2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2027-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPCA53966.2022.00031

Parallel Time Batching: Systolic-Array Acceleration of Sparse Spiking Neural Computation

Jeong-Jun Lee, Wenrui Zhang and Peng Li
Department of Electrical and Computer Engineering
University of California, Santa Barbara
{*jeong-jun, wenruizhang, lip*}*@ucsb.edu*

***Abstract*—Spiking Neural Networks (SNNs) are brain-inspired computing models incorporating unique temporal dynamics and event-driven processing. Rich dynamics in both space and time offer great challenges and opportunities for efficient processing of sparse spatiotemporal data compared with conventional artificial neural networks (ANNs). Specifically, the additional overheads for handling the added temporal dimension limit the computational capabilities of neuromorphic accelerators. Iterative processing at every time-point with sparse inputs in a temporally sequential manner not only degrades the utilization of the systolic array but also intensifies data movement.**

In this work, we propose a novel technique and architecture that significantly improve utilization and data movement while efficiently handling temporal sparsity of SNNs on systolic arrays. Unlike time-sequential processing in conventional SNN accelerators, we pack multiple time points into a single time window (TW) and process the computations induced by active synaptic inputs falling under several TWs 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. We further boost the utilization efficiency by simultaneously scheduling non-overlapping sparse spiking activities onto the array. The proposed architectures offer a unifying solution for general spiking neural networks with commonly exhibited temporal sparsity, a key challenge in hardware acceleration, delivering 248X energy-delay product (EDP) improvement on average compared to an SNN baseline for accelerating various networks. Compared to ANN based accelerators, our approach improves EDP by 47X on the CIFAR10 dataset.

***Keywords*-Spiking Neural Networks (SNNs), Neural Network Accelerator, Systolic Array, Parallel Processing**

I. Introduction

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].

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., [8]–[12], much less research has been devoted to SNN hardware accelerator architectures [13]–[17]. 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

978-1-6654-2027-3/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCA53966.2022.00031

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

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 [13]. 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 [18,19]. 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.

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

The main contributions of this work are:

Parallel Time Batching (PTB) - 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. PTB significantly improves latency and energy dissipation by efficiently handling the spatially and temporally sparse nature of general spiking models.

Spatiotemporally-non-overlapping Spiking Activity Packing (StSAP) - We identify non-overlapping structures of time batches and maximize systolic array utilization by scheduling an increased number of time batches onto the array, leading to further improved latency.

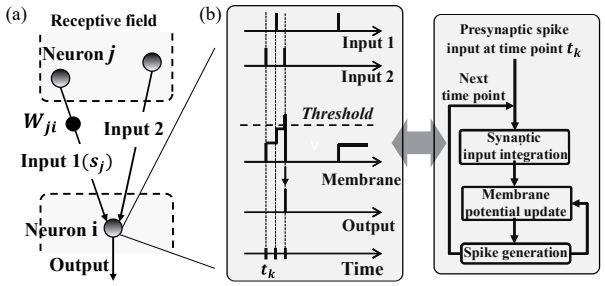

Figure 1: (a): Layer operation in SNNs. (b): Main steps of spatiotemporal operation in a spiking neuron.

Systolic array-based Accelerator Architecture - We propose a systolic array-based architecture capable of exploiting PTB and StSAP. We optimize the key architectural parameters of the proposed accelerator, and demonstrate significantly improved energy efficiency and latency.

We evaluate the proposed techniques with a spiking CNN (S-CNN) architecture simulator based on high-performance S-CNNs trained using state-of-the-art SNN training methods [20] on realistic neuromorphic datasets including DVS-Gesture [21] and CIFAR10-DVS datasets [22], and synthetic spiking based AlexNet [23] model. We examine how temporal granularity in terms of the time window (TW) and the proposed techniques PTB and StSAP impact data movement, utilization and energy efficiency. The proposed architecture and techniques significantly improve the energy efficiency, latency, and energy-delay product (EDP) by 248X on average, compared to a baseline systolic array architecture.

II. Background

A. 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 CNN hardware accelerators are based on 8- to 16-bit precision [8,12,24,25], which may be further compressed using techniques such as weight quantization [26,27]. 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 [13,14].

