TAIL: Exploiting Temporal Asynchronous Execution for Efficient Spiking Neural Networks with Inter-Layer Parallelism

Haomin Li[1,2,†], Fangxin Liu[1,2,†], Zongwu Wang[1,2], Dongxu Lyu[1], Shiyuan Huang[1,2], Ning Yang[1,2], Qi Sun[3], Zhuoran Song[1], Li Jiang[1,2]
1. Shanghai Jiao Tong University, 2. Shanghai Qi Zhi Institute, 3. Zhejiang University
haominli@sjtu.edu.cn, liufangxin@sjtu.edu.cn, ljiang_cs@sjtu.edu.cn

***Abstract*—Spiking neural networks (SNNs) are an alternative computational paradigm to artificial neural networks (ANNs) that have attracted attention due to their event-driven execution mechanisms, enabling extremely low energy consumption. However, the existing SNN execution model, based on software simulation or synchronized hardware circuitry, is incompatible with the event-driven nature, thus resulting in poor performance and energy efficiency. The challenge arises from the fact that neuron computations across multiple time steps result in increased latency and energy consumption. To overcome this bottleneck and leverage the full potential of SNNs, we propose TAIL, a pioneering temporal asynchronous execution mechanism for SNNs driven by a comprehensive analysis of SNN computations. Additionally, we propose an efficient dataflow design to support SNN inference, enabling concurrent computation of various time steps across multiple layers for optimal Processing Element (PE) utilization. Our evaluations show that TAIL greatly improves the performance of SNN inference, achieving a $6.94\times$ speedup and a $6.97\times$ increase in energy efficiency on current SNN computing platforms.**

***Index Terms*—Spiking Neural Networks, Brain-inspired Computing, Asynchronous Execution**

I. Introduction

Deep neural networks (DNNs) have witnessed explosive growth and widespread adoption in various domains, such as image recognition and autonomous driving [1]. However, as the demand for enhanced deep learning model capabilities and the exponential increase in training data persist, the size of deep learning models continues to expand. Consequently, the computational overhead required for inference tasks has also grown substantially, sometimes reaching up to 10^2 Giga Floating Point Operations (GFLOPs) [2], [3].

Spiking neural networks (SNNs) have emerged as energy-efficient alternatives to the conventional deep neural networks (DNNs), also known as artificial neural networks (ANNs) [4]. SNNs differ in that they operate based on spiking inputs, integrating temporal information through neuronal dynamics and employing binary spike signals (0-nothing or 1-spike event) for communication among neurons. SNNs achieve their energy efficiency by capitalizing on computational sparsity at each time step and replacing the multiply-accumulate (MAC) operations commonly found in ANNs with simple additions [5].

† These authors contributed equally. This work is supported by the National Natural Science Foundation of China (Grant No.62402311) and Natural Science Foundation of Shanghai (Grant No.24ZR1433700). Li Jiang is the corresponding author.

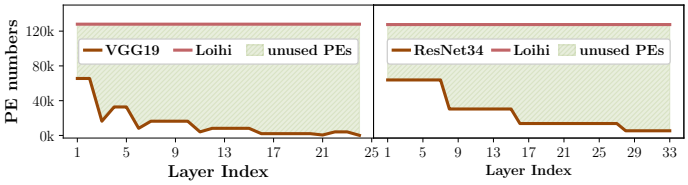

Fig. 1. The Gap between spiking accelerators and models. The red line indicates the number of PEs on chip and the brown line indicates the number of PEs required by each layer of the model. The evaluation is done on CIFAR-100 dataset.

In spite of the significant advances, existing SNN accelerators still suffer from low energy efficiency and long processing latency under the time-driven synchronous mechanism, where all neurons are updated at every time step [4], [6]. 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, ANNs infer in a single shot, whereas SNNs require computation over multiple timesteps, resulting in high inference latency [7]–[11]. Although the recent work has shrunk the inference time steps down to 16 on small datasets with their accuracy at par with ANNs [12], this work still requires hundreds of time steps for large-scale datasets (e.g., ImageNet) [5], [13]. It is essential to design an efficient SNN accelerator for competitive ANN datasets and tasks.

Existing solutions fail to efficiently address these challenges, primarily due to their reliance on software-based frameworks or hardware-based but time-driven synchronous mechanisms [7], [8], [14]–[16]. These approaches adopt a synchronous mechanism where the simulator iterates over all neurons for each time step, updating membrane potentials based on prior states and accumulated weights [17]–[19]. Consequently, only one layer generates sorted spikes as outputs at a time, which are then employed as sorted input spikes for the subsequent layer.

