FC, CONV and recurrent layers. Furthermore, flexibility in TW size selection for STT enables the proposed architecture to accelerate individual applications with different optimizations.

### Computational overhead

The number of spikes required for layer acceleration decreases with larger TW size, and hence the computational overheads by STT. Given the actual spiking activities, the overhead reduction and a compression of the spike information differ across layers and networks. Approximately, the number of required AC operations is inversely proportional to the TW size, as shown in Fig. 7.6.

### Data movement

STT enables fewer weight data movements associated with active pre-synaptic neurons across the different levels of the memory hierarchy. In conventional approaches, iterative weight access based on the active pre-synaptic neurons at each time point is

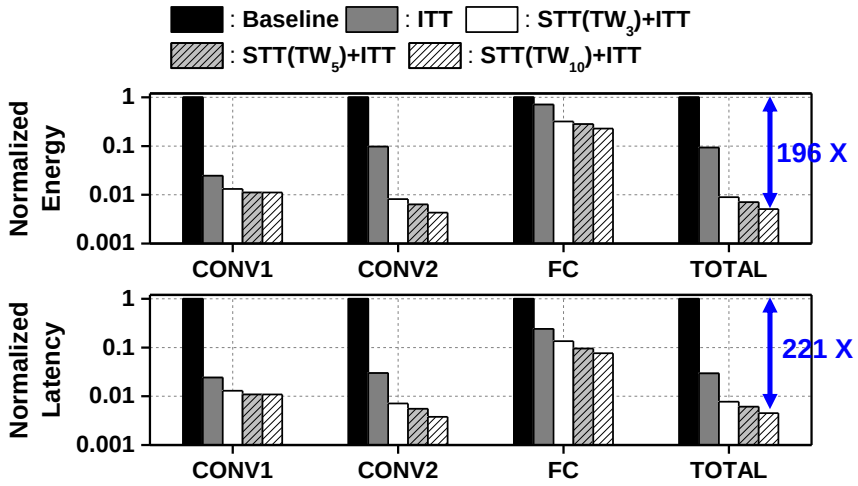


Figure 7.8: Normalized energy dissipation and latency of layers with different TW sizes for NMNIST.

inevitable due to the sequential processing. However, STT reduces temporal resolution, and more sparsely populated spikes mitigate read and write memory access at each level of memory. For example, the spiking activity of a bursting neuron, which fires across five consecutive time points, forces to integrate the corresponding weight in those five-time points, repeatedly. This may incur data movements from higher-level caches depending on spiking activities of the pre-synaptic layer and the memory size. In contrast, the weight is required only once throughout all time points in TW when the information compression is applied. Data movement and reuse are further improved by the proposed ITT.

### 7.3.3 ITT: Data Reuse

ITT significantly improves the data reuse by providing the data sharing opportunity across TWs and post-synaptic neurons, and minimizes the memory access and stall cycles originating from additional latency for iterative memory access. ITT maps spike inputs in multiple TWs into different columns and enables weight reuse across PEs in the same row.

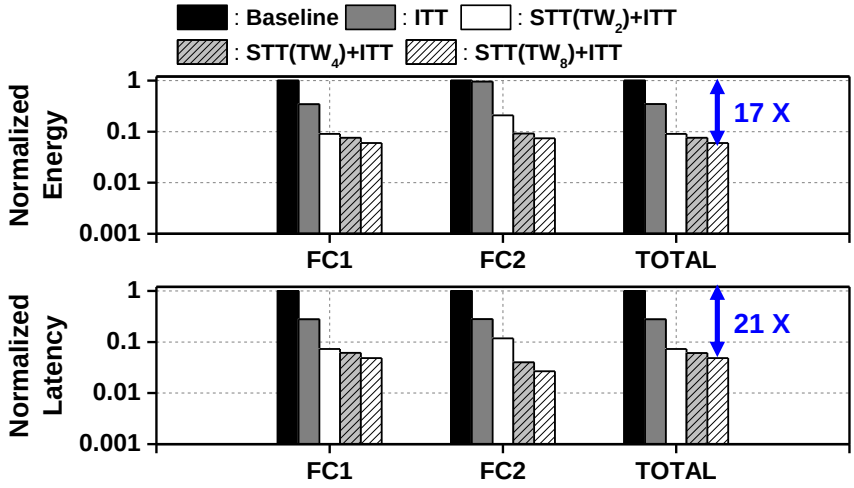


Figure 7.9: Normalized energy dissipation and latency of layers with different TW sizes for DVS-Gesture.

