Systolic Array Acceleration of Spiking Neural Networks with Application-Independent Split-Time Temporal Coding

Jeong-Jun Lee
University of California, Santa Barbara
California, USA
jeong-jun@ucsb.edu

Peng Li

ABSTRACT

Spiking Neural Networks (SNNs) are brain-inspired computing models with event-driven based low-power operations and unique temporal dynamics. However, temporal dynamics in SNNs pose a significant overhead in accelerating neural computations and limit the computing capabilities of neuromorphic accelerators. Especially, unstructured sparsity emergent in both space and time, i.e., across neurons and time points, and iterative computations across time points cause a primary bottleneck in data movement.

In this work, 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. The proposed architecture delivers 97X latency and 78X energy efficiency improvements on average over a conventional SNN baseline on different benchmarks.

CCS CONCEPTS

• **Hardware → Hardware accelerators**.

KEYWORDS

spiking neural networks, accelerators, computer architecture

1 INTRODUCTION

Non-spiking artificial neural networks (ANNs) process information with continuous-valued signals representing averaged firing rates of neurons resulting from activation functions such as rectified linear unit (ReLU) and sigmoid [1]. In contrast, spiking neural networks (SNNs) handles unraveled information in space and time, i.e., across different neurons (space) and different time points (time), with explicitly modeled all-or-none firing spikes. As reported in recent studies, spatial and temporal dynamics with biologically inspired [8] and backpropagation based [7, 12, 16] SNN training algorithms have demonstrated competitive performances for various tasks.

From a hardware acceleration point of view, SNNs have considered better positioned for low-power operations than ANNs with biologically plausible computing models including event-driven processing and binary-valued signals. However, computations along the temporal dimension and unstructured sparsity in both space and time complicate the hardware acceleration of spike-based models. The unique temporal dimension in SNNs offers an opportunity in processing complex spatiotemporal data but introduces iterative and unstructured data movement at each time point.

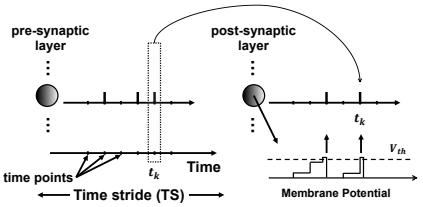


Figure 1: Operations in spiking neural networks (SNNs)

This work aims to develop a systolic array-based architecture to tap the full potential of SNN acceleration with 1) Split-Time Temporal coding (STT) and 2) Integration Through Time (ITT).

Split-Time Temporal coding (STT) : 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.

Integration Through Time (ITT) : ITT 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.

Systolic array-based Architecture : We develop a systolic array-based architecture supporting STT and ITT. The proposed architecture is capable of accelerating various types of layers. We overcome the causality and tightly coupled dependencies by using the prefix sum without additional resources.

2 BACKGROUND

As shown in Fig. 1, 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:

$$\vec{p}^{Post}[t_k] = \mathbf{W}_{Post,Pre} \times \vec{s}^{Pre}[t_k] \tag{1}$$

Step 2: Membrane potential update at t_k:

$$\vec{v}^{Post}[t_k] = \vec{v}^{Post}[t_{k-1}] + \vec{p}^{Post}[t_k] - V_{leak}^{Post} \tag{2}$$

Step 3: Conditional spike output generation at t_k:

$$\vec{s}^{Post}[t_k] = \boldsymbol{f}(\vec{v}^{Post}[t_k]) \tag{3}$$

$$\boldsymbol{f}(v_i^{Post}[t_k]) = \begin{cases} 1, & \text{if } v_i^{Post}[t_k] \geq V_{th}^{Post} : v_i^{Post}[t_k] = 0 \\ 0 & \text{else} : v_i^{Post}[t_k] = v_i^{Post}[t_k] \end{cases} \tag{4}$$

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

*J. Lee is currently with Meta Platforms Inc. This work was performed while he was affiliated with the University of California, Santa Barbara.