While integration and activation steps in ANNs exclude temporal information, a spiking neuron integrates its inputs over time, as shown in Fig. 1(b). The spatiotemporal information processing in SNNs empower various models

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

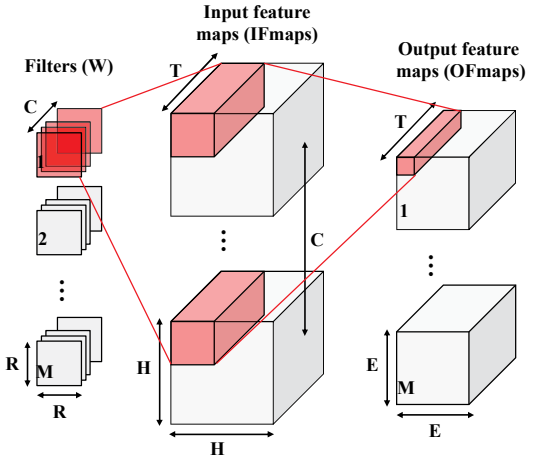

Figure 2: Computation of convolution layer in S-CNNs.

and applications [3,28,29]. However, the added temporal dimension causes intertwined spatiotemporal interactions, rendering SNN hardware accelerators to confront complex data movement/computation, which we address in later sections.

B. SNN Basics

Time point is a minimum unit of time in SNNs. At each time point *t*, operations in a single spiking neuron comprise three main steps as shown in Fig. 1: 1) integration of spike inputs from its receptive field (pre-synaptic spike inputs), 2) membrane potential update based on the integrated spike inputs and the membrane potential of the previous time point, 3) conditional generation of a spike output whenever the updated membrane potential exceeds a pre-determined threshold. The above three steps can be represented as:

Step 1: Synaptic input integration at t_k:

$$p_i^O[t_k] = \sum_{j=1}^{M^{RF}} w_{ji} \times s_j^{RF}[t_k] \tag{1}$$

Step 2: Membrane potential update:

$$v_i^O[t_k] = v_i^O[t_{k-1}] + p_i^O[t_k] - V_{leak}^l \tag{2}$$

Step 3: Conditional spike output generation:

$$s_i^O[t_k] = \begin{cases} 1, & \text{if } v_i^O[t_k] \geq V_{th}^O \quad \rightarrow v_i^O[t_k] = 0 \\ 0 & \text{else} \qquad\qquad \rightarrow v_i^O[t_k] = v_i^O[t_k] \end{cases} \tag{3}$$

where the *RF* and *O* represents the receptive field from the pre-synaptic layer, and the output (post-synaptic) layer, and j and i represent the neuron indices in the two layers, respectively, as shown in Fig. 1(a). $p_i^O[t_k]$, $v_i^O[t_k]$, and $s_i^O[t_k]$ denote the integrated partial sum of the spike inputs from the receptive field, membrane potential and spike output of the neuron i in the post-synaptic layer at time t_k, respectively. w_{ji} is the feedforward synaptic weight between neurons i and j, and M^{RF} is the number of neurons in the receptive field. V_{th} and V_{leak} are the firing threshold and leaky parameter, respectively. We distinguish the two most popular spiking neuron models, i.e., leaky integrate-and-fire (LIF) model [30] or integrate-and-fire (IF) model [31], depending on whether the leaky parameter is considered or not. In the above, **Step 1** and **Step 2** constitute the dominant complexity of hardware acceleration due to their large computational overhead.

Table I: 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

C. Basics of Spiking CNNs (S-CNNs)

The proposed architecture accelerates a given deep SNN consisting of multiple fully-connected and/or convolutional layers in a layer-by-layer manner. The fundamental operations of a single spiking neuron in spiking-CNNs (S-CNNs) follow the aforementioned three main steps (1)∼(3) where the computation involves multiple filters:

At a given time-point t_k,

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

$$\mathbf{P}[m][x][y][t_k] = \sum_{c=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} \mathbf{W}[m][c][i][j] \times \mathbf{I}[c][Ux+i][Uy+j][t_k] \tag{4}$$

Step 2: Membrane potential update:

$$\mathbf{V}[m][x][y][t_k] = \mathbf{V}[m][x][y][t_{k-1}] + \mathbf{P}[m][x][y][t_k] \tag{5}$$

Step 3: Conditional spike output generation:

$$\mathbf{O}[m][x][y][t_k] = \begin{cases} 1, & \text{if } \mathbf{V}[m][x][y][t_k] \geq V_{th}^O : \mathbf{V}[m][x][y][t_k] = 0 \\ 0 & \text{else} : \mathbf{V}[m][x][y][t_k] = \mathbf{V}[m][x][y][t_k] \end{cases} \tag{6}$$

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

Step 4: Move onto the next time-point (t_{k+1}), and repeat.

where **P**, **V**, **O**, **I** and **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

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

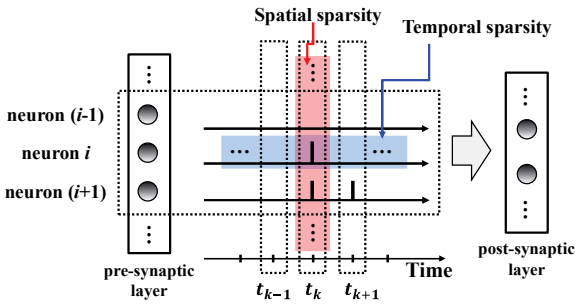

Figure 3: Spatial and temporal sparsity emergent in SNNs.