Therefore, it is nontrivial to fully utilize the two inherent parallelisms of SNN: intra-layer and inter-layer. Mainstream implementation uses layer-wise propagation that first calculates the result of one layer at all the time steps, and then sends them

978-3-9826741-0-0/DATE25/© 2025 EDAA

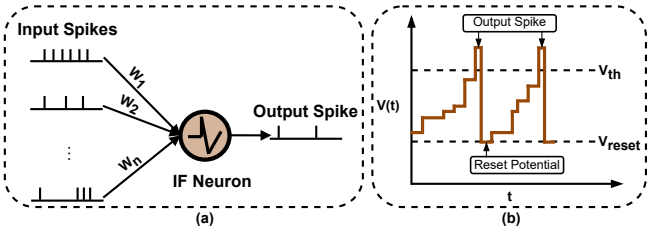

Fig. 2. Neuron dynamics in the SNN, illustrating the increase in neuron potential upon receiving input spikes and the generation of an output spike when the potential surpasses the threshold.

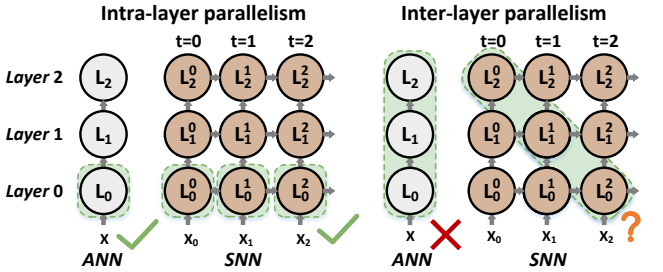

Fig. 3. Intra/Inter-layer Parallelism on ANNs and SNNs. ANNs only support intra-layer parallelism, whereas SNNs possess the potential to leverage inter-layer parallelism, in addition to supporting intra-layer parallelism.

to the next layer [14], [19]. This method utilizes the intra-layer parallelism but abandons the inter-layer parallelism. Figure 1 illustrates the under-utilization of Processing Elements (PEs) across different computing platforms. Notably, when deploying a VGG/ResNet model for CIFAR-100 on Loihi, over 70% of PEs remain unused during 70% of the inference time, indicating untapped execution efficiency potential in SNNs. Loihi 2 [14], [20], equipped with ten times the number of PEs compared to Loihi, presents an even more substantial pool of idle computational resources. Leveraging these idle PEs to support inter-layer parallelism has the potential to unlock further parallelism and significantly accelerate the SNN inference process.

To exploit the sparse nature and even-driven potential of SNN execution, in this paper, we present TAIL, a pioneering asynchronous execution mechanism for fast and efficient SNN inference. Our approach introduces a novel dataflow design, tailored to fully leverage idle PEs. This optimization allows for concurrent neuron computations across layers, unlocking the maximum benefits of asynchronous execution to achieve enhanced performance. In summary, this paper makes the following contributions:

- **Temporal Asynchronous Execution:** We introduce a pioneering time-step level asynchronous execution mechanism for SNNs, informed by a comprehensive analysis of SNN computation, substantially improving execution performance.
- **Efficient Dataflow Design:** We propose an efficient dataflow design, TAIL, to facilitate SNN inference. This design empowers concurrent computation of different time steps across multiple layers, ensuring effective utilization of PEs.
- **Performance Gains:** Through the implementation of TAIL on various SNNs, our evaluation demonstrates that a substantial improvement in performance, with a $6.94\times$ speedup and a $6.97\times$ energy saving on existing SNN computing platforms.

II. Backgrounds

A. Spiking Neural Network

One inference of SNN involves multiple time steps, each of which completes a forward calculation of the entire network. Spiking neurons are used as the basic computing units of SNN. Figure 2 shows a spiking neuron with n synaptic weights. $V(t)$ is the state variable of the neuron, corresponding to the neuron membrane voltage. V_{th} is the threshold voltage of the neuron and controls whether the neuron spikes [4].