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, (**Step 1**) incurs matrix-vector multiplication and takes place at each time point, comprising the dominant complexity. Importantly, the above steps are repeated at each time point, across all time points.

Processing neural computations of a recurrent layer in SNNs follow the same three steps in the feedforward layer with additional synaptic inputs.

Step 1*: Feedforward synaptic input integration at t_k:

$$\vec{p}_F^{Post}[t_k] = \mathbf{W}_{Post,Pre} \times \vec{s}^{Pre}[t_k] \quad (5)$$

$$\vec{p}_R^{Post}[t_k] = \mathbf{W}_{Post,Post} \times \vec{s}^{Post}[t_{k-1}] \quad (6)$$

$$\vec{p}^{Post}[t_k] = \vec{p}_F^{Post}[t_k] + \vec{p}_R^{Post}[t_k] \quad (7)$$

where $\vec{p}_R^{Post}[t_k]$ is a vector, representing the partial sum of the recurrent input integration.

3 SPLIT-TIME TEMPORAL CODING (STT)

3.1 Proposed STT

We propose a novel technique to locally employ coding and sparisification 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 is to employ local structurization and sparsification and improve the computational/data movement overhead by reducing the redundancy in locally rate-coded firing activities on a TW basis while retaining local rate information by using prefix sum. Importantly, STT is universally applicable for accelerating spiking models, with flexibility in choosing the TW size. The spike timing of the single spike coded for each TW carries firing rate information with time-left-from-first-spike (TFFS), as shown in Fig. 2. All layers in the network operate on TW-based local coding based on the proposed STT, with the following rules:

Rule 1. We limit the maximum firing count of each neuron in a TW 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 count within a TW is represented by the timing of a single spike. As such for the input layer, the spike information of original input firing activity is converted with STT based on the number of spikes in each TW.

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

As shown in Fig. 3(a), we first convert the rate-coded original firing activities into STT-based firing activities at the input layer. For example, as in Fig. 2, if the TW size is 5 and the number of spikes in a TW is 4, the time-left-from-first-spike (TFFS) in the corresponding TW is determined by: TFFS = (TW size) - TTFS = 5 - 1 = 4, representing the firing rate of the TW. At the output layer, STT-based firing activities are decoded to firing rate for the decision making. For example, the spike train in Fig. 3(c) is decoded by integrating rates across TWs: $\sum$(TFFS) = $\sum$ (TW size)-(TFFS) = 2 + 4 + 1 = 7, following **Rule 3**.

3.2 STT-based Acceleration

STT-based hardware acceleration significantly simplifies the input integration step, the dominant computational complexity. First, STT reduces the repeated weight access across multiple time points to a single weight access per input neuron for a given TW. Since an input neuron fires up to once in a TW, the corresponding weight is used only once for input integration.

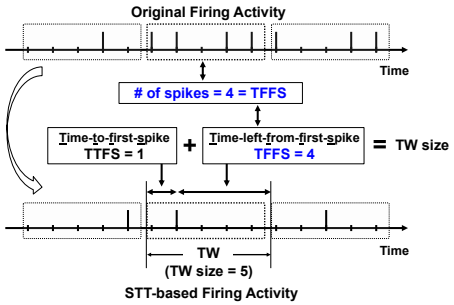


Figure 2: Local structurization and sparsification with the proposed STT. Time-left-from-first-spike (TFFS) presents the firing rate of the corresponding TW.

Second, to retain the local and also global information, we use the prefix sum of the STT-based integrated synaptic inputs in a TW while this efficient process is still based on a single spike per TW, following **Rule 2**. As will be shown in Fig. 6, 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.

Finally, STT allows parallel acceleration through time via using a small amount of memory for each time point of a TW. For the case in Fig. 3(b), conventionally, the input integration step requires accessing weight data 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} sequentially. These unstructured firing patterns across different neurons and time points render repeated weight access without data reuse. Differently, with STT, W_A is integrated to the partial sums (Psums) at t_{k+1}, W_B is integrated to the Psums at t_{k+3}, and W_C is integrated to the Psums at $t_k$in parallel.

