SATO: Spiking Neural Network Acceleration via Temporal-Oriented Dataflow and Architecture

Fangxin Liu[1,2], Wenbo Zhao[1], Zongwu Wang[1], Yongbiao Chen[1], Tao Yang[1], Zhezhi He[1], Xiaokang Yang[1,3] and Li Jiang[1,2,3]*

[1]Shanghai Jiao Tong University, [2]Shanghai Qi Zhi Institute

[3]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University

ABSTRACT

Event-driven spiking neural networks (SNNs) have shown great promise for being strikingly energy-efficient. SNN neurons integrate the spikes, accumulate the membrane potential, and fire output spike when the potential exceeds a threshold. Existing SNN accelerators, however, have to carry out such accumulation-comparison operation in serial. Repetitive spike generation at each time step not only increases latency as well as overall energy budget, but also incurs memory access overhead of fetching membrane potentials, both of which lessen the efficiency of SNN accelerators. Meanwhile, inherent highly sparse spikes of SNNs lead to imbalanced workloads among neurons that hurdle the utilization of processing elements (PEs).

This paper proposes SATO, a temporal-parallel SNN accelerator that accumulates the membrane potential for all time steps in parallel. SATO architecture contains a novel binary adder-search tree to generate the output spike train, which decouples the chronological dependence in the accumulation-comparison operation. Moreover, SATO can evenly dispatch the compressed workloads to all PEs with maximized data locality of input spike trains based on a bucket-sort-based method. Our evaluations show that SATO outperforms the previous ANN accelerator 8-bit version of “Eyeriss” by 30.9× in terms of speedup and 12.3×, in terms of energy-saving. Compared with the state-of-the-art SNN accelerator “SpinalFlow”, SATO can also achieve 6.4× performance gain and 4.8× energy reduction, which is quite impressive for inference.

1 INTRODUCTION

Biologically-inspired spiking neural networks (SNNs) use event-based models to simulate biological neurons and provide high prediction accuracy with minimal energy consumption [1, 5, 6, 16]. Spiking neurons are the main computing and storage units in SNNs that collect input spikes and emit output spikes according to the membrane potential, like their biological counterparts. Series of spikes, called spike trains, transmit information between neurons by their firing times and firing frequencies. In SNN implementations, the time window of a spike train is divided into time steps to support neuron calculation in a synchronized manner [16].

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530592

The neuron calculation in SNN has the potential to improve hardware efficiency because inputs are binary spikes [16]. In that case, weights are only added instead of being multiplied with input as conducted in deep artificial neural networks (ANNs), exempting expensive multiply operations in SNNs [6]. Based on this, several previous works, such as Neurogrid [2], IBM TrueNorth [1], Intel Loihi [5], NEBULA [18], Xilinx S2N2 [9], etc., have shown effective deployment of SNN in customized hardware for training and inference. IBM TrueNorth [1] processor is a digital implementation of SNN, consisting of many tiles that can simultaneously process the input signal of a single neuron in a single time step. Besides, to increase the parallelism inside tiles, SpinalFlow [14] utilizes PE arrays to process multiple neurons in parallel with sparse encoding. An Eyeriss-based SNN accelerator [3] describes an output stationary flow that can minimize the movement of partial-sums and save a portion of the time for accumulating membrane potential.

In spite of the significant advances, existing SNN accelerators still suffer from low energy efficiency and long processing latency under the time-driven mechanism, which updates all neurons at every time step [6, 16]. The main hurdle is the chronological accumulation of the membrane potentials in SNN computation, which demands a serial process of spikes at each time step. In such a procedure, a problem that we call the *chronological dependence* happens, i.e., SNNs require repetitive spike generation at each time step and depend on the aggregated potential of the previous time steps, resulting in higher inference latency and energy consumption [14, 18]. Although recent work has shrunk the inference time steps down to 16 on small datasets with their accuracy at par with ANNs [7], this work still requires hundreds of time steps for large-scale datasets (e.g., ImageNet) [8, 10, 15]. It is essential to design an efficient SNN accelerator for competitive ANN datasets and tasks.

In addition, recent studies have shown temporal-encoded SNNs [8, 11, 15] (a.k.a., SNN-T) with their inherent higher sparsity and ability to encode temporal information in inputs can match the accuracy of an ANN, even on large-scale datasets [4, 16, 20]. However, the efficiency advantages of temporal-encoded SNNs will not be evident subject to the *chronological dependence.*

In this paper, we propose a novel SNN accelerator named SATO that decouples the *chronological dependence* during the neuron computation to overcome the bottleneck of computation over multiple time steps and leverage the full potential of temporal-encoded SNNs. We first design a temporal-parallel dataflow that simultaneously accumulates the input of both each neuron and each time step. After that, the accumulated results of a neuron in all time steps are fed into a novel search-adder tree module, which can immediately determine the time steps to fire output spikes. In addition, to balance the biased workload between processing elements (PEs), we propose an effective and efficient bucket-sorting method to dynamically group and dispatch the workloads evenly to PEs. Finally, we design a novel SNN accelerator architecture leveraging fine-grained parallelism. Our contributions can be listed as follows:

- We design a novel SATO architecture that can greatly improve the performance of SNN accelerator by synthesizing a temporal-parallel dataflow and a search-adder tree to determine the times of output spikes.
- We propose a bucket-sort based dispatcher that balances the workloads (i.e., the number of non-zero spikes) among PEs and maximizes the data locality of the input spike train that can be shared by PE groups.
- Experiments on various SNN-T show that SATO achieves average 30.9×, 22.1× and 6.4× speedup improvement upon ANN baseline "Eyeriss" and the state-of-the-art SNN accelerators "S2N2" and "SpinalFlow" with smaller area cost. Our design also remains the best among these designs in terms of energy efficiency.