We use the recurrent layer trained for the NTIDIGITs as a representative layer to analyze the impact of the proposed techniques in data movements, as shown in Fig. 7.7. Clearly, larger TW sizes reduce access to higher-level caches and improve energy dissipation. Compared to a conventional approach without the proposed ideas, we observe a huge improvement in memory access to the L1 cache and global buffer. In general, such impact varies from layer to layer based on the actual firing activity, data movements in and between the array and memory hierarchy as a result of the dataflow determined by given memory sizes and layer specifications. Without using the proposed ITT and STT, iterative operations through 300-time points may markedly degrade the overall performance of the accelerator.

### 7.3.4 Comprehensive Evaluations

We examine how the proposed STT and ITT with the key architectural parameter TW size improve the overall accelerator performance. Also, we evaluate the tunable tradeoffs between machine learning and accelerator performance in terms of the TW size.

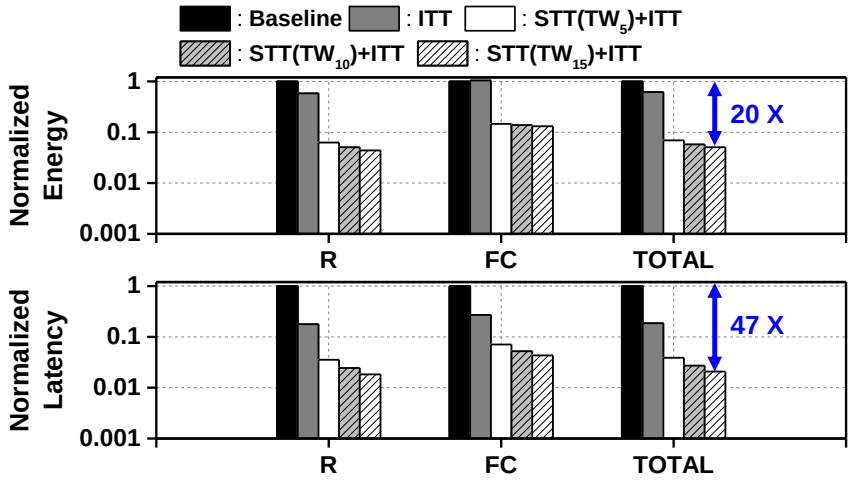


Figure 7.10: Normalized energy dissipation and latency of layers with different TW sizes for NTIDIGITS.

### Latency

We observe a huge improvement in latency by using STT and ITT in all three networks, as shown in Fig. 7.8 $\sim$ 7.10 As discussed in Sections 7.3.2 and 7.3.3, 1) STT reduces latency of the computations in the array proportionally to the TW size by processing TW instead of a time-point, and 2) ITT minimizes the additional delay due to stall cycles that wait for the required data, by reusing the weight data horizontally. In general, a larger TW size compresses the temporal information with a greater stride in the time domain and further reduces computational overheads and data movements, hence the latency.

However, after a certain TW size, the additional improvement with a much larger TW size decreases. This is due to the fact that spikes are often clustered in a certain range in the time domain as shown in Fig. 7.3, and the number of TWs is the reciprocal of TW size. Also, the impact on latency may vary with the spiking activity depending on how uniformly the spikes are spread through neurons and time points. For example, five spikes from five different neurons in the time domain will still have the same computational complexity when STT is applied. In the other case, if there exist five spikes from a single

neuron while other neurons are silent, STT may significantly reduce the computations and weight access. The proposed techniques improved the latency by 97X on average, across three different networks.

### Energy Dissipation

Energy dissipation is reduced as TW size increases in all layers, similar to the latency. Generally, larger TWs provide the opportunity to reuse the same weight across more time points. Especially, the benefit from data movement/reuse is maximized when the layer has relatively a great amount of weight, as in CONV2 in NMNIST. Having more weights emphasizes the benefits compared to the baseline since it exacerbates more data movement and access to higher-level caches in conventional approaches due to iterative weight access.

As discussed, the impact of the proposed techniques on energy dissipation also depends on the temporal sparsity level. For example, a single spike throughout the entire time domain from a particular neuron cannot be reused across TWs and will result in less benefit. Importantly, however, firing activities from a neuron are often clustered in a certain range in the time domain and weights can be reused through the TWs. Across three different networks, our methods delivered 78X energy dissipation improvement, on average.

### Machine Learning Performance

Our experimental results present a huge accelerator performance improvement with temporal information compression using STT. However, there exists a fundamental trade-off between accelerator performance and machine learning performance. While STT significantly improves latency and energy dissipation by using structured and sparse spiking activities, using STT may cause a local temporal information loss in a TW.