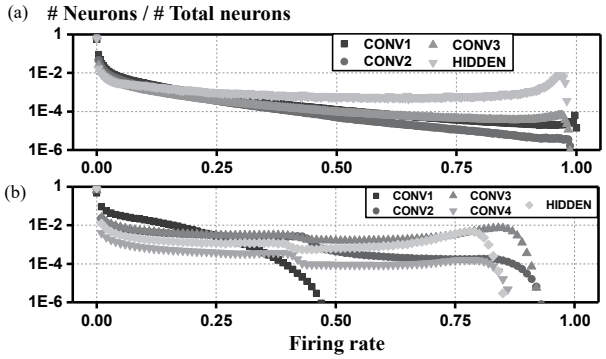

Figure 4: Normalized firing rate and distribution of neurons in (a): DVS-Gesture, and (b): CIFAR10-DVS.

Table I and Fig. 2. (4)∼(6) correspond to each of the three steps discussed in (1)∼(3).

D. 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 [11,12,32,33]. 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 this work.

III. 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.

A. 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 and postsynatic layers in Fig. 3.

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. 4(a) and (b) show the normalized average firing rate distributions of two well-trained SNNs based on the neuromorphic DVS-Gesture [21] and CIFAR10-DVS [22] 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 [13], 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 [14] abandons opportunities for performance improvements.

B. 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 [15,34,35]. 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. 7. From the memory point of

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

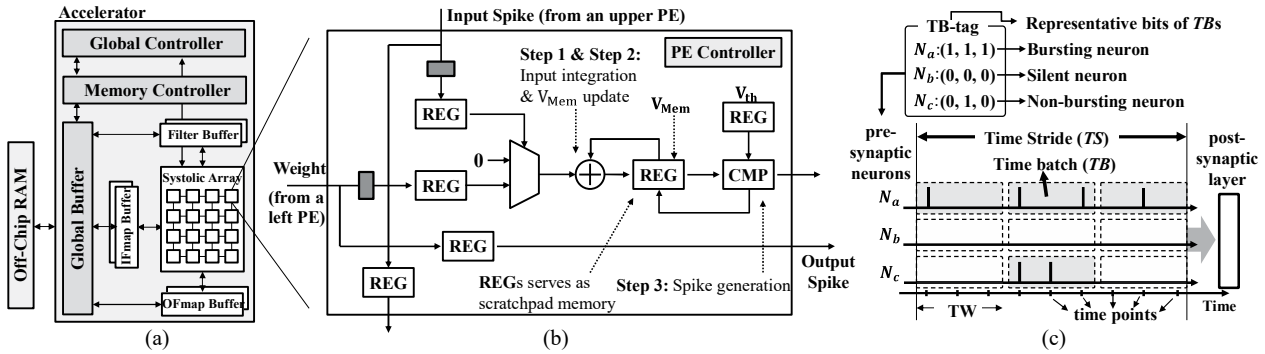

Figure 5: (a): Overall architecture. (b): Simplified schematic representation of the processing element (PE) in systolic array. (c): Schematic representation of *time point*, *time batch* (*TB*), *TB*-tag and *time stride* (*TS*).

Table II: 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
[13]	Low	No	Yes	Low
[14]	High	Limited	No	High
Ours	High	Yes	Yes	High

Ref*: Conventional approach: [15,34,35]
Ref**: Industrial neuromorphic chips: TrueNorth [6], Loihi [7]
[a]: Applicability for general SNNs [b]: Parallel processing in time domain
[c]: Learning performance (achievable accuracy)

view, the time-sequential process requires alternating access to different weight matrices for different time points.

Other Existing SNN Accelerators - As discussed in Section I, [13] proposed an efficient method to accelerate temporally-encoded SNNs. However, [13] 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. [14] introduced dataflow optimization for SNNs while operating in the time domain. However, the tiling technique in [14] does not consider firing sparsity, has limited weight reuse, and may lead to a significant PE under-utilization and comprised energy efficiency. Table II 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.

IV. 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.

A. 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(a). As in Fig. 5(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 [14], 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 [36], [32]. The 2-D systolic array exploits spatial and temporal parallelisms for which spike input and weight data 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 paper, we focus on synaptic input integration, the dominant complexity of SNN acceleration.