2 BACKGROUND AND MOTIVATION

2.1 Spiking Neural Network

Fig. 1 depicts an SNN composed of a post-synaptic neuron, which is driven by two pre-synaptic neurons. $u_i(t)$ is the model state variable (corresponding to the neuron membrane potential). As the post-synaptic neuron receives spikes from pre-synaptic neurons, the synaptic weight for that inputs are added to the potential.

For better understanding, we compare the neuron computation required for the non-spike neuron (in ANNs) and spiking neuron (in SNNs) with the input. The calculation of the input signal to a non-spike neuron is as follows:

$$\mathbf{I} = \sum_{i=0}^{N} w_i x_i \tag{1}$$

where x_i is the input signal and w_i is the corresponding weight. Here, we focus on the Integrate and Fire model (IF) neuron model [16], the calculation of the input signal to the spiking neuron is as follows:

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \sum_{i \in \{i | X_i(t)=1\}} w_i \tag{2}$$

where, $\mathbf{V}(t)$ is the neuron membrane potential, which represents the state variable of the model, $X_i(t)$ is the input spike train from the i-th pre-synaptic neuron, and w_i is the weight associated with these inputs. $X_i(t) = 1$ means that a spike arrives at time step t. After the neuron membrane potential $\mathbf{V}(t)$ reach the pre-defined threshold V_{th}, it fires the spike at time step t and then resets membrane back to the reset potential V_{reset}. After that, subsequent input spikes will accumulate on the membrane again. Since the input values in SNN are 1 or 0, the mathematical dot product operation in Eq. 1 is transformed into the addition operation in Eq. 2. The information transmitted in the SNN is based on a temporal-encoding [4, 16] - the neuron activation value is represented by the latency to the first spike of the corresponding spike train over a given time window.

2.2 SNN Accelerators

Existing SNN accelerators [9, 14, 18] have a similar dataflow: map the post-synaptic neurons calculations onto PEs in parallel and integrate spikes along the time steps in serial. Fig. 2(a) illustrates the aforementioned dataflow and pseudocode, wherein M represents the number of pre-synaptic neurons, N refers to the number of post-synaptic neurons, and T is the number of time steps. The accelerator processes the spikes as follows: ❶ integrate the spikes at time step t; ❷ accumulate the integrated spike to the membrane potentials; ❸ compare the current membrane potential to the prescribed firing threshold; ❹ fire the output spike at time step t and reset the membrane potential if the potential exceeds the threshed. The inner Neuron Loop is unrolled and mapped to PEs in parallel. All PEs execute one round of process ❶ ~ ❹ in serial. They continue the T rounds overall.

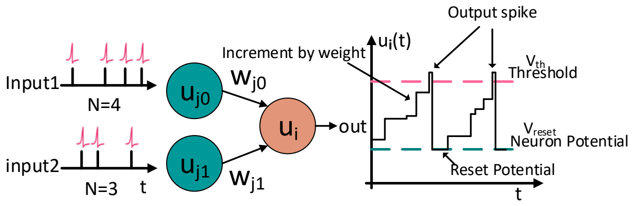


Figure 1: A basic spiking neuron with 2-input. Incoming spikes through the synapse (weight) impact the membrane potential.

In the conventional design, the accumulation of membrane potential depends on the accumulated membrane potential of previous time steps, making SNNs require sequential computation across multiple time steps of the spike train. The number of cycles required to perform an inference is at least the number of time steps. The latency of the accelerator significantly increases when T is large. The parallelism of PE is limited by the number of neurons (i.e., neuron-level parallelism). Conventionally, prior works [14, 16] such as SpinalFlow [14] exploit the sparsity of the spike train to speed up the SNN.