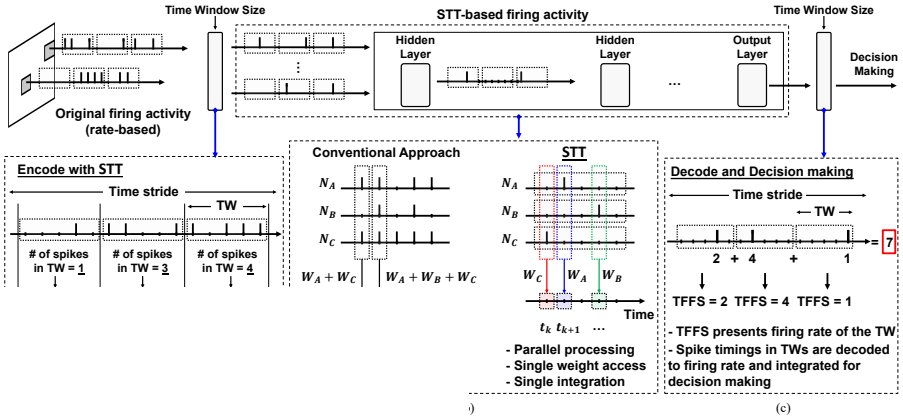
4 PROPOSED ARCHITECTURE

4.1 Overview of the Proposed Architecture

Fig. 5 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, and a systolic array composed of tiled processing elements (PEs) with unidirectional links. As shown in Fig. 5(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 the corresponding time point with a simple decoder, and the scratch-pad in each PE stores the Psums of all time points in a given TW.

4.2 Integration Through-Time (ITT)

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



rations. (a): STT-encoder at the input layer (b): Comparison osed STT-based approach (c): STT-decoder at the output layer

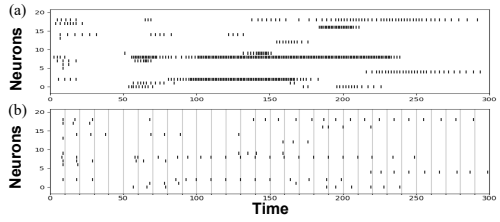


Figure 4: Spike raster plot of 20 neurons from the recurrent layer for accelerating NTIDIGITS. (a): Original firing activities without using STT and (b): STT-based firing activities with TW size = 10.

unit of workload and maps spiking activities in multiple TWs onto the systolic array, concurrently. As shown in Fig. 5(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 (1). Integration of synaptic inputs across multiple time points with ITT can be expressed by modifying (1) as:

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

$$\begin{aligned}
&\mathbf{p}^{Post}[TW_n, TW_{n+1}, ..., TW_{n+m}] \\
&= \mathbf{p}^{Post}[(t_{k(n-1)+1}, ..., t_{kn}), ..., (t_{k(n+m-1)+1}, ..., t_{k(n+m)})] \\
&= \mathbf{W}_{Post,Pre} \times \mathbf{s}^{Pre}[TW_n, TW_{n+1}, ..., TW_{n+m}] \\
&= \mathbf{W}_{Post,Pre} \times \mathbf{s}^{Pre}[(t_{k(n-1)+1}, ..., t_{kn}), ..., (..., t_{k(n+m)})]
\end{aligned} \tag{8}$$

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 TWs. 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 (2) ~ (4) and all the other expressions follows the definition described in (1) ~ (4).

4.3 Mapping to Systolic Array

With STT and ITT, our mapping strategy enables parallel processing in both 1) time: across time points and 2) space: across output neurons, which significantly improves data movement and processing time.

Mapping Input and Outputs: As shown in Fig. 5(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: 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. 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 3, 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.