For Integrate and Fire (IF) neuron model [4], in timestep t, each neuron accumulates the activated synaptic weights onto their membrane voltage based on input spikes x_is:

$$V(t) = V(t-1) + \Sigma_i w_i x_i \tag{1}$$

When the membrane voltage of a neuron exceeds the threshold voltage, the neuron will fire a spike:

$$output(t) = \begin{cases} 1 & if\ V(t) > V_{th} \\ 0 & otherwise \end{cases} \tag{2}$$

B. SNN Acceleration

In the realm of SNN acceleration, massive accelerators has emerged in recent studies, spanning digital, analog, and hybrid designs [18], [20]–[22]. Stanford's Neurogrid [21] takes a hybrid approach, introducing digital modules to augment parallelism and event routing. NEBULA [7] pioneers an analog implementation, leveraging Magnetic Tunnel Junction (MTJ)-based neuron and synapse models to mitigate latency concerns associated with multiple time steps. The IBM TrueNorth [22] processor adopts a digital stance, employing tiles to process all input spikes per time-step. However, challenges persist in terms of throughput, latency, and intra-tile parallelism. SpinalFlow [8] introduces a novel dataflow for temporal-encoded SNN, enabling parallel computation of multiple neurons through a PE array. S2N2 [15], an FPGA-based accelerator, champions a streaming SNN architecture with support for fixed-per-layer axonal and synaptic delays. The Eyeriss-based design [23] strategically minimizes data movement and hides time for membrane potential accumulation through an output stationary flow.

Despite the immense potential of SNN accelerators, their current inefficiencies pose challenges to achieving low-latency capabilities. This obstacle stems from the necessity of multiple forward passes for each spike train, a consequence of employing either software-based frameworks or hardware-based time-driven synchronous execution mechanisms for SNN deployment. This gap not only impacts energy efficiency but also hampers performance, especially in scenarios where extensive time steps are needed to achieve high accuracy.

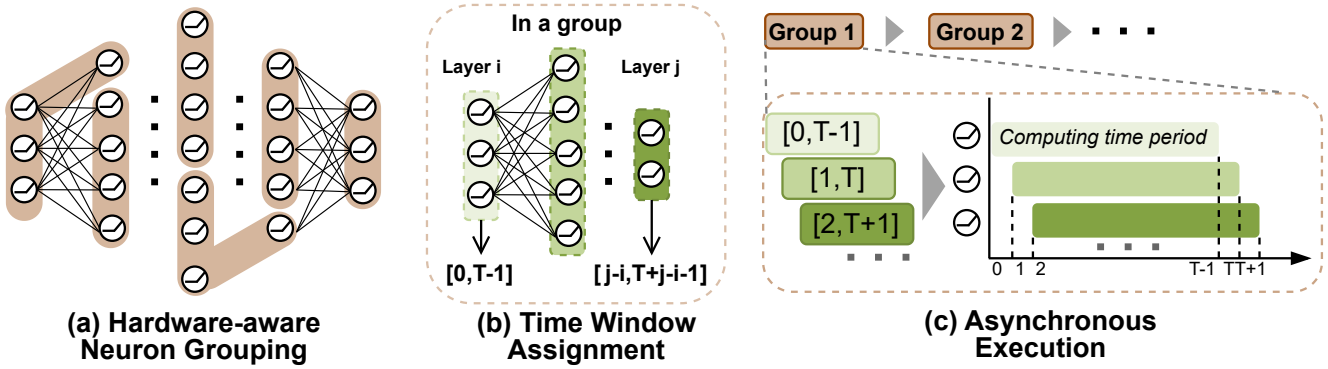

Fig. 4. TAIL Framework. (a) Hardware-Aware Neuron Grouping. The neurons are first grouped with awareness of the hardware configuration. (b) Time Window Assignment. The layers in a group are assigned with different time windows. (c) Asynchronous Execution. The layers in each group can execute asynchronously according to the time windows.

C. Motivation

In the event-driven computation, the SNN accelerator suffers from the resource contention during the scheduling of events when processing neuron computation and weight accumulation simultaneously. Implementing the asynchronous mechanism to update states significantly reduces the number of neuron computations. The weight accumulation, however, becomes a major source of performance bottleneck. Consequently, there is a pressing need to explore avenues for parallelizing both neuron computation and weight accumulation. As shown in Figure 3, SNN parallelism manifests in two forms: intra-layer parallelism, where output spikes align with input spikes, and inter-layer parallelism, involving the immediate transfer of output spikes as input to the next layer. While current synchronization implementations of SNNs predominantly leverage intra-layer parallelism, but often neglect the potential benefits of inter-layer parallelism. This oversight results in a diminished overall parallel processing capacity, which motivates us to design novel execution mechanisms to accelerate SNNs.

III. TAIL Framework

In this section, we present our efficient asynchronous execution framework, TAIL, designed to effectively harness both intra- and inter-layer adaptive opportunities in a hardware-friendly fashion. Our framework begins with the introduction of a novel resource-aware workload grouping strategy that strategically combines the advantages of intra- and inter-layer parallelism for adapting to the workload of different neurons within a SNN model. Subsequently, we present a general asynchronous mechanism capable of adapting to these neuron groups by selectively identifying neurons with suitable workloads across layers.