2.3 Motivation

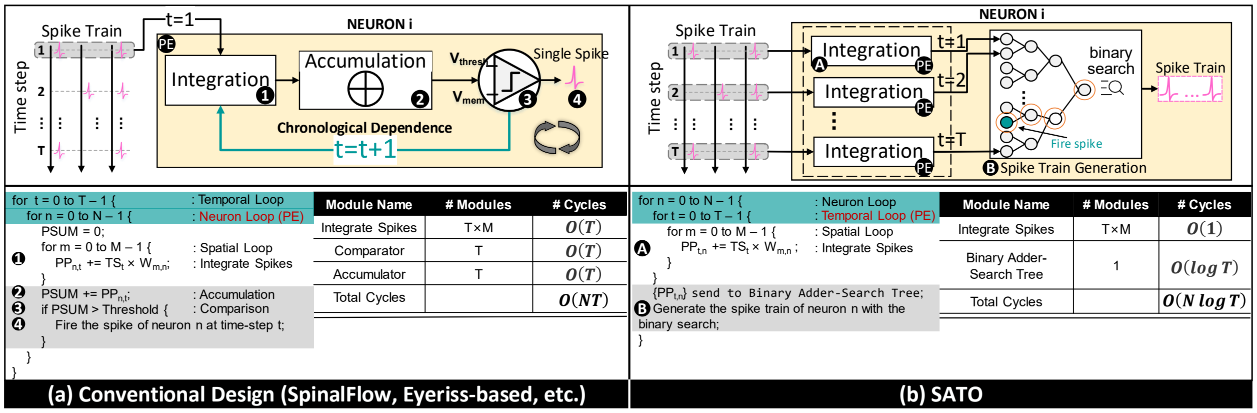
Existing SNN accelerators still suffer from low energy efficiency and long processing latency under the time-driven mechanism: (i) SNNs iterate over all the neurons for each time step (i.e., neuron computation) and accumulate membrane potentials of previous steps at each step, leading to additional memory requirements for storing intermediate potentials and expensive memory access costs in the total power budget. (ii) Accumulating spikes over multiple time steps leads to more operations, which reduces energy efficiency. (iii) A larger number of time steps represent the longer network latency. Thus, we propose a novel redesign of the SNN dataflow and architecture to decouple the chronological dependence and parallelize the integration of received spikes at each time step for boosting SNN efficiency.

3 SATO SCHEME

3.1 Overview

The overview of SATO in Fig. 2(b) is depicted as follow:

In Step Ⓐ, we map the integration of spikes of all time steps on PEs without accumulating membrane potential, which expands the neuron-level parallelism to additional temporal-level parallelism. *This method increases the scalability of the SNN implementations and allows the computational logic of PE to remain fairly simple but effective compared to the conventional schemes.* We will discuss this in Section 3.2.



```
for  t = 0 to T – 1 {                       : Temporal Loop
    for n = 0 to N – 1 {                    : Neuron Loop (PE)
        PSUM = 0;
        for m = 0 to M – 1 {                : Spatial Loop
❶           $PP_{n,t}$ += $TS_t \times W_{m,n}$;       : Integrate Spikes
        }
❷       PSUM += $PP_{n,t}$;                 : Accumulation
❸       if PSUM > Threshold {               : Comparison
❹           Fire the spike of neuron n at time-step t;
        }
    }
}
```

Module Name	# Modules	# Cycles
Integrate Spikes	T×M	$O(T)$
Comparator	T	$O(T)$
Accumulator	T	$O(T)$
Total Cycles		$O(NT)$

```
for n = 0 to N – 1 {                        : Neuron Loop
    for t = 0 to T – 1 {                    : Temporal Loop (PE)
        for m = 0 to M – 1 {                : Spatial Loop
Ⓐ           $PP_{t,n}$ += $TS_t \times W_{m,n}$ ;      : Integrate Spikes
        }
    }
    {$PP_{t,n}$} send to Binary Adder-Search Tree;
Ⓑ Generate the spike train of neuron n with the
binary search;
}
```

Module Name	# Modules	# Cycles
Integrate Spikes	T×M	$O(1)$
Binary Adder-Search Tree	1	$O(\log T)$
Total Cycles		$O(N \log T)$

Figure 2: Comparison of the dataflow in our SATO to it in the state-of-the-art SNN accelerators. (a) Conventional Design and (b) Our Design. The conventional SNN accelerators, as well as SpinalFlow, perform parallel computation on neurons, while SATO performs parallel computation on both integration of spikes at each time step and neurons.

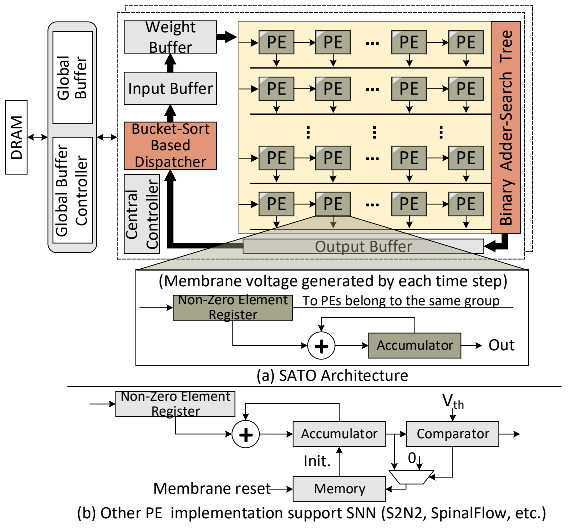

DRAM
Global Buffer
Global Buffer Controller
Weight Buffer
Input Buffer
Bucket-Sort Based Dispatcher
Central Controller
PE
Binary Adder-Search Tree
Output Buffer
(Membrane voltage generated by each time step)
Non-Zero Element Register
To PEs belong to the same group
Accumulator
Out
(a) SATO Architecture
Non-Zero Element Register
Vth
Accumulator
Comparator
Init.
0
Membrane reset
Memory
(b) Other PE implementation support SNN (S2N2, SpinalFlow, etc.)