**Neuromorphic MNIST**

| Method | Network | Accuracy | Timepoints |
|---|---|---|---|
| HM2BP [4] | 400-400 | 98.88% | 400 |
| SLAYER [43] | 500-500 | 98.95% | 300 |
| SLAYER [43] | CNN[a] | 99.22% | 300 |
| TSSL-BP [39] | CNN[a] | 99.25% | 30 |
| **STT (TWS=3)** | CNN[a] | **99.18%** | $TW_3 \times 10$ |
| **STT (TWS=5)** | CNN[a] | **99.12%** | $TW_5 \times 6$ |
| **STT (TWS=10)** | CNN[a] | **98.76%** | $TW_{10} \times 3$ |
| **STT (TWS=15)** | CNN[a] | **98.10%** | $TW_{15} \times 2$ |

CNN[a]: 12C5-P2-64C5-P2.

**DVS-Gesture**

| Method | Network | Accuracy | Timepoints |
|---|---|---|---|
| RNN [85] | P4-512 | 52.78% | |
| LSTM* [85] | P4-512 | 88.19% | |
| TSSL-BP [39] | P4-512 | 87.15% | 300 |
| **STT (TWS=2)** | P4-512 | **86.46%** | $TW_2 \times 150$ |
| **STT (TWS=4)** | P4-512 | **85.76%** | $TW_4 \times 75$ |
| **STT (TWS=8)** | P4-512 | **84.37%** | $TW_8 \times 38$ |

* includes much greater number of tunable parameters.

Table 7.3: Performance on fully-connected and convolutional networks: NMNIST and DVS-Gesture. TWS denotes the applied time window size.

When the original input spiking activity is converted based on the STT, a single spike in a TW conveys the entire information of the corresponding TW based on the spike timing. We use a prefix sum to avoid iterative weight access and to retain the temporal information, which is equivalent to the ones using left-aligned rate code. While using a single spike can exactly represent the rate information, it may lose some of the temporal information within the TW since the timings of the spikes are not considered.

Nevertheless, STT-based acceleration delivers competitive performance as summarized in Table 7.3 $\sim$ Table 7.4. We adopted the training algorithm in [39] and conducted STT-based inference test on well-trained networks with different TW sizes. For example, [39] achieved 93.29% accuracy and STT-based simulation achieved 92.40% inference accuracy with TW size=5 for the NTIDIGITS dataset. $TW_i$ denotes that the time window

| N-TIDIGITS | | | |
|---|---|---|---|
| **Method** | **Network** | **Accuracy** | **Timepoints** |
| HM2BP [4] | 250-250 | 89.69% | 300 |
| BP (GRU) [41] | 200-200-100 | 89.92% | |
| BP (LSTM) [41] | 250-250 | 91.25% | |
| TSSL-BP [39] | 400[a] | 93.29% | 300 |
| **STT (TWS=5)** | 400[a] | **92.40%** | $TW_5 \times 60$ |
| **STT (TWS=10)** | 400[a] | **91.19%** | $TW_{10} \times 30$ |
| **STT (TWS=15)** | 400[a] | **89.41%** | $TW_{15} \times 20$ |

400[a]: Recurrent layer with LISR [107]

Table 7.4: Performance on recurrent networks: N-TIDIGITS. TWS denotes the applied time window size.

size is $i$, and $TW_5 \times 60$ represents that the original spiking activity, spanned through 300-time points, is encoded to 60 consecutive TWs where each TW contains 5-time points with at most a single spike. We observe that the proposed STT can deliver competitive inference performance up to a certain TW size across various networks as in Table 7.3 $\sim$ Table 7.4 while providing a significant improvements on hardware acceleration.

### EDP evaluation

We adopt an energy-delay product (EDP) to simultaneously consider latency and energy dissipation for evaluation of the proposed techniques and to compare the impact of the TW size selection. We multiply the total execution time with the total energy dissipated at each layer and add up the EDP values of all layers in the network. Importantly, the proposed techniques can be generally applied to rate-based SNN models for an aggressive inference acceleration with flexibility in choosing TW size, depending on the application objective which is further discussed in the next section. As shown in Fig. 7.11, our work delivers 15,000X EDP improvement with the maximum TW sizes which do not significantly drop the accuracy (¡3%), on average, across four different benchmarks.