A. Overview

As depicted in Figure 4, our TAIL framework unfolds through three crucial phases: hardware-aware neuron grouping, time-window assignment, and asynchronous execution. During the neuron grouping stage, all spiking neurons within the model are partitioned into specific groups, each aligning with the maximum parallel computation capacity of the hardware resources. It's worth noting that each group may include neurons from different layers, emphasizing the adaptability of our approach. The subsequent assignment of time windows for neurons within each group aims to maximize intra-group computation parallelism while carefully addressing computational dependencies. Finally, the asynchronous execution of neurons within each group, coordinated at the time-step level based on the assigned windows, facilitates the inter-layer parallelism in SNNs.

B. Resource-Aware Neuron Grouping

In the context of our accelerator equipped with N processing elements (PEs), our objective is to optimize the full utilization of these PEs. To realize this, we conduct a thorough traversal of all neurons within the neural network, commencing from the last layer and progressing in reverse order. Throughout this traversal, we strategically partition each set of N neurons into distinct groups. Then, our TAIL imposes an efficient dataflow that allows asynchronous spiking operations of the PE. The working principle of the TAIL for global neurons across layers as follow: 1) When no data dependencies exist among the N neurons, we can optimizes the execution of these neuron workloads on massively parallel PEs; 2) In the presence of data dependencies, a subsequent time window assignment becomes imperative to ensure the necessary parallelism within the designated neuron group. This strategic assignment guarantees the effective utilization of the PEs within the accelerator, aligning with our goal of maximizing computational efficiency.

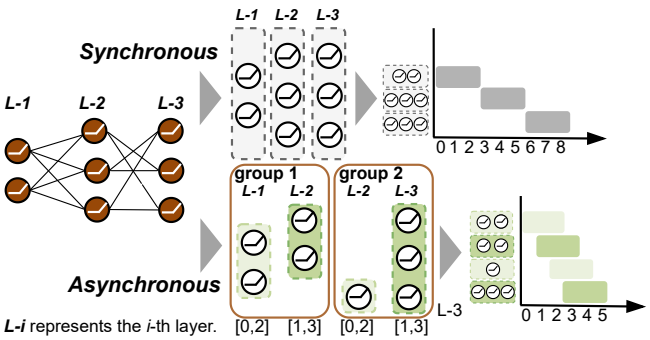

Fig. 5. Example of Asynchronous Execution.

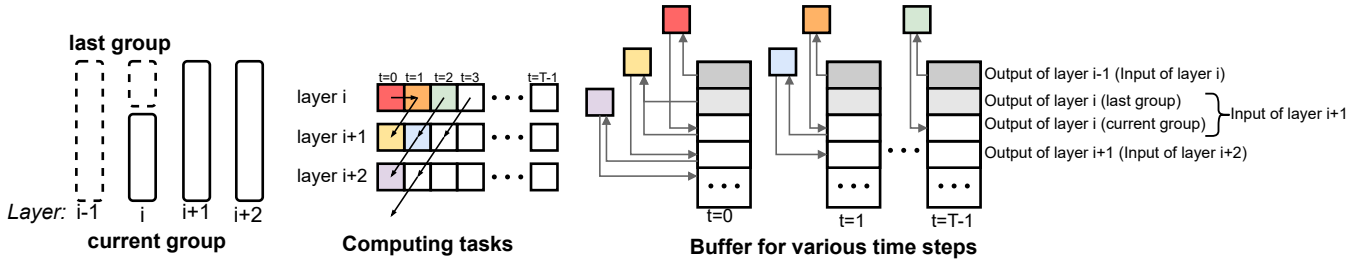

Fig. 6. TAIL Execution Flow and Data Flow.

C. Time Window Assignment

In our time window assignment, designed for groups of neurons spanning different layers, we ensure a seamless progression of consecutive layers through our grouping strategy. Let's break down the specifics: Suppose we have neurons extending from layer i to layer j in a network configured with a time step T. Each neuron is then assigned a unique time window:

- Neurons from layer i are allotted the timestep $[0, T-1]$.
- Neurons in layer j receive the timestep $[j-i, T+j-i-1]$.

Crucially, while the time windows of two neurons with direct computational dependencies don't precisely overlap, there exists a single moment within the window where no overlap occurs. This intentional misalignment fosters efficient asynchronous execution, contributing to the overall optimization of our computational process.