Figure 3: Overview of SATO architecture and SATO PE details.

In Step Ⓑ, to perform spike train generation in a PE array, we orderly feed the integration results located at each time step to the adder tree and combine them with the binary search to determine the time step that the first fired spike. *This can be implemented by a novel binary adder-search tree module.* We will discuss this in Section 3.4.

Since all PEs process received spikes of each time step in parallel, the performance is bounded by the slowest PE, which requires workload balance among PEs. We develop a workload dispatch strategy to achieve workload balance and leverage the locality of the received spikes. *This innovative workload dispatch strategy minimizes the maximal number of received spikes for better PE utilization.* We will discuss this in Section 3.3. The overall SATO architecture and the detail of PE are shown in Fig. 3(a). The PE in our design is much simpler than other designs in Fig. 3(b).

3.2 Proposed SATO DataFlow

Membrane Potential Integration of each time step is processed on PEs in parallel. As depicted in Fig. 2(b), our SATO changes the dataflow, expanding the parallelism of the system to the temporal level (i.e., temporal-level parallelism). In this way, we can process the integration of received spikes at each time step in parallel, followed by the spike train generation. Our proposed dataflow can exploit both temporal-parallelism and neuron-level parallelism, increasing the scalability of the accelerator, and thus can be optimized for different types of SNNs. Therefore, SATO can reduce the minimum latency according to the hardware condition, which will be discussed in Section 4.2.

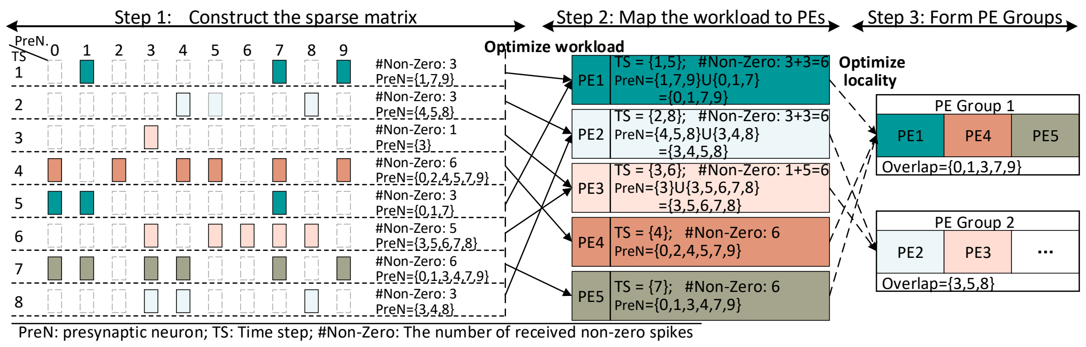
To optimize the performance of the system, the PE array acceleration has been adjusted in the loop nest of the existing SNN accelerators. We place T in an inner-loop position enabling the received spikes at each time step can be integrated by PEs in parallel, i.e., the PE parallelism is converted from M to T (loop interchange, the green area in Fig. 2). Therefore, the received spike decoupling chronological dependence makes the computing logic of PE in our dataflow fairly simple but effective, which performs the integration operation of received spikes in parallel.

Spike Train Generation is packaged with the PE array. To optimize the spike train generation efficiency, membrane potential accumulation has been peeled and placed in the outer-loop position (loop peeling, the gray area in Fig. 2). Once PEs complete the integration of received spikes of each time step, the results are fed to a novel binary adder-search tree in Section 3.4 to generate the spike train. In SNN-T, the spike train only has a single spike, which can be realized by adopting binary search with the time complexity $O(\log T)$.

3.3 Bucket-Sort Based Workload Dispatcher

PEs process spikes in parallel, so the performance of PEs is limited by the slowest PE, which requires balancing workloads among PEs. To reduce the delay of accessing the input spikes and simplify the hardware overhead, the allocation of spikes should consider the locality of non-zero spikes to further optimize the overhead of input spikes. Therefore, this work designs a workload dispatch strategy and exploits the inherent sparsity of the spike train and the data locality to optimize the workload and overhead of the PEs. As shown in Fig. 4, the three steps of our strategy consist of the construction of sparse matrix, PE mapping, and dispatch workload into PE groups.

Step 1: Construct the sparse spike matrix according to the spiking time of pre-synaptic neurons. Each row corresponds to a time step and displays the fired neuron at the time step. Each column corresponds to a pre-synaptic neuron and displays the firing state in each time step.