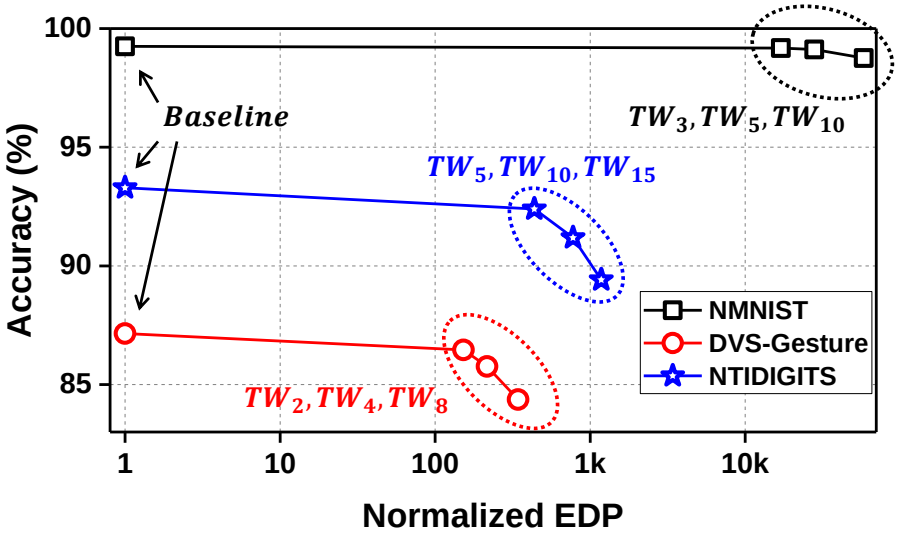


Figure 7.11: Machine learning performance (inference) - Accelerator performance (normalized EDP) tradeoffs on various datasets.

## 7.4 Summary and Discussions

In this work, we propose novel techniques 1) STT: structurization and sparsification of irregular firing activities on a TW basis and 2) ITT: parallel acceleration of TWs and data reuse across space and time, with the systolic array-based accelerator supporting the two techniques. As shown in Fig. 7.11, applying STT and ITT significantly improves the accelerator performance without a huge accuracy drop across various types of networks, i.e., FC, CONV and recurrent. Our work introduces a universally applicable solution to optimize the tradeoffs between the machine learning (ML) performance and hardware accelerator (HW) performance with flexibility in choosing TW size, depending on the objectives.

### ML-HW Performance Trade-off

STT significantly reduces computational overhead by introducing local temporal resolution reduction per TW, well tunable based on TW size, without hyper parameter

tuning, while maintaining global temporal information of the original spikes. Aggressive reduction with larger TW sizes ensure more substantial accelerator performance improvement, but may lose local temporal information in TW, leading to a non-negligible classification accuracy drop.

We adopt an energy-delay product (EDP) to simultaneously consider latency and energy dissipation for evaluation of the proposed techniques and to compare the impact of the TW size selection. We multiply the total execution time with the total energy dissipated at each layer and add up the EDP values of all layers in the network. As shown in Fig. 7.11, ML-HW performance trade off can be flexibly adjusted depending on application objectives where small TW sizes with STT and ITT can still deliver significant improvement. Our work delivers 15,000X EDP improvement with the maximum TW sizes which do not significantly drop the accuracy, on average across different benchmarks, as shown in Fig. 7.11.

Technically, all spiking models incorporate the synaptic input integration step, the Step 1 (2.1) mentioned in Section 2.2. Therefore, the proposed techniques are generally applicable across all typical spiking operations, all layer structures including fully-connected, convolutional and recurrent layers, and general SNNs with various layer types. As a tunable trade-off, we leave the selection of TW size for STT and ITT as an open-ended solution for other works to decide based on the application goal.

# Chapter 8

# Conclusion

## 8.1 Conclusion

This dissertation mainly focuses on hardware-friendly training algorithms, systematic dataflow exploration of various spiking neural networks (SNNs) for efficient and parallel acceleration of neural computations, and temporal resolution reduction technique with STT, which allows to process regulated spiking activities. We briefly summarize the major contributions of this dissertation as follows.

In Chapter 3, we propose a novel spike-level direct feedback alignment (ST-DFA) algorithm for training multi-layer spiking neural networks (SNNs) with improved bio-plausibility and scalability over traditional backpropagation algorithms. Moreover, it is demonstrated that the ST-DFA algorithm with its hardware-friendly optimized implementation enable efficient on-chip training of FPGA SNN neural processors while delivering competitive classification performance for practical speech and image recognition tasks. Compared to the hardware implementation of the state-of-the-art BP algorithm HM2-BP, the design of the proposed ST-DFA reduces functional resources by 76.7% and backward training latency by 31.6% while gracefully trading off classification

performance.

Acceleration of SNNs is challenged by fundamental issues: unstructured sparsity emergent in both space and time, and stereotypical approach to process spiking models in a time-sequential manner which requires iterative data access.

In Chapter 4, a novel scheme is introduced to enable the parallel acceleration of computation across multiple time points, which further leads to large performance and efficiency gains via optimization of the tiling strategy. We demonstrate how the tiling strategy can be reconfigured on a layer-by-layer basis and jointly optimized with the accelerator hardware to achieve large gains in throughput and energy efficiency. Furthermore, an SNN dataflow simulator has been developed to aid systemic design space exploration. The proposed techniques are able to improve EDP of the accelerator by several orders of magnitude and more than one order of magnitude over a non-optimized and existing SNN dataflow, respectively.