B. 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.

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

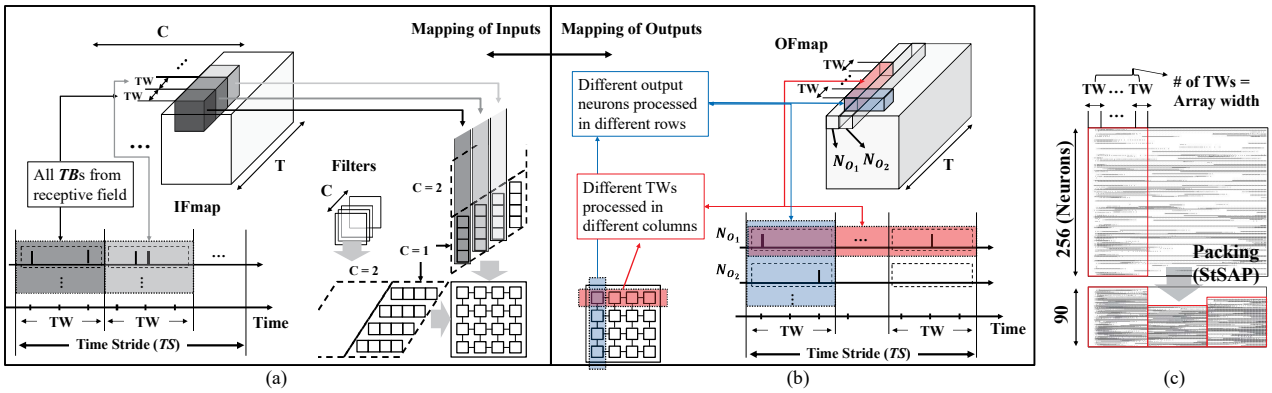

Figure 6: Mapping of the (a): inputs and (b): outputs into the systolic array. (c): Example of enhanced spike input density in DVS-Gesture dataset with temporally-non-overlapping spikingactivity packing (StSAP).

In Fig. 5(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(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(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(c). All other neurons are defined as non-bursting neurons, e.g., Neuron c (N_c) in Fig. 5(c).

C. 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.

1) 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. 6(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. 6(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 (4) $\sim$ (6) 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}:

$$p_{ji}^{O}[t_k,..,t_{k+TW-1}] = w_{ji} \times s_j^{RF}[t_k,..,t_{k+TW-1}]$$
$$p*_i^{O}\,[t_k,..,t_{k+TW-1}] = \sum_{j=1}^{M^{RF}} p_{ji}^{O}[t_k,..,t_{k+TW-1}] \quad (7)$$

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, ... , $(TW-1)$).

$$v_i^{O}[t_{k+m}] = p*_i^{O}\,[t_{k+m}] + v_i^{O}[t_{k+m-1}]$$
$$s_i^{O}[t_{k+m}] = \begin{cases} 1, \text{if } v_i^{O}[t_{k+m}] \geq V_{th}^{O} : v_i^{O}[t_{k+m}] = 0 \\ 0 \qquad \text{else}: \quad v_i^{O}[t_{k+m}] = v_i^{O}[t_{k+m}] \end{cases} \quad (8)$$

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 (1) $\sim$ (6). PTB groups **Step 1** in (4) across multiple time-points with the batch size TW as in (7). With mapping of the computations into PEs as described in Fig. 6, PTB enables parallel processing in both 1) space - for output neurons at different positions, and 2) time - executing different TWs, for **Step A** in (7), the dominant complexity.

2) Energy reduction: To exploit unstructured firing sparsity as shown in Fig. 4, PTB minimizes the weight access with two different types of reuse, as shown in Fig. 7(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. 7(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.