Step 1: Construct the sparse matrix
Step 2: Map the workload to PEs
Step 3: Form PE Groups
Optimize workload
Optimize locality
PreN.
TS
#Non-Zero: 3 PreN={1,7,9}
#Non-Zero: 3 PreN={4,5,8}
#Non-Zero: 1 PreN={3}
#Non-Zero: 6 PreN={0,2,4,5,7,9}
#Non-Zero: 3 PreN={0,1,7}
#Non-Zero: 5 PreN={3,5,6,7,8}
#Non-Zero: 6 PreN={0,1,3,4,7,9}
#Non-Zero: 3 PreN={3,4,8}
PE1 TS = {1,5}; #Non-Zero: 3+3=6 PreN={1,7,9}U{0,1,7} ={0,1,7,9}
PE2 TS = {2,8}; #Non-Zero: 3+3=6 PreN={4,5,8}U{3,4,8} ={3,4,5,8}
PE3 TS = {3,6}; #Non-Zero: 1+5=6 PreN={3}U{3,5,6,7,8} ={3,5,6,7,8}
PE4 TS = {4}; #Non-Zero: 6 PreN={0,2,4,5,7,9}
PE5 TS = {7}; #Non-Zero: 6 PreN={0,1,3,4,7,9}
PE Group 1
PE1 PE4 PE5
Overlap={0,1,3,7,9}
PE Group 2
PE2 PE3 ...
Overlap={3,5,8}
PreN: presynaptic neuron; TS: Time step; #Non-Zero: The number of received non-zero spikes

Figure 4: The flow of our workload dispatch strategy, which is composed of three steps: the construction of sparse matrix (Step 1), PE mapping (Step 2), and dispatch workload into PE groups (Step 3).

Step 2: Map the workloads to PEs. PEs in SATO process the potential accumulation for each time step, which means a PE may process one or more rows of the sparse matrix generated in Step1. The workload balance means that the maximum number of spikes processed by PEs should be minimized. Because each time step generates inconsistent sparsity, to achieve load balance, we have to ensure that the number of non-zero spikes in the time step processed by the PE is almost the same. We define a metric, Δnnz, representing the average number of non-zero spikes processed by each PE, i.e., $\Delta nnz = \frac{\#nnz}{\#PE}$, determines the optimal number of non-zero spikes processed by each PE. The workload of PEs is perfectly balanced when each PE processes Δnnz spikes. On this basis, we also need to take the data locality into account in our designed strategy. When a PE needs to process multiple rows of the sparse matrix generated in Step 1, our strategy determines the best choice of workload dispatched by PE according to the following two principles. First, we ensure that the number of spikes processed by the PE is close to Δnnz; Second, each PE processes as many spikes as possible repeatedly, by which the overhead of its load data is as small as possible.

Step 3: Dispatch workload into PE groups. To further optimize the overhead of PEs, we achieve data sharing by dispatching workloads into PE groups. The PEs in the same PE group take the same data by sharing memory. Here, to maximize the data locality of spikes, it requires PEs to share more input spikes. In other words, it is desirable to hold PEs with a larger number of overlaps non-zero spikes $n_{overlap}$ in the same group. As shown in Fig. 4, each group contains 3 PEs, i.e., these 3 PEs share the loaded data. PE1, PE4, and PE5 have the most duplicated non-zero spikes, up to 5 ($n_{overlap}$ = 5), to be allocated into a group. PE2 and PE3 have the most repeated spikes ($n_{overlap}$ = 3), so they are given into one group. Here, we regard the PE group as the bucket. Thus, this process can be implemented by Bucket-Sort with the time complexity $O(\#\text{Group} \times \#PE)$, where #Group and $\#PE$ are constant.

This is distinct from the general scheme where the on-chip buffer for each neuron storing receives spikes within the time window $\text{T} = \#TS$. Specifically, the maximum on-chip buffer size is $O(\text{T} \times \text{M})$ for the Eyeriss-based scheme and $O(\text{T} \times nnz)$ for SpinalFlow.

3.4 Hardware Detail

Bucket-Sort Based Dispatcher (BSD) is the key to balance the workload among PEs and maximize the data locality. Based on the workload dispatch strategy discussed in Section III-B, which is actually composed of the population count (popcount), ADD and the logical AND instructions available in most modern processors [21], we propose a BSD design without adding much complexity to the inference process. As illustrated in Fig. 5, we first load the input by

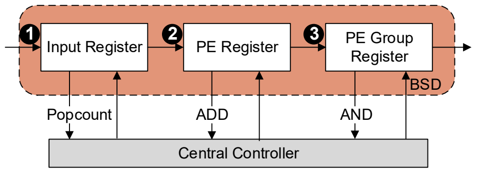
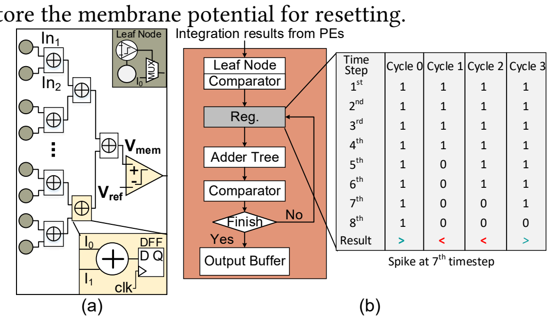


Figure 5: The bucket-sort based dispatcher of the SATO architecture.

rows into the input register and conduct the count for each row (i.e., counting the '1' at each time steps), which is a simple popcount of the row (denote as ❶). Then, we map the workload to PE based on the number of spikes processed by PE and save results into the PE register(❷). Next, we calculate the number of extra spikes when inserting PE into the group with logic AND operation and generate a group PE table stored in the register (❸). As a result, BSD assists in feeding the data to the PEs.