4.4 STT-based Layer Acceleration

The operations in a single PE follow the three steps (1) ~ (4) with an AC unit and a small scratch-pad shared through the steps, as shown in Fig. 6(a). In Step 1, the synaptic input integration step, the 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 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. 6(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 3 and shown in Fig. 6(b), we utilize the

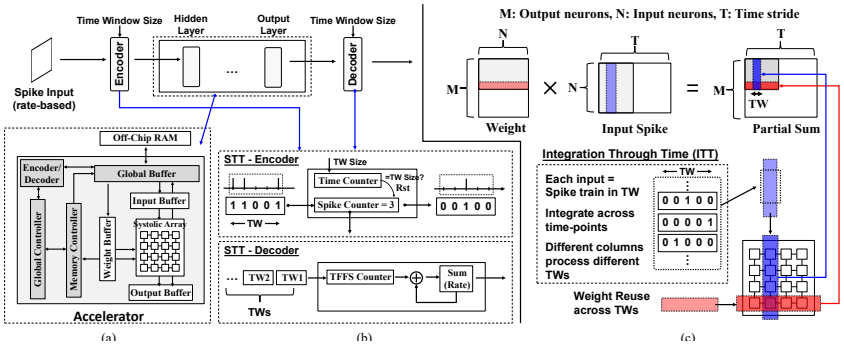


Figure 5: ... er, respecti...

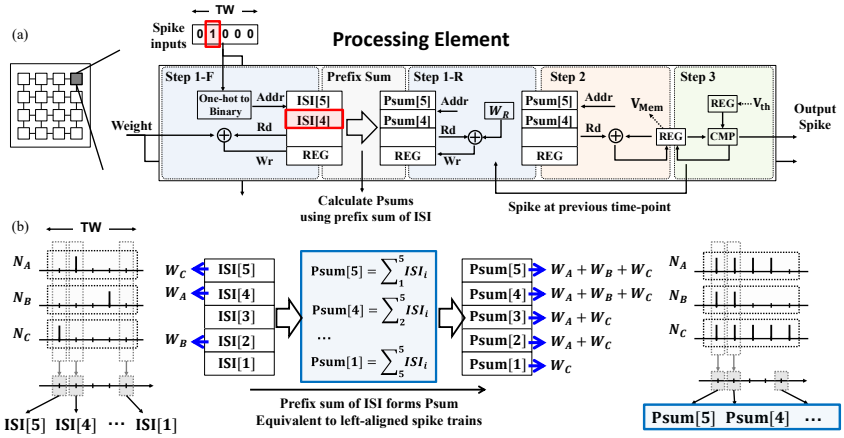


Figure 6: Schematic representations of (a): Operations in a PE for accelerating feedforward and recurrent layer (b): Calculating partial sums (Psums) using a prefix sum of the integrated synaptic inputs (ISI)

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. 6(b), the use of prefix sum yields the same Psum results as using left-aligned, rate codes where the rate equals TFFS.

For the rest of the operation, PE processes Step 2 and Step 3 with the integrated Psums. At a given time point t_k, the 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 that for feedforward acceleration except for one additional step for integrating recurrent synaptic inputs, denoted as Step 1-R in Fig. 6(a). To simplify the recurrent layer processing, we adopt the self-recurrent structure in [17].

5 EVALUATION METHODOLOGY

We develop an analytic architecture simulator to support various types of layers, unique characteristics in SNNs, and trace data access/movement for evaluating the latency and energy dissipation for accelerating a specific task. In Table 1, the user-defined inputs for the simulator are summarized.

5.1 Systolic-Array and Memory Modeling

Systolic array: A systolic array is a central computing unit of our simulator and fetches spike inputs and weights from the top and left edges, respectively. As in many other works, we use a

Table 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

128 processing elements (PEs) [5] in the systolic array along with double-buffered L1 caches to provide required data to the array.

Memory hierarchy: Similar to many other analytic architecture evaluation models, we adopt an off-load model with a three-level memory hierarchy. We follow the standard practice [9, 14] to use double-buffering to hide the latency for memory-intensive neural networks and especially separate L1 cache for each type of data Architecture specifications are summarized in Table 2.