D. Asynchronous Execution

In the context of asynchronous execution, we treat neuron workloads in each group as an individual computational task, carrying out these tasks one after another. The key to asynchronous execution lies in the intricate domain of in-group computation, carefully guided by the assigned time windows for each neuron.

For a more detailed understanding, let's consider a scenario where neurons span from layer i to layer j within a group. The total computation time steps for this group are precisely determined as $T + j - i$. The essence of this process unfolds as follows: 1) Neurons originating from layer i undergo the computation of an output spiking sequence, characterized by a length of T. This computation happens within T time steps, ranging from time step 0 to time step $T-1$. 2) Conversely, for neurons from layer j, the computation of the spiking sequence occurs from time step $j-i$ to time step $T+j-i-1$. In such a way, TAIL ensures a seamless alignment between in-group computation and the designated time windows for each neuron, thereby optimizing the overall efficiency of our asynchronous execution paradigm.

Illustrative Example. Taking the example illustrated in Figure 5, where $T = 3$, the number of processing elements (PEs) is 4, and the MLP structure is 2-3-3, let's explore the application of the asynchronous execution mechanism to this three-layer MLP.

Initiating the process, we partition the neurons into two groups. The first group encompasses 2 neurons from the first layer and 2 neurons from the second layer, while the second group consists of 1 neuron from the second layer and 3 neurons from the third layer.

Subsequently, assigning time windows to neurons follows a specific pattern. For instance, the 2 neurons from the first layer in the first group are allocated the time window $[0, 2]$, whereas the two neurons from the second layer are assigned the time window $[1, 3]$. Similar time window assignments are made for neurons in the second group.

In actual asynchronous execution, the first group is allocated computation time from time step 0 to time step 3, while the second group is allocated time from time step 2 to time step 5. Considering the network structure, neurons in the first layer complete computation during time steps 0 to 2, neurons in the second layer complete computation during time steps 1 to 3 and 2 to 4, respectively, and neurons in the third layer complete computation during time steps 3 to 5. In synchronous execution, 9 time steps are required to finish the inference, while asynchronous execution only consumes 6 time steps. This highlights the efficiency of the asynchronous execution scheme, saving a notable 3 time steps compared to synchronous execution.

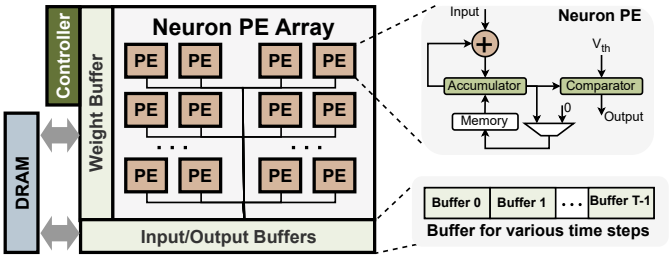

Fig. 7. TAIL Architecture.

E. Architecture Design

In this section, we introduce the architecture design to implement the TAIL algorithm, as illustrated in Figure 7. The key components of the TAIL architecture include neuron PE arrays, weight buffers, input/output buffers, and a controller.

We first describe how to augment the original MAC unit to support asynchronous execution, that can be easily embedded in existing SNN accelerators. The neuron PE contains an accumulator and a comparator, and reset circuit. The input/output buffers include T buffers, which are used to store the inputs and outputs of the neuron PE at each time step, respectively. the input/output buffer can be divided into T parts, each of

TABLE I
BENCHMARK CONFIGURATIONS.

Datasets/Models	Accuracy(%)			
	VGG 16	VGG 19	ResNet 18	ResNet 34
CIFAR 100	70.30	70.93	68.18	70.25%
ImageNet	68.47	69.96	69.37	69.93%

which is used to store the inputs and outputs of the neuron PE at each time step. Additionally, the input/output buffer can be divided into T parts, each of which is used to store the inputs and outputs of the neuron PE at each time step.

During the inference process, TAIL conducts computations in a group-by-group fashion. Initially, the weights corresponding to the neurons within the group are fetched and stored in the weight buffer. Simultaneously, a portion of the computation results from the preceding group is retained in the input/output buffer, while the remaining results are marked as invalid. At each time step, the input spike for the neuron PE is fetched, and the spike generated by the neuron PE is output to the buffer corresponding to that particular time step. This orchestration ensures a streamlined execution, aligning with the asynchronous computations in TAIL.