In Chapter 5, we presents the first analysis framework to evaluate temporal parallel processing of systolic array-based hardware architecture for accelerating SNNs, which efficiently manages the sparse nature of spiking computations and supports a diverse family of spiking models. The proposed architecture is built upon a novel parallel time batching (PTB) technique and a spatiotemporally-non-overlapping spiking activity packing (StSAP) strategy. PTB introduces parallel acceleration of time windows (TWs) that incorporates multiple time-points, and significantly improves energy efficiency and under-utilization by reducing iterative data access and idling of processing units. StSAP densifies the grouped input spikes (TBs) by combining non-bursting neurons with greedy policy, which further benefits the utilization efficiency of the array. We also observe that larger TW size does not always provide monotonic improvements, and hence perform a joint optimization of PTB and StSAP with varying TW sizes for different 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. In Chapter 6, we extend the parallel time computation technique to recurrent SNNs, which basically decouples the processing of feedforward synaptic connections from that of recurrent connections. The proposed SaARSP architecture is built upon a decoupling scheme and novel time window size optimization (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.

In Chapter 7, we propose a novel, universally applicable solution for sparsification and structurization of any rate-based spiking activities and explore the impact of temporal granularity defined by the time window (TW) size. STT significantly improves accelerator performance by reducing the spike redundancy on a TW basis and handling the TW as the basic unit of operation with structured firing activities across TWs. Also, we introduce the ITT upon STT technique, which enables parallel acceleration in time based on simultaneous processing of multiple TWs across columns of the systolic array. ITT enables the data reuse across TWs with uniform processing times for TWs, leading to further improved performance on top of STT. Lastly, we develop a systolic array-based architecture supporting STT and ITT, which is capable of accelerating various types of layers.

We hope this dissertation could help the neuromorphic community to attain energy-efficient, high performance, and move forward.

# Bibliography


[1] Y. Hao, X. Huang, M. Dong, and B. Xu, *A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule*, *Neural Networks* **121** (2020) 387–395.

[2] Y. Zhang, P. Li, Y. Jin, and Y. Choe, *A digital liquid state machine with biologically inspired learning and its application to speech recognition*, *IEEE transactions on neural networks and learning systems* **26** (2015), no. 11 2635–2649.

[3] W. Zhang and P. Li, *Spike-train level backpropagation for training deep recurrent spiking neural networks*, in *Advances in Neural Information Processing Systems*, pp. 7800–7811, 2019.

[4] Y. Jin, W. Zhang, and P. Li, *Hybrid macro/micro level backpropagation for training deep spiking neural networks*, in *Advances in Neural Information Processing Systems*, pp. 7005–7015, 2018.

[5] W. Maass, *Networks of spiking neurons: the third generation of neural network models*, *Neural networks* **10** (1997), no. 9 1659–1671.

[6] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et. al.*, *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 Systems* **34** (2015), no. 10 1537–1557.

[7] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et. al.*, *Loihi: A neuromorphic manycore processor with on-chip learning*, *IEEE Micro* **38** (2018), no. 1 82–99.

[8] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et. al.*, *Learning representations by back-propagating errors*, *Cognitive modeling* **5** (1988), no. 3 1.

[9] S. M. Bohte, J. N. Kok, and H. La Poutre, *Error-backpropagation in temporally encoded networks of spiking neurons*, *Neurocomputing* **48** (2002), no. 1-4 17–37.

[10] J. H. Lee, T. Delbruck, and M. Pfeiffer, *Training deep spiking neural networks using backpropagation*, *Frontiers in neuroscience* **10** (2016) 508.

[11] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, *Spatio-temporal backpropagation for training high-performance spiking neural networks*, *Frontiers in neuroscience* **12** (2018).

[12] P. J. Werbos *et. al.*, *Backpropagation through time: what it does and how to do it*, *Proceedings of the IEEE* **78** (1990), no. 10 1550–1560.

[13] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et. al.*, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998), no. 11 2278–2324.

[14] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, *Converting static image datasets to spiking neuromorphic datasets using saccades*, *Frontiers in neuroscience* **9** (2015) 437.

[15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, *Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, *IEEE journal of solid-state circuits* **52** (2016), no. 1 127–138.

[16] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, *Automated systolic array architecture synthesis for high throughput cnn inference on fpgas*, in *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6, 2017.

[17] Y. Shen, M. Ferdman, and P. Milder, *Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer*, in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 93–100, IEEE, 2017.