PE Array consists of PEs capable of integrating spikes at each time step. The output of each PE is applied to the binary adder-search tree to generate the spike train. Based on the dataflow we devised, the PE in the SATO architecture only needs to calculate the membrane potential increment at each time step. Such a PE design is simpler and more efficient than previous works [9, 14] illustrated in Fig. 3(b). The simplifications of PEs in SATO are manifested in two-fold: 1). Taking advantage of sparsity, it no longer needs to process a complete row at a time, so PE does not require a large scratchpad, which occupies half of the core area of the baseline PE, so this is an important saving; 2). Thanks to the designed temporal-oriented dataflow, we don't need the comparator and the memory to store the membrane potential for resetting.



Time Step	Cycle 0	Cycle 1	Cycle 2	Cycle 3
1st	1	1	1	1
2nd	1	1	1	1
3rd	1	1	1	1
4th	1	1	1	1
5th	1	0	1	1
6th	1	0	1	1
7th	1	0	0	1
8th	1	0	0	0
Result	>	<	<	>

Figure 6: The circuit sketch of binary adder-search tree (a); The binary adder-search tree of the proposed architecture (b).

Binary adder-search tree is responsible for processing the results from the PE array and generating the spike train. We can directly take the binary search for determining the time step that fired the spike for SNN-T. As shown in Fig. 6(b), once the outputs of PEs are fed to the leaf nodes, we first check whether the integrated result exceeds the threshold. If yes, we can shorten the required length of the accumulated time step. Then, we conduct the adder tree based on the register that assists in activating the leaf node to participate in the calculation. Next, we compare the final result with a predefined threshold, which actually generates the binary string. We find that the neuron fires the spike at 7-th time step and generates the spike train for SNN-T with three cycles.

4 EXPERIMENT

4.1 Experimental Methodology

Table 1: The characteristic of SNNs for verifying SATO.

Model	Structure	Coding	Timestep	DataSet	Acc.
SNN-T [4]	784-340-10	Temporal	800	MNIST	97.9%
STiDi-BP [12]	784-500-10	Temporal	600	MNIST	97.4%
SM+SR [15]	VGG-7	Temporal	544	CIFAR-10	91.05%
SSTDP [11]	VGG-7	Temporal	16	CIFAR-10	91.31%
TSC-SNN [8]	VGG-16	Temporal	2480	ImageNet	69.96%

Table 2: The power and area of components for SATO.

Components	Param	Spec	Power	Area (mm^2)
PEs	count	128	23.37 mW	0.0107
	bitwidth	8-bit		
	frequency	200 MHz		
Binary Adder-Search Tree				
Adder	bitwidth	8-bit	7.97 mW	0.0034
Register	bitwidth	16-bit	12.20 mW	0.0059
Comparator	bitwidth	16-bit	4.39 mW	0.0003
Total			**47.73 mW**	**0.0203**

Datasets and Networks. The benchmark networks evaluated in this work are based on popular image recognition datasets, such as MNIST, CIFAR-10, and ImageNet. We evaluate SATO with temporal encoded SNN. Tab. 1 lists the characteristics (structure, time step, accuracy, etc.) of each benchmark in various SNNs. To valid our SATO algorithm, we implement it in the Pytorch framework.

Modeling Accelerator Architecture. To evaluate the power and area of the components in SATO, we used 28 nm technology node. We model the behavior of the PEs in Verilog and synthesize it using Synopsys Design Compiler [19]. As for the global buffer, we use CACTI [13] to model it. For a fair comparison, we apply the same baseline (8-bit) as other SNN solutions, which have the same clock period (200MHz) and memory bandwidth. Computing Core (Core) consists of two parts, namely the PE array and the binary adder-search tree, which perform the function of the neuron model. The area and power breakdown of components in SATO are summarized in Tab. 2. The area and power of the Core in SATO are $0.0203mm^2$ and $47.73mW$, both smaller than SpinalFlow with the same amount. We set the number of PE groups mentioned in Section 3.3 as 8. To identify SNN efficiency, we use the Eyeriss architecture with row-stationary dataflow as an ANN baseline, which engages operates at the resolution consistent with SNN accelerators. For instance, 4-bit Eyeriss has the same input resolution as SNNs with 16 time steps.