F. Execution Flow

In Figure 6, we provide a overview of the detailed execution flow within TAIL. Assuming the last execution of a group completes the computation of specific neurons in layer i, with the current group encompassing the remaining neurons in layer i and subsequent layers, a set of results (the spike train) is retained in the input/output buffers. The execution unfolds as follows:

Task Ordering: TAIL prioritizes computational tasks, with the computation for neurons in layer i at time step 0 taking precedence (highlighted in red).

Starting Computation: PEs fetch results from layer $i-1$ at time step 0 and store the output back into the buffer.

Asynchronous Execution: Subsequently, computations at time step 1 for neurons in layer i (highlighted in orange) and at time step 0 for neurons in layer $i+1$ (highlighted in yellow) occur simultaneously. The former fetches results from layer $i-1$ at time step 1, while the latter retrieves results from layer i at time step 0, encompassing both last and current group results.

Further computations take place concurrently, including neurons within layer i at time step 2 (highlighted in green), neurons in layer $i+1$ at time step 1 (highlighted in blue), and neurons from layer $i+2$ at time step 0. By employing buffers for various time steps, our architecture seamlessly aligns computation tasks with data pulling and storing. This design ensures simplicity, avoiding complex control overheads.

IV. EVALUATION

A. Experimental Setup

Networks, Datasets, and Baselines To validate our TAIL algorithm, we implement it in the SpikingJelly framework [24]. The benchmark networks evaluated in this work are based on

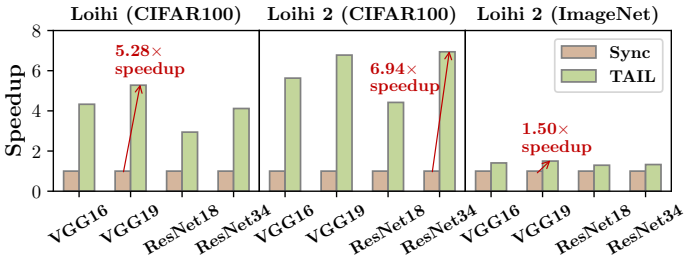

Fig. 8. Performance comparison between TAIL and synchronized execution.

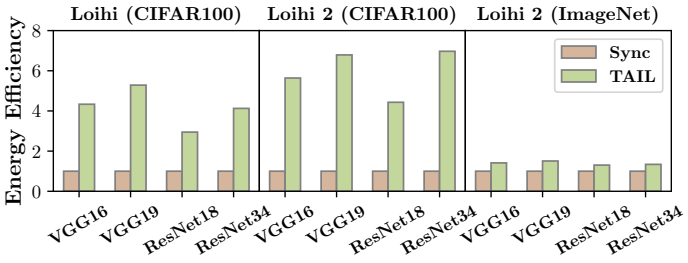

Fig. 9. Energy Consumption comparison between TAIL and synchronized execution.

popular image recognition datasets, including CIFAR-100 and ImageNet. Table I lists the accuracy of each benchmark in various SNN models, with timestep = 32. Recent effort results [2] show that they can be quantized to INT8 without accuracy loss, so we did not apply retraining for the quantization to INT8. However, TAIL itself is not precision limited and it can be easily extended to support INT16 and above.

Modeling Accelerator Architecture. To evaluate the power and area of the components in TAIL, we used 28 nm technology node. We model the behavior of the PEs in Verilog and synthesize it using Synopsys Design Compiler [25] at 100MHz. As for the global buffer, we use CACTI [26] to model it. For a fair comparison, we use the same PE implementation for baselines and TAIL, the only difference being the difference in the number of PEs. Specifically, for high resolution datasets like ImageNet, we use Loihi 2 for evaluation, while for lower resolution datasets like CIFAR-100, we perform evaluation on both Loihi and Loihi 2.

B. Experimental Results

Performance. We first compared the inference latency of our approach with an architecture based on a synchronized execution mechanism. The results, illustrated in Figure 8, highlight the significant advantages of our asynchronous execution mechanism across various models.

On the CIFAR-100 dataset using Loihi, even with a relatively small number of neuron PEs, our TAIL achieves an impressive speedup ratio of up to $5.28\times$ on the VGG-19 model. Moving to Loihi 2, equipped with one million neuron PEs, our TAIL reaches the highest speedup ratio of $6.94\times$. This remarkable result was achieved solely through modifying the execution mechanism. Even on the parameter-rich VGG-19 model, TAIL achieves a notable $6.78\times$ speedup.