[18] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, *Sparse-tpu: Adapting systolic arrays for sparse matrices*, in *Proceedings of the 34th ACM International Conference on Supercomputing*, pp. 1–12, 2020.

[19] H. Kung, B. McDanel, and S. Q. Zhang, *Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization*, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 821–834, 2019.

[20] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, *Spinalflow: an architecture and dataflow tailored for spiking neural networks*, in

*2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 349–362, IEEE, 2020.

[21] J.-J. Lee and P. Li, *Reconfigurable dataflow optimization for spatiotemporal spiking neural computation on systolic array accelerators*, in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 57–64, IEEE, 2020.

[22] D. Neil and S.-C. Liu, *Minitaur, an event-driven fpga-based spiking network accelerator*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **22** (2014), no. 12 2621–2628.

[23] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, *Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation*, *IEEE Journal of Solid-State Circuits* **48** (2013), no. 8 1943–1953.

[24] C. Mayr, S. Hoeppner, and S. Furber, *Spinnaker 2: A 10 million core processor system for brain simulation and machine learning*, *arXiv preprint arXiv:1911.02385* (2019).

[25] I. M. Comsa, T. Fischbacher, K. Potempa, A. Gesmundo, L. Versari, and J. Alakuijala, *Temporal coding in spiking neural networks with alpha synaptic function*, *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May, 2020).

[26] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, *Stdp-based spiking deep convolutional neural networks for object recognition*, *Neural Networks* **99** (Mar, 2018) 56–67.

[27] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, *Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning*, *ACM SIGARCH Computer Architecture News* **42** (2014), no. 1 269–284.

[28] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, *High-performance fpga-based cnn accelerator with block-floating-point arithmetic*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **27** (2019), no. 8 1874–1885.

[29] 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*, *arXiv preprint arXiv:1602.02830* (2016).

[30] W. Nogami, T. Ikegami, R. Takano, T. Kudoh, *et. al.*, *Optimizing weight value quantization for cnn inference*, in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.

[31] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, *Long short-term memory and learning-to-learn in networks of spiking neurons*, in *Advances in Neural Information Processing Systems*, pp. 787–797, 2018.

[32] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, *Stdp-based spiking deep convolutional neural networks for object recognition*, *Neural Networks* **99** (2018) 56–67.

[33] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[34] A. Krizhevsky, V. Nair, and G. Hinton, *The cifar-10 dataset*, *online: http://www. cs. toronto. edu/kriz/cifar. html* **55** (2014), no. 5.

[35] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, *Cifar10-dvs: an event-stream dataset for object classification*, *Frontiers in neuroscience* **11** (2017) 309.

[36] M. Liberman, R. Amsler, K. Church, E. Fox, C. Hafner, J. Klavans, M. Marcus, B. Mercer, J. Pedersen, P. Roossin, D. Walker, S. Warwick, and A. Zampolli, *TI 46-word LDC93S9*, 1991.

[37] R. Lyon, *A computational model of filtering, detection, and compression in the cochlea*, in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, vol. 7, pp. 1282–1285, IEEE, 1982.

[38] P. Lichtsteiner, C. Posch, and T. Delbruck, *A 128 $\times$128 120 db 15$\mu$s latency asynchronous temporal contrast vision sensor*, *IEEE journal of solid-state circuits* **43** (2008), no. 2 566–576.

[39] W. Zhang and P. Li, *Temporal spike sequence learning via backpropagation for deep spiking neural networks*, *Advances in Neural Information Processing Systems* **33** (2020).

[40] R. G. Leonard and G. Doddington, *Tidigits speech corpus*, *Texas Instruments, Inc* (1993).

[41] J. Anumula, D. Neil, T. Delbruck, and S.-C. Liu, *Feature representations for neuromorphic audio spike streams*, *Frontiers in neuroscience* **12** (2018) 23.

[42] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, *et. al.*, *A low power, fully event-based gesture recognition system*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7243–7252, 2017.

[43] S. B. Shrestha and G. Orchard, *Slayer: Spike layer error reassignment in time*, in *Advances in Neural Information Processing Systems*, pp. 1419–1428, 2018.

[44] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, *Scale-sim: Systolic cnn accelerator simulator*, *arXiv preprint arXiv:1811.02883* (2018).

[45] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et. al.*, *Google's neural machine translation system: Bridging the gap between human and machine translation*, *arXiv preprint arXiv:1609.08144* (2016).

[46] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, *Random synaptic feedback weights support error backpropagation for deep learning*, *Nature Communications* (11, 2016) 13276 EP.

[47] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, *Event-driven random back-propagation: Enabling neuromorphic deep learning machines*, *Frontiers in neuroscience* **11** (2017) 324.