4.2 Experimental Results

Energy Result: Fig. 7(a) shows the energy comparison of different SNNs deployed on various accelerators, which are all normalized to baseline SNN accelerator (i.e., Eyeriss design [3]). Compared to Eyeriss, S2N2 and SpinalFlow, SATO consumes 91.3% ,83.4% and

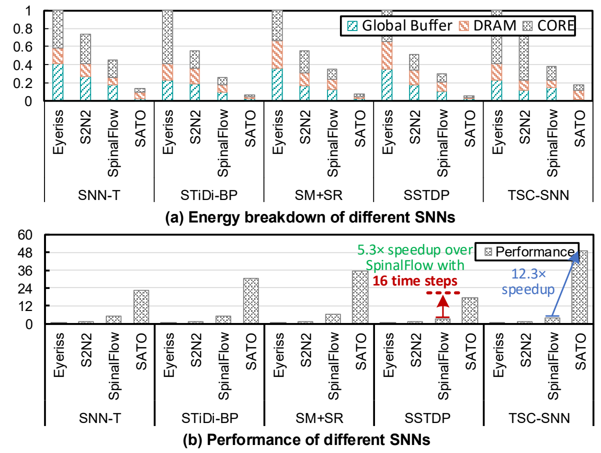


Figure 7: The Normalized energy breakdown (a); performance (b) of various SNNs.

69.7% less energy, respectively. The energy reduction is mainly due to the reduced overhead of PEs and the less non-sparse data transferred between DRAM and global buffers, which is supported by our temporal-oriented dataflow design using the sparsity of the spike train. SATO consumes less energy from the global buffer because SATO benefits from a workload dispatch strategy that allows good workload balance and exploits data locality, reducing the overhead of reading data from the global buffer. The reduction in the Core energy is mainly due to the reduced overhead of the Core we designed, consisting of the PE and binary adder-search tree reported in Tab. 2.

Performance Result: Fig. 7(b) shows the total execution cycles of various accelerators in different networks. Compared with Eyeriss, S2N2, and SpinalFlow, SATO achieves an average 30.9×, 22.1×, and 6.4× performance improvement due to the fact that the proposed SATO can exploit the sparsity and workload dispatch strategy. Specifically, the execution time of SpinalFlow is related to the number of time steps. Thus, for SNNs with too large time steps, such as TSC-SNN on the ImageNet, the benefits are not as good as other networks. In contrast, our workload dispatch strategy makes the number of non-zero spikes executed by our PEs calculations evenly apportioned. There is no need to maintain the alignment of the last time step and the maximum number of non-zero spikes as in SpinalFlow. Simultaneously, compared to SpinalFlow, SATO optimizes the dataflow to compute the aggregated potential at each time step in parallel.

Comparison Results: The comparisons among Eyeriss (ANN baseline), Eyeriss-SNN, S2N2, SpinalFlow and SATO are shown in Table 3. Eyeriss-SNN closely follows the Eyeriss architecture without the multiplier unit in PEs [14], adding index generation logic to exploit sparse spikes. These accelerators all explore the SNN implementation under the time-driven mechanism. However, only SATO considers decoupling the chronological dependence. SATO achieves achieves highest throughput (average 5.8× improvement compared to SpinalFlow with smaller area budget). The high throughput could be attributed to (i) The proposed mapping method can balance workloads among PEs, which avoid the performance of SATO is bounded by the slowest PE. (ii) Our SATO and optimization strategy can efficiently generate the spike train through the binary adder-search tree. The smaller area overhead is mainly because the

Table 3: COMPARISONS AMONG FOUR SNN ACCELERATORS and ONE ANN ACCELERATOR (BASELINE).

	Eyeriss [3]	Eyeriss-SNN [3, 14]	S2N2 [9]	SpinalFlow [14]	SATO
Technology	ASIC (28nm)	ASIC (28nm)	FPGA (28nm)	ASIC (28nm)	ASIC (28nm)
Frequency	200 MHz	200 MHz	200 MHz (scaled)	200 MHz	200 MHz
PEs	168	168	-	128	128
ALU per PE	8-b MAC	8-b Add, Cmp	8-b Add, Cmp	8-b Add, Cmp	8-b Add
Area (mm2)	1.068	0.808	-	2.09	1.13
Power (mW)	564.2	294.3	-	162.4	127.8
Throughput (GOP/s)	126.7 (1×)	56.6 (0.5×)	177.3 (1.4×)	684.5 (5.4×)	3,970.1 (31.3×)
Area Effi. (GOP/s/mm2)	118.7 (1×)	70 (0.6×)	-	327.5 (2.8×)	3,513.3 (29.5×)

budget of the PEs and register in our design is much simpler and smaller since SATO decouples temporal dependency in the neuron computation and achieves workload balance among PEs.

Impact of the number of PEs: Fig. 8 shows the change in energy per inference on TSC-SNN as the PEs increase from 128 to 1024. In the SpinalFlow design, PE is responsible for calculating per neuron. The number of time steps is generally much greater than 128 [16, 17], so the number of PEs is related to performance gains and energy consumption. As the number of PEs increases, the size of the corresponding weight buffer will increase. In SATO, each PE is responsible for the calculation of multiple time steps. As the number of PEs increases, we calculate more time steps in parallel to obtain more performance benefits.