5.2 Performance Modeling

With the dataflow, the simulator assigns unique addresses for each data and traces read and write in PEs and each level of the memory hierarchy. Following the estimation methods in many previous works [4, 9, 14], the simulator calculates latency, memory access and energy dissipation.

Table 2: Architecture specifications.

Components	Proposed Architecture
Number of PEs	128
ALU in PEs	Adder, Comparator - 8-bit

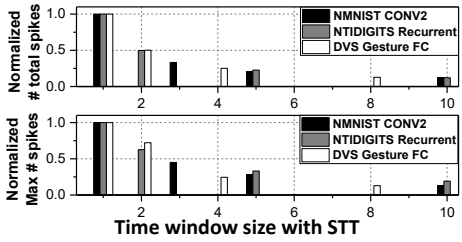


Figure 7: Normalized number of total spikes and maximum number of spikes in a neuron with different time window sizes.

Latency: The latency is estimated with the worst delay between data access from higher-level cache and computations in the array. The total latency is calculated by adding all latencies across the entire process.

Memory Access: For the given user-defined inputs, the simulator generates a dataflow that pre-determines the data loading onto the array. With the given memory size and data loading schedule, the simulator counts read/write in all levels of memory as in [9, 14]. For example, when a specific data is required for the array computation but is absent in the L1, it induces global buffer read and L1 write if the data is present in the global buffer.

Energy Dissipation: With CACTI model [10] configured for 32nm CMOS technology, energy dissipation in memory is calculated by multiplying the number of memory access and the energy per memory access. Energy dissipation for computations is evaluated by the number of required AC operations [9] based on the actual spiking activities in the network.

5.3 Training Algorithm and Benchmarks

Training Algorithm: All the reported machine learning performance are simulated on NVIDIA Titan XP GPU and the implementation of the proposed STT is conducted on Pytorch framework [13]. We adopt one of the the state-of-the-art SNN training method [16] for training the network.

Benchmarks: The proposed STT and ITT are evaluated on various image and speech tasks including neuromorphic image dataset N-MNIST [11], neuromorphic video dataset DVS-Gesture [2], and neuromorphic speech dataset N-TIDIGITS [3] with various layer types, i.e., fully connected, convolutional and recurrent, in the network.

6 RESULTS

6.1 STT: Temporal Information Compression

STT reconstructs the spike information with higher, but structured sparsity by dividing the time stride into multiple TWs, squeezing the entire spike information in each TW to the timing of a single spike. As introduced in Section 3, 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.

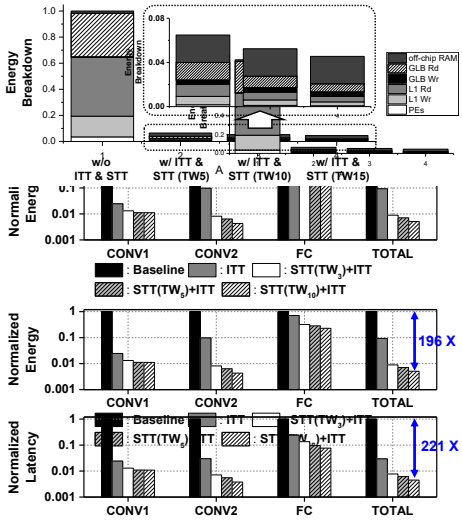


Figure 9: Normalized energy dissipation and latency of layers with different TW sizes for NMNIST.

Computational overhead: The number of spikes required for layer acceleration decreases with the TW size, so does the computational overheads by STT. Approximately, the number of required AC operations is inversely proportional to the TW size, as shown in Fig. 7.

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 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.

6.2 ITT: Data Reuse

ITT significantly improves data reuse by providing data sharing opportunities 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 the PEs in the same row.

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. 8. Clearly, larger TW sizes reduce access to the higher-level caches and improve energy dissipation. Compared to the conventional approach without the proposed ideas, we observe a huge improvement in memory access to the L1 cache and global buffer.