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

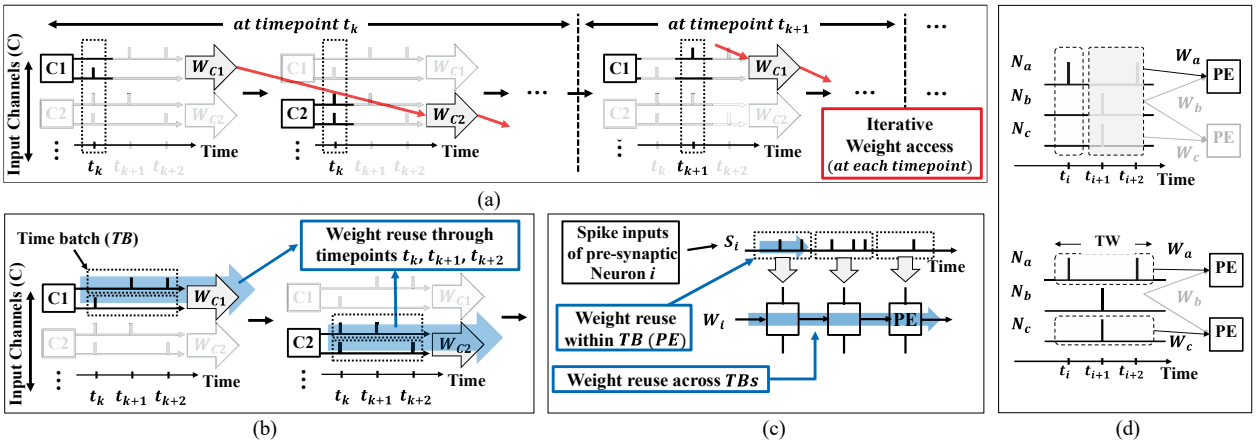

Figure 7: Simplified schematic representations of (a): Conventional approach which lacks parallel processing in time domain (executions are performed in time-serial manner). It requires alternating weight access. (b): Proposed approach using parallel time batching (PTB). (c): Weight reuse within, and across TBs. (d): Hiding the absence of spike with TB (grouped spiking activity).

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. 7(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. 7(c). In summary, PTB enables weight data reuse within each TB (PE) and across different TBs (PEs).

3) 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. 7(d).

D. 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.

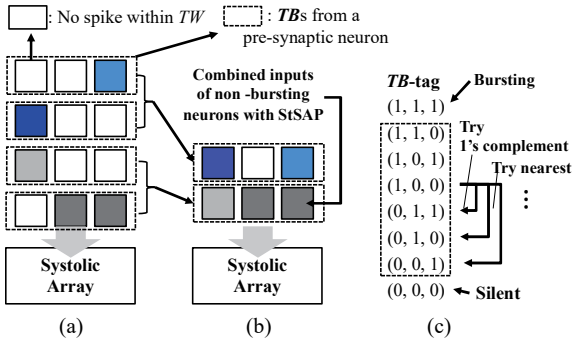

Figure 8: Schematic representation of StSAP. Mapping of the spike inputs from non-bursting neurons (a): without StSAP, (b): with StSAP. (c): Greedy policy applied to find nearest 1's complement based on TB-tag.

1) 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, 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

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

Table III: 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 IV: 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 $\times$ 8-bit
DRAM Bandwidth	30GB/sec
Bit precisions	Weight/Membrane Potential - 8-bit Input/Output Spike - TWS $\times$ 1-bit (TWS: TW size)

Fig. 8(c). We limit the number of neurons for packing to two to simplify the packing process.

2) 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. 8, for example, four TBs from non-bursting neurons are packed into two with StSAP. Fig. 6(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 computations.

Importantly, StSAP directly points to the actual activation values and fundamentally different from the existing packing strategies for DNNs [11,12,37]–[41]. For example, [11] and [12] 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, [11] and [12] 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.

V. Evaluation Methodology

We introduce an analytic architecture simulator to support unique features in SNNs and trace data movement for assessing the latency and energy dissipation. The user-specified inputs for the simulator is summarized in Table III.

A. Modeling systolic array & memory hierarchy

Systolic array - The developed simulator adopts a systolic array as a central compute substrate. The array comprises tiled processing elements (PEs) with unidirectional links. The structure of the PEs is detailed in Section IV. We use a fixed number of PEs for a fair comparison. In particular, we use a 128-PE systolic array. Similar sizes have been adopted in other works [8,13]. While using a fixed number of PEs, we consider different shapes of the array, as the array dimension is an important variable that impacts the performance of the accelerator. Mostly, we analyze the impact of TW size or our packing strategy based on 16×8 array by default.

Memory hierarchy - We follow the standard practice of three-level memory hierarchy for memory-intensive neural computations [8,36,42]. Similar to many other analytic models [10,32,36], 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. Architecture specifications are summarized in Table IV.

B. Performance modeling

The simulator generates unique addresses for each data type with respect to the inputs/outputs for the array. 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 [14,32,36].

Latency - The systolic array fetches the required data from the working buffer whenever the data is ready, pursuing stall-free operation while the loading buffer continuously seizes the data needed for the following computation. Therefore, the resulting latency per array iteration is estimated with the worst delay between data access and array computation. The total latency is calculated with sum of all latencies.

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 [8,14,32,36]. Using CACTI [43] configured for 32nm 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 [36].

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

Table V: CONV/FC layer shape configurations in three different networks used in this work.

Dataset	Layer	H	R	E	C	M
DVS-Gesture	CONV1	32	3	32	2	64
(Actual data)	CONV2	32	3	32	64	128
Timesteps: 300	CONV3	16	3	16	128	256
	FC1	8	8	1	256	256
	FC2	1	1	1	256	11
CIFAR10-DVS	CONV1	42	3	40	2	128
(Actual data)	CONV2	40	3	40	128	128
Timesteps: 100	CONV3	20	3	20	128	128
	CONV4	20	3	20	128	256
	FC1	10	10	1	256	1024
	FC2	1	1	1	1024	10
AlexNet	CONV1	224	11	55	3	96
(Synthesized)	CONV2	27	5	27	48	256
Timesteps: 300	CONV3	13	3	13	256	384
	CONV4	13	3	13	192	384
	CONV5	13	3	13	192	256
	FC1	6	6	1	256	4096
	FC2	1	1	1	4096	4096
	FC3	1	1	1	4096	1000