[48] A. Nøkland, *Direct feedback alignment provides learning in deep neural networks*, in *Advances in neural information processing systems*, pp. 1037–1045, 2016.

[49] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[50] D. E. Rumelhart, J. L. McClelland, P. R. Group, *et. al.*, *Parallel distributed processing*, vol. 1. MIT press Cambridge, 1988.

[51] B. Schrauwen and J. Van Campenhout, *Bsa, a fast and accurate spike train encoding scheme*, in *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 4, pp. 2825–2830, IEEE, 2003.

[52] H. Mostafa, *Supervised learning based on temporal coding in spiking neural networks*, *IEEE transactions on neural networks and learning systems* **29** (2018), no. 7 3227–3235.

[53] D. Neil, M. Pfeiffer, and S.-C. Liu, *Phased lstm: Accelerating recurrent network training for long or event-based sequences*, in *Advances in Neural Information Processing Systems*, pp. 3882–3890, 2016.

[54] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, *EMNIST: an extension of mnist to handwritten letters*, *arXiv preprint arXiv:1702.05373* (2017).

[55] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, *Memory-centric accelerator design for convolutional neural networks*, in *2013 IEEE 31st international conference on computer design (ICCD)*, pp. 13–19, IEEE, 2013.

[56] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, *A 240 g-ops/s mobile coprocessor for deep neural networks*, in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 682–687, 2014.

[57] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et. al.*, *Dadiannao: A machine-learning supercomputer*, in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE, 2014.

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

[59] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, *Fully hardware-implemented memristor convolutional neural network*, *Nature* **577** (2020), no. 7792 641–646.

[60] Y. Cao, Y. Chen, and D. Khosla, *Spiking deep convolutional neural networks for energy-efficient object recognition*, *International Journal of Computer Vision* **113** (2015), no. 1 54–66.

[61] P. Gysel, M. Motamedi, and S. Ghiasi, *Hardware-oriented approximation of convolutional neural networks*, *arXiv preprint arXiv:1604.03168* (2016).

[62] S.-Q. Wang, L. Wang, Y. Deng, Z.-J. Yang, S.-S. Guo, Z.-Y. Kang, Y.-F. Guo, and W.-X. Xu, *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.

[63] A. Afifi, A. Ayatollahi, and F. Raissi, *Implementation of biologically plausible spiking neural network models on the memristor crossbar-based cmos/nano circuits*, in *2009 European Conference on Circuit Theory and Design*, pp. 563–566, IEEE, 2009.

[64] L. Chen, C. Li, T. Huang, Y. Chen, and X. Wang, *Memristor crossbar-based unsupervised image learning*, *Neural Computing and Applications* **25** (2014), no. 2 393–400.

[65] Q. Wang, Y. Li, and P. Li, *Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing*, in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 361–364, IEEE, 2016.

[66] Y. Liu, Y. Jin, and P. Li, *Exploring sparsity of firing activities and clock gating for energy-efficient recurrent spiking neural processors*, in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2017.

[67] Y. Liu, Y. Jin, and P. Li, *Online adaptation and energy minimization for hardware recurrent spiking neural networks*, *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **14** (2018), no. 1 1–21.

[68] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, *et. al.*, *Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps*, *IEEE transactions on neural networks and learning systems* **30** (2018), no. 3 644–656.

[69] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, *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*, pp. 754–768, 2019.

[70] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, *Cacti 6.0: A tool to model large caches*, *HP laboratories* **1** (2009) 1–24.

[71] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, *Going deeper in spiking neural networks: Vgg and residual architectures*, *Frontiers in neuroscience* **13** (2019) 95.

[72] L. Zhang, S. Zhou, T. Zhi, Z. Du, and Y. Chen, *Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1319–1326, 2019.

[73] P.-Y. Tan, P.-Y. Chuang, Y.-T. Lin, C.-W. Wu, and J.-M. Lu, *A power-efficient binary-weight spiking neural network architecture for real-time object classification*, *arXiv preprint arXiv:2003.06310* (2020).

[74] H.-T. Kung, B. McDanel, and S. Q. Zhang, *Mapping systolic arrays onto 3d circuit structures: Accelerating convolutional neural network inference*, in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 330–336, IEEE, 2018.

[75] H.-T. Kung, *Why systolic architectures?*, *Computer* **15** (1982), no. 01 37–46.

[76] H. Kung and C. E. Leiserson, *Systolic arrays (for vlsi)*, in *Sparse Matrix Proceedings 1978*, vol. 1, pp. 256–282, Society for industrial and applied mathematics Philadelphia, PA, USA, 1979.

[77] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, *Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices*, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **9** (2019), no. 2 292–308.