6.3 Comprehensive Evaluations

We examine how the proposed STT and ITT with the key architectural parameter TW size improve the overall performance.

Latency: We observe a huge improvement in latency by using STT and ITT in all three networks, as shown in Figs. 9. As discussed in Sections 6.1 and 6.2, 1) STT reduces latency of the computations in the array proportionally to the TW size by processing a TW instead of a time-point, and 2) ITT minimizes the additional delay due to stall cycles resulted from waiting for the required data, by reusing the weight data horizontally. However, after a certain TW size, the additional improvement with a much larger TW size decreases. This

Table 3: Performance on fully-connected, convolutional and recurrent networks: NMNIST, DVS-Gesture and NTIDIGITS. TWS denotes the applied time window size.

Neuromorphic MNIST

Method	Network	Accuracy	Timepoints
HM2BP [7]	400-400	98.88%	400
SLAYER [15]	500-500	98.95%	300
SLAYER [15]	CNN[a]	99.22%	300
TSSL-BP [16]	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 [6]	P4-512	52.78%	
LSTM* [6]	P4-512	88.19%	
TSSL-BP [16]	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.

N-TIDIGITS

Method	Network	Accuracy	Timepoints
HM2BP [7]	250-250	89.69%	300
BP (GRU) [3]	200-200-100	89.92%	
BP (LSTM) [3]	250-250	91.25%	
TSSL-BP [16]	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 [17]

is due to the fact that spikes are often clustered in a certain range in the time domain as shown in Fig. 4, and the number of TWs is the reciprocal of TW size. The proposed techniques improved the latency by 97X on average, across the three 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 data, as in CONV2 in NMNIST. Importantly, firing activities from a neuron are often clustered in time and in practice 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, STT may cause a local temporal information loss in a TW. We adopted the training algorithm in [16] and conducted STT-based inference test on well-trained networks with different TW sizes. We observe that the proposed STT can deliver competitive inference performance up to a certain TW size across various networks as in Table 3 while providing a significant improvements on hardware acceleration.

ML-HW Performance Trade-off: STT significantly reduces computational overhead by introducing local temporal resolution reduction per TW, tunable based on TW size while maintaining global temporal information of the original spikes without complex hyper parameter tuning. We use energy-delay product (EDP) to simultaneously consider latency and energy dissipation for evaluation of the proposed techniques and to analyze 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. 10, the ML-HW performance trade

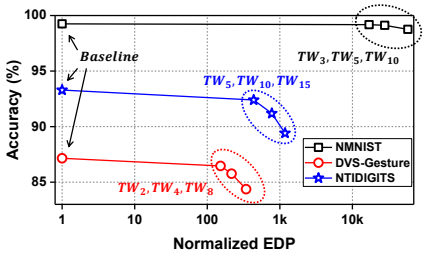


Figure 10: Machine learning performance (inference) - Accelerator performance (normalized EDP) tradeoffs on various datasets. STT and ITT are not applied on the baseline.

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 in Fig. 10.

7 ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grants No. 1948201 and No. 2310170.

REFERENCES

[1] Agarap, A. F. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).

[2] Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., 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.

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

[4] Chen, P.-Y., Peng, X., and Yu, S. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37*, 12 (2018), 3067–3080.

[5] Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits 52*, 1 (2016), 127–138.

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

[7] Jin, Y., Zhang, W., and Li, P. Hybrid macro/micro level backpropagation for training deep spiking neural networks. *Advances in neural information processing systems 31* (2018).

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

[9] Kwon, H., Chatarasi, P., Pellauer, M., Parashar, A., Sarkar, V., and Krishna, T. 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.

[10] Muralimanohar, N., Balasubramonian, R., and Jouppi, N. P. Cacti 6.0: A tool to model large caches. *HP laboratories 27* (2009), 28.

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

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

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (2019).

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

[15] Shrestha, S. B., and Orchard, G. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems 31* (2018).

[16] Zhang, W., and Li, P. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems 33* (2020), 12022–12033.

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