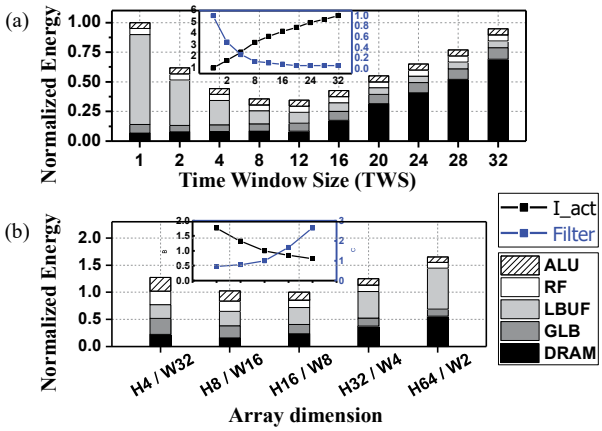

Figure 9: Energy dissipation breakdown of CONV2 in DVS-Gesture with different (a): TW size. (b): array size (TW=8).

C. Benchmarks

We use a comprehensive set of S-CNN spiking activity data, either actual or synthetic, to evaluate the proposed architecture. Table V shows the shape configurations of each convolutional (CONV) and fully-connected (FC) layer.

DVS-Gesture, CIFAR10-DVS - The images/gestures are recorded by a dynamic vision sensor (DVS) camera and converted into neuromorphic data with spikes spanned through time. We train two S-CNNs with the state-of-the-art algorithm [20], using the widely adopted neuromorphic DVS-Gesture [21] and CIFAR10-DVS [22] datasets, respectively, and evaluate the architecture performance using the actual spiking activity data extracted from the trained models. The DVS-Gesture dataset consists of 1,463 test samples with 11 different classes of hand gestures, and CIFAR10-DVS comprises 10,000 test samples with 10 different classes of images. To speed up the simulation, each sample is converted into a 300-/100-time step binary matrix by compressing the time resolution.

AlexNet - Furthermore, we adopt the network structure of the widely used AlexNet [23] DNN model with synthetic spiking activities. The simulation takes 300 timesteps and the averaged firing activities distribution is set based on the activity data from the DVS-Gesture and CIFAR10-DVS datasets.

VI. Results

We evaluate the performance of the proposed architecture focusing on the impact of the proposed PTB and StSAP described in Section IV based on the setups described in Section V. 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.