[78] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, *Eie: Efficient inference engine on compressed deep neural network*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 243–254.

[79] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, *Sparten: A sparse tensor accelerator for convolutional neural networks*, in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 151–165, 2019.

[80] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, *Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks*, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 749–763, 2019.

[81] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, *Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations*, in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 600–614, 2019.

[82] Z. Du, D. D. B.-D. Rubin, Y. Chen, L. Hel, T. Chen, L. Zhang, C. Wu, and O. Temam, *Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches*, in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 494–507, IEEE, 2015.

[83] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, *Regularization of neural networks using dropconnect*, in *International conference on machine learning*, pp. 1058–1066, PMLR, 2013.

[84] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, *Direct training for spiking neural networks: Faster, larger, better*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1311–1318, 2019.

[85] W. He, Y. Wu, L. Deng, G. Li, H. Wang, Y. Tian, W. Ding, W. Wang, and Y. Xie, *Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences*, *Neural Networks* **132** (2020) 108–120.

[86] S. Deng and S. Gu, *Optimal conversion of conventional artificial neural networks to spiking neural networks*, *arXiv preprint arXiv:2103.00476* (2021).

[87] W. Zhang and P. Li, *Skip-connected self-recurrent spiking neural networks with joint intrinsic parameter and synaptic weight training*, *Neural Computation* **33** (2021), no. 7 1886–1913.

[88] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, *Hfirst: A temporal approach to object recognition*, *IEEE transactions on pattern analysis and machine intelligence* **37** (2015), no. 10 2028–2040.

[89] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, *Hats: Histograms of averaged time surfaces for robust event-based object classification*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1731–1740, 2018.

[90] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998), no. 11 2278–2324.

[91] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[92] J. Anumula, D. Neil, T. Delbruck, and S.-C. Liu, *Feature representations for neuromorphic audio spike streams*, *Frontiers in neuroscience* **12** (2018) 23.

[93] J. S. Garofolo, *Timit acoustic phonetic continuous speech corpus*, *Linguistic Data Consortium, 1993* (1993).

[94] S. Saha, H. Duwe, and J. Zambreno, *Cynapse: A low-power reconfigurable neural inference accelerator for spiking neural networks*, *Journal of Signal Processing Systems* **92** (2020), no. 9 907–929.

[95] W. Guo, H. E. Yantir, M. E. Fouda, A. M. Eltawil, and K. N. Salama, *Toward the optimal design and fpga implementation of spiking neural networks*, *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[96] S.-G. Cho, E. Beigné, and Z. Zhang, *A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm cmos*, in *2019 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, IEEE, 2019.

[97] K. Akbarzadeh-Sherbaf, S. Safari, and A.-H. Vahabie, *A digital hardware implementation of spiking neural networks with binary force training*, *Neurocomputing* **412** (2020) 129–142.

[98] S. Guo, L. Wang, S. Wang, Y. Deng, Z. Yang, S. Li, Z. Xie, and Q. Dou, *A systolic snn inference accelerator and its co-optimized software framework*, in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 63–68, 2019.

[99] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, *Spiking neural networks hardware implementations and challenges: A survey*, *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **15** (2019), no. 2 1–35.

[100] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, *A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)*, *IEEE Transactions on Biomedical Circuits and Systems* **12** (2018), no. 1 106–122.

[101] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, *et. al.*, *Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores*, in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, IEEE, 2013.

[102] Y.-H. Chen, J. Emer, and V. Sze, *Using dataflow to optimize energy efficiency of deep neural network accelerators*, *IEEE Micro* **37** (2017), no. 3 12–21.

[103] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzàlez, *E-pur: an energy-efficient processing unit for recurrent neural networks*, in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pp. 1–12, 2018.

[104] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, *Converting static image datasets to spiking neuromorphic datasets using saccades*, *Frontiers in neuroscience* **9** (2015) 437.

[105] N. K. Jha, S. Ravishankar, S. Mittal, A. Kaushik, D. Mandal, and M. Chandra, *Draco: Co-optimizing hardware utilization, and performance of dnns on systolic accelerator*, in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 574–579, IEEE, 2020.

[106] S. Park, S. Kim, B. Na, and S. Yoon, *T2fsnn: deep spiking neural networks with time-to-first-spike coding*, in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[107] W. Zhang and P. Li, *Spiking neural networks with laterally-inhibited self-recurrent units*, in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[108] A. Khodamoradi, K. Denolf, and R. Kastner, *S2n2: A fpga accelerator for streaming spiking neural networks*, in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 194–205, 2021.

[109] J. Shen, D. Ma, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, *Darwin: a neuromorphic hardware co-processor based on spiking neural networks*, *Science China Information Sciences* **59** (2016), no. 2 1–5.