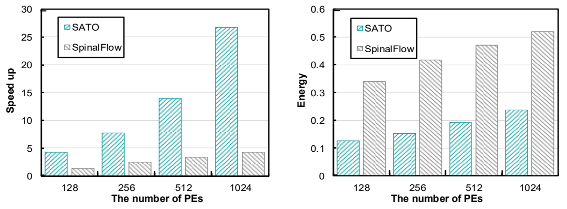


Figure 8: Speed up and energy per inference for TSC-SNN on SATO and SpinalFlow, normalized to Eyeriss, varies as the number of PEs.

Impact of time steps: Fig. 9 shows, for the PE array size of 128, the comparison of SNN network accuracy, performance (higher is better), and energy consumption (lower is better) varies as the number of time steps in SSTDP on the CIFAR-10, which are all normalized to SpinalFlow. As the number of time steps increases, the accuracy of the SNN increases, and the performance gains of SATO are also improving. The main reason behind this is that SATO can parallelize the calculation of each time step through PEs. Compared with the SNN with 4 time steps, SATO also achieves 2.9× speedup over the SpinalFlow. From Fig. 9(b), we find that with a larger number of time steps, SATO achieves a better energy efficiency. For the SNN with 16 time steps, compared to SpinaFlow, SATO consumes nearly 70% less energy. This is mainly due to the simplified PEs and workload dispatch strategy. Benefiting from the workload dispatch strategy, SATO handles fewer non-sparse spikes than the other schemes, leading to less energy consumed. Note that even if the time step of SNNs reduces as the algorithm evolves, SATO still gains notable performance and energy efficiency over the SpinalFlow. Meanwhile, SATO achieves better performance and energy efficiency when SNNs scale the number of time steps for greater accuracy.

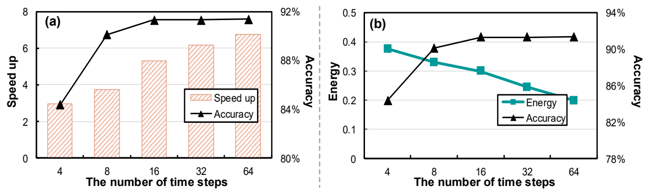


Figure 9: Analysis of the time steps.

5 CONCLUSION

In this work, we propose a temporal-oriented accelerator for spiking neural networks, namely SATO. It optimizes the calculation of time-step with well-designed dataflow and makes the system highly scalable. SATO, combined with the proposed workload allocation strategy, reduces the PE design overhead by exploiting sparsity and the traffic among memories using locality. Our evaluation shows that the proposed SATO scheme outperforms other similar schemes in performance and energy.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (NSFC) (Grant No. 61834006, 62102257 and U19B2035) and the National Key Research and Development Program of China (2018YFB1403400). We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University. Li Jiang is the corresponding author (ljiang_cs@sjtu.edu.cn).

REFERENCES

[1] Filipp Akopyan et al. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *TCAD* (2015).
[2] Ben Varkey Benjamin et al. 2014. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. of the IEEE* (2014).
[3] Yu-Hsin Chen et al. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for CNNs. *ACM SIGARCH Computer Architecture News* (2016).
[4] Iulia M Comsa et al. 2020. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP*. IEEE.
[5] Mike Davies et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro* (2018).
[6] Lei Deng et al. 2020. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks* (2020).
[7] Wei Fang et al. 2021. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *ICCV*.
[8] Bing Han and Kaushik Roy. 2021. Deep Spiking Neural Network: Energy Efficiency Through Time based Coding. In *ECCV*. 388–404.
[9] Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner. 2021. S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks. In *FPGA*. 194–205.
[10] Yang Li et al. 2021. BSNN: Towards Faster and Better Conversion of ANNs to SNNs with Bistable Neurons. *arXiv* (2021).
[11] Fangxin Liu et al. 2020. SSTDP: Supervised Spike Timing Dependent Plasticity for Efficient Spiking Neural Network Training. *Frontiers in Neuroscience* (2020).
[12] Maryam Mirsadeghi et al. 2021. STiDi-BP: Spike time displacement based error backpropagation in multilayer SNNs. *Neurocomputing* (2021).
[13] Naveen Muralimanohar et al. 2009. CACTI 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Tech. Rep* (2009).
[14] Surya Narayanan et al. 2020. SpinalFlow: an architecture and dataflow tailored for spiking neural networks. In *ISCA*. IEEE.
[15] Seongsik Park and Sungroh Yoon. 2021. Training Energy-Efficient Deep Spiking Neural Networks with Time-to-First-Spike Coding. *arXiv* (2021).
[16] Kaushik Roy et al. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature* (2019).
[17] Abhronil Sengupta et al. 2019. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in neuroscience* (2019).
[18] Sonali Singh et al. 2020. NEBULA: a neuromorphic spin-based ultra-low power architecture for SNNs and ANNs. In *ISCA*. IEEE.
[19] Synopsys. [Online]. https://www.synopsys.com/community/university-program/teaching-resources.html.
[20] Amirhossein Tavanaei and Anthony Maida. 2019. BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* (2019).
[21] Yaman Umuroglu et al. 2018. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *FPL*. IEEE.