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

1) 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

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

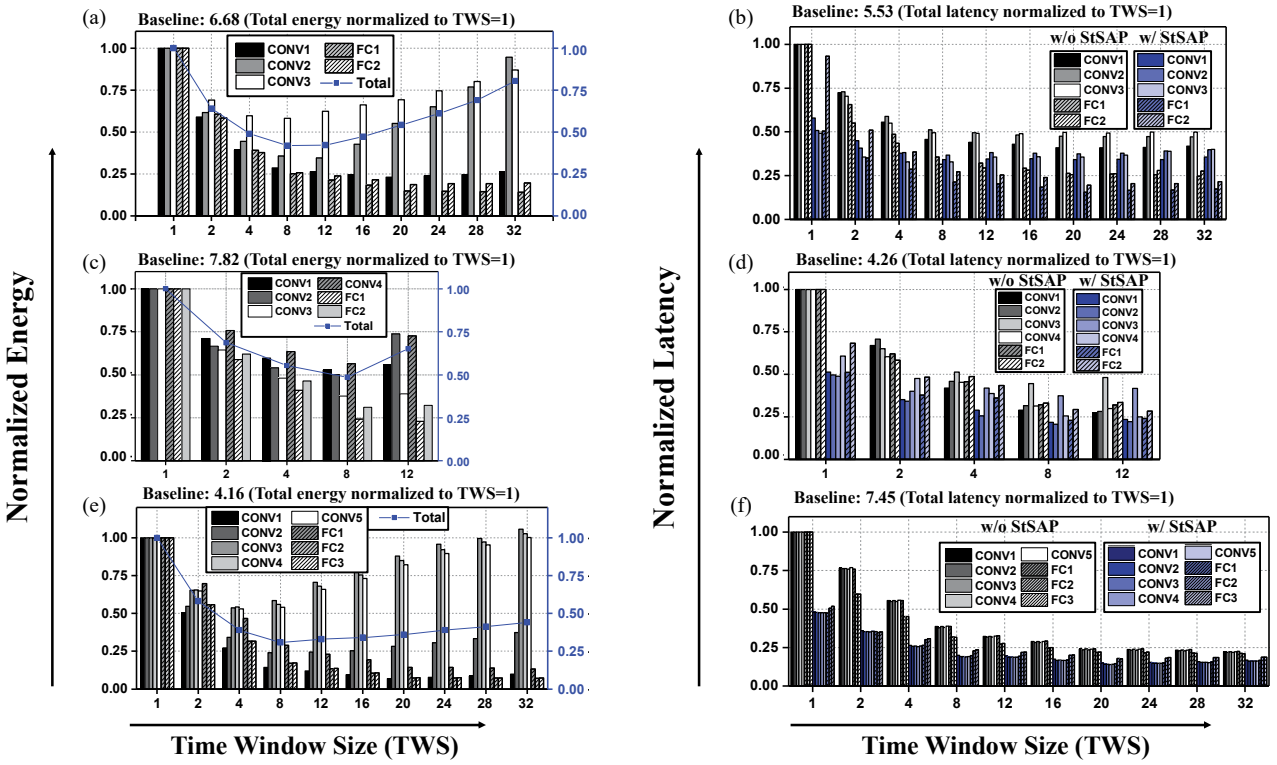

Figure 10: Normalized energy dissipation and latency of layers with different TW sizes, with- and without StSAP, in each dataset. **(a),(b): DVS-Gesture, (c),(d): CIFAR10-DVS, and (e),(f): Alexnet**. PTB with non-optimized TW size (TWS=1) improves the total energy dissipation and latency by DVS-Gesture: 6.68X and 5.53X, CIFAR10-DVS: 7.82X and 4.26X, and Alexnet: 4.16X and 7.45X, over the baseline.

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.

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. 9(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 VI-B.

2) 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. 9(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.

B. 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 [14] 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.

1) 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

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

weight data reuse. The opposite can be said for CONV layers with many input channels but small sized filters.

2) 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. 10. 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. 10 (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 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. 10 (e).

Latency: We observe a clear improvement of latency by using PTB in all three networks, as shown in Fig. 10. As discussed in Section IV, 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.

3) 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. 10. 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

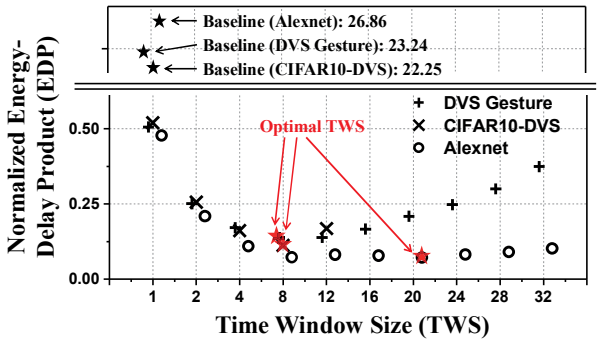

Figure 11: Total energy-delay product (EDP) of three different benchmarks. EDP values are normalized to the baseline result, which exclude merging and TW size optimization.

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.

4) 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. 11 shows the normalized EDP of three different networks with varying TW sizes. The baseline is based on the approach proposed in [14], 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. 10 (e). Since the overheads of later CONV 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.

VII. Discussion

SNNs have shown great potentials and results in both energy efficiency and performance [3,20,44]. 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 [13,45]. We discuss broader impacts and promises of our work.

Machine learning performance: As shown in Table VI, SNNs achieved comparable performance to ANNs with

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

Table VI: Performance comparison of ANNs and SNNs.

Workload	ANN	SNN
MNIST	99.80% [46], 99.04% [26]	99.62% [3], *99.53% [20]
CIFAR10	90.49% [47], 93.75% [48]	*91.41% [20], 93.58% [49]
DVS-Gesture	92.01% [48], 93.75% [48]	95.49% [50], *93.75% [20]
CIFAR10-DVS	31.00% [51], 52.40% [52]	58.10% [44], *56.86% [20]

*: This work.

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: [13] showed outperforming energy efficiency over ANNs [8] 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 [32,47] for energy dissipation and latency, respectively, as shown in Fig. 12(b). We adopted same network structure in [20,47] and architecture specifications using [32] for fair comparison. We adopted the training algorithm in [20] 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. 12(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 12(a), firing rate of well-trained networks ranges from 1∼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. 12(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 is possible if the optimal TW size