When testing on the complex ImageNet dataset, a maximum speedup of $1.50\times$ is observed. The slightly lower speedup on

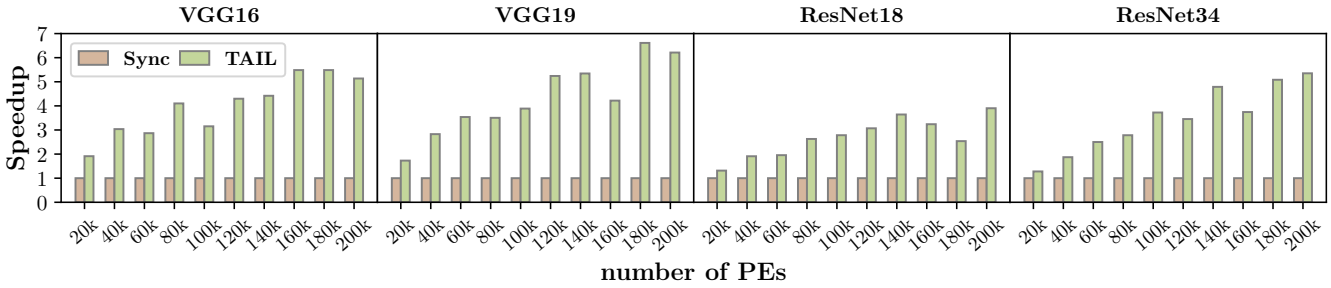

Fig. 10. Speedup of models on accelerators with different number of PEs.

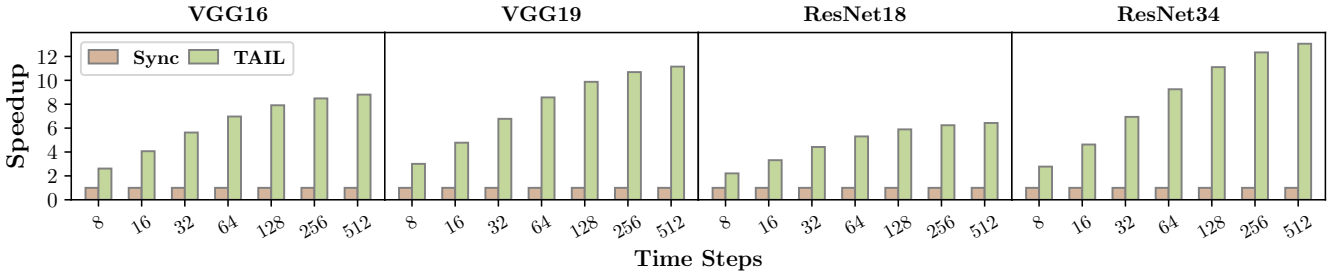

Fig. 11. Speedup of models on Loihi 2 with different time steps.

ImageNet is attributed to the high image resolution of ImageNet, resulting in a large number of neurons in a single layer, limiting the full utilization of inter-layer parallelism. However, we anticipate that technological advancements leading to an increased number of neuron PEs on the SNN accelerator will further enhance the asynchronous execution mechanism's advantage.

Energy. We further provide experimental results on energy consumption, illustrated in Figure 9. On Loihi 2, our TAIL achieves up to $6.97\times$ energy savings compared to the synchronized execution mechanism. Across all configurations, TAIL realizes an average of $3.84\times$ energy savings. This is mainly due to the fact that the asynchronous execution mechanism drastically reduces the time required for a single inference.

Notably, deeper network configurations exhibit higher speedups during deployment across models like the VGG and ResNet family. This observation highlights the asynchronous execution mechanism's potential for substantial speedup gains, highlighting its efficiency and scalability across diverse datasets and model architectures.

Impact of #PE. To explore the scalability of the TAIL, we conduct experiments varying the number of neuron PE on the SNN accelerators and observed its impact on the performance, as depicted in Figure 10. Focusing on the CIFAR-100 dataset, we systematically adjust the number of PEs from $20k$ to $200k$ with a stride of $20k$. From the plot, the speedup steadily increases with the growing number of PEs, reaching a maximum of $6.6\times$. This clear trend highlights the inherent acceleration capabilities of the TAIL. Notably, the VGG series outperforms the ResNet series in acceleration, likely due to the significantly larger number of parameters in the VGG models. Moreover, within the same model series, varying results among different models underscore the superior performance of TAIL execution mechanism in deeper networks. This observation emphasizes its effectiveness in handling the more complex network architectures.