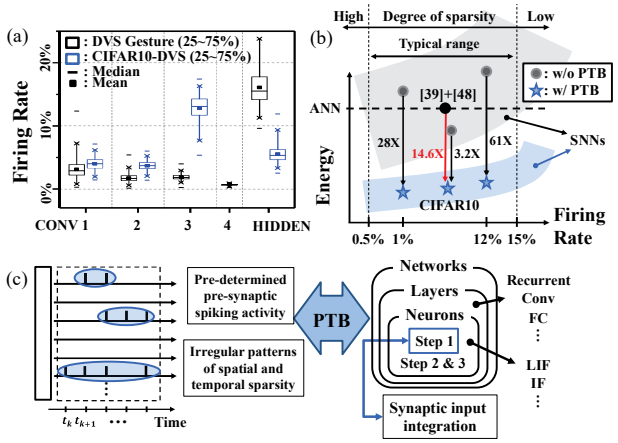

Figure 12: (a) Firing rate of well-trained networks. (b) PTB significantly improves energy efficiency across wide range of sparsity-levels. With PTB, SNN showed better result than ANN. (c) PTB supports diverse family of spiking models.

is chosen offline, based on the given architecture, sparsity-level and layer type.

VIII. Conclusion

Unique features of SNNs pose challenges for developing efficient accelerators. Especially, spatial and temporal sparsity emerged in SNNs causes unstructured sparsity of spiking activities and degrades the overall performance of accelerators. This paper 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. 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.

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGEMENTS

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 Grants No. 1948201 and No. 2000851.

DISCLAIMER

This paper was prepared as an account of work sponsored by agencies of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] Y. Hao *et al.*, "A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule," *Neural Networks*, vol. 121, pp. 387–395, 2020.

[2] Y. Zhang *et al.*, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

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

[4] Y. Jin *et al.*, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," *Advances in neural information processing systems*, vol. 31, pp. 7005–7015, 2018.

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

[6] F. Akopyan *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*, vol. 34, no. 10, pp. 1537–1557, 2015.

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

[8] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[9] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[10] Y. Shen *et al.*, "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)*. IEEE, 2017, pp. 93–100.

[11] X. He *et al.*, "Sparse-tpu: Adapting systolic arrays for sparse matrices," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[12] H. Kung *et al.*, "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*, 2019, pp. 821–834.

[13] S. Narayanan *et al.*, "Spinalflow: an architecture and dataflow tailored for spiking neural networks," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 349–362.

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

[15] D. Neil *et al.*, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

[16] E. Painkras *et al.*, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[17] C. Mayr *et al.*, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv preprint arXiv:1911.02385*, 2019.

[18] I. M. Comsa *et al.*, "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. [Online]. Available: http://dx.doi.org/10.1109/ICASSP40776.2020.9053856

[19] S. R. Kheradpisheh *et al.*, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, p. 56–67, Mar 2018. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2017.12.005

[20] W. Zhang *et al.*, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[21] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[22] H. Li *et al.*, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017.

[23] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[24] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[25] X. Lian *et al.*, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[26] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

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

[28] G. Bellec *et al.*, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797.

[29] S. R. Kheradpisheh *et al.*, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[30] W. Gerstner *et al.*, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[31] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[32] A. Samajdar *et al.*, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.

[33] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[34] Y. Cao *et al.*, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[35] S.-Q. Wang *et al.*, "Sies: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array," *Journal of Computer Science and Technology*, vol. 35, pp. 475–489, 2020.

[36] H. Kwon *et al.*, "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*, 2019, pp. 754–768.

[37] Y.-H. Chen *et al.*, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[38] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[39] A. Gondimalla *et al.*, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 151–165.

[40] A. Delmas Lascorz *et al.*, "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*, 2019, pp. 749–763.

[41] K. Kanellopoulos *et al.*, "Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix opera-

[52] A. Sironi *et al.*, "Hats: Histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp.

tions," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 600–614.

[42] S. Li *et al.*, "An fpga design framework for cnn sparsification and acceleration," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 28–28.

[43] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 1, pp. 1–24, 2009.

[44] Y. Wu *et al.*, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

[45] Z. Du *et al.*, "Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 494–507.

[46] L. Wan *et al.*, "Regularization of neural networks using dropconnect," in *International conference on machine learning*. PMLR, 2013, pp. 1058–1066.

[47] Y. Wu *et al.*, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 1311–1318.

[48] W. He *et al.*, "Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences," *Neural Networks*, vol. 132, pp. 108–120, 2020.

[49] S. Deng *et al.*, "Optimal conversion of conventional artificial neural networks to spiking neural networks," *arXiv preprint arXiv:2103.00476*, 2021.

[50] W. Zhang *et al.*, "Skip-connected self-recurrent spiking neural networks with joint intrinsic parameter and synaptic weight training," *Neural Computation*, vol. 33, no. 7, pp. 1886–1913, 2021.

[51] G. Orchard *et al.*, "Hfirst: A temporal approach to object recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 10, pp. 2028–2040, 2015.
1731–1740.

Authorized licensed use limited to: Univ of Calif Santa Barbara. Downloaded on September 18,2022 at 18:56:45 UTC from IEEE Xplore. Restrictions apply.