Impact of Time Steps. We explore the effect of time step on the acceleration effect as shown in Figure 11. We sweep the time steps from 8 to 512 and perform inference latency comparisons across the four models. The speedup of TAIL becomes more obvious as the time step increases, reaching a maximum speedup of $13\times$ at a time step of 512. This enhancement stems from the ability of asynchronous execution mechanism to support inter-layer parallelism, effectively overlapping multiple rounds of computation latency within a layer. The crucial role of the time step lies in determining the rounds of computations for a single layer. As the time step increases, the distinction between single-layer computation and multi-layer asynchronous parallel computation becomes increasingly evident. Unlike certain prior approaches that rely on quantization or sparsity and often necessitate sacrificing accuracy, our method stands out. It particularly excels in scenarios requiring precise inference, where a higher number of time steps corresponds to elevated accuracy.

V. Conclusion

In this paper, we aim to construct an efficient execution mechanism that can reduce the latency caused by the large number of time steps of spiking neural networks. We analyze the computational process of SNNs and propose an asynchronous execution framework at the time-step level that can achieve inter-layer parallelism, based on the observation of the low PEs utilization of existing SNN accelerators. Moreover, we design the corresponding hardware architecture to fully utilize the advantages of the asynchronous execution mechanism and simplify the mapping of computation to data. Experiments show that TAIL can deliver significant performance gain without accuracy loss, so TAIL is promising to deploy and unleash the potential of SNN models.

REFERENCES

[1] S. Lu and A. Sengupta, "Exploring the connection between binary and spiking neural networks," *Frontiers in Neuroscience*, 2020.

[2] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, "Comprehensive snn compression using admm optimization and activity regularization," *IEEE transactions on neural networks and learning systems*, 2021.

[3] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1029–1042.

[4] K. Roy *et al.*, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, 2019.

[5] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *ECCV*, 2021, pp. 388–404.

[6] L. Deng *et al.*, "Rethinking the performance comparison between snns and anns," *Neural Networks*, 2020.

[7] S. Singh *et al.*, "Nebula: a neuromorphic spin-based ultra-low power architecture for snns and anns," in *ISCA*. IEEE, 2020.

[8] S. Narayanan *et al.*, "Spinalflow: an architecture and dataflow tailored for spiking neural networks," in *ISCA*. IEEE, 2020.

[9] F. Liu, H. Li, N. Yang, Z. Wang, T. Yang, and L. Jiang, "Teas: Exploiting spiking activity for temporal-wise adaptive spiking neural networks," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 842–847.

[10] F. Liu, Z. Wang, W. Zhao, N. Yang, Y. Chen, S. Huang, H. Li, T. Yang, S. Pei, X. Liang *et al.*, "Exploiting temporal-unrolled parallelism for energy-efficient snn acceleration," *IEEE Transactions on Parallel & Distributed Systems*, no. 01, pp. 1–16, 2024.

[11] H. Li, F. Liu, Z. Sun, Z. Wang, S. Huang, N. Yang, and L. Jiang, "Neuronquant: Accurate and efficient post-training quantization for spiking neural networks," in *2025 30th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[12] W. Fang *et al.*, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *ICCV*, 2021.

[13] S. Park and S. Yoon, "Training energy-efficient deep spiking neural networks with time-to-first-spike coding," *arXiv*, 2021.

[14] H. Lee *et al.*, "Neurosync: A scalable and accurate brain simulator using safe and efficient speculation," in *HPCA*, 2022.

[15] A. Khodamoradi, K. Denolf, and R. Kastner, "S2n2: A fpga accelerator for streaming spiking neural networks," in *FPGA*, 2021, pp. 194–205.

[16] Z. Wang, F. Liu, N. Yang, S. Huang, H. Li, and L. Jiang, "Compass: Sram-based computing-in-memory snn accelerator with adaptive spike speculation," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1090–1106.

[17] J.-J. Lee *et al.*, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in *HPCA*, 2022.

[18] Z. Li, B. Yan *et al.*, "Resipe: Reram-based single-spiking processing-in-memory engine," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[19] J. Li *et al.*, "Firefly: A high-throughput hardware accelerator for spiking neural networks with efficient dsp and memory optimization," *VLSI*, 2023.

[20] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, 2018.

[21] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. of the IEEE*, 2014.

[22] F. Akopyan *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *TCAD*, 2015.

[23] J.-J. Lee *et al.*, "Reconfigurable dataflow optimization for spatiotemporal snn on systolic array accelerators," in *ICCD*, 2020.

[24] W. Fang, *et al.*, "Spikingjelly," https://github.com/fangwei123456/spikingjelly, 2020.

[25] Synopsys, https://www.synopsys.com/community/university-program/teaching-resources.html, [Online].

[26] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, jun 2017.