DepAsync: An Asynchronous SNN Accelerator Based on Core-Dependency

Zhuo Chen, De Ma, Xiaofei Jin, Qinghui Xing, Ouwen Jin, Xin Du, Shuibing He, and Gang Pan, *Senior Member, IEEE*

***Abstract*—Spiking Neural Networks (SNNs) are widely used in brain-inspired computing and neuroscience research. Several many-core accelerators have been built to improve the running speed and energy efficiency of SNNs. However, current accelerators generally need explicit synchronization among all cores after each timestep of SNNs, which poses a challenge to overall efficiency. This paper proposes DepAsync, an asynchronous architecture that eliminates inter-core synchronization, facilitating fast and energy-efficient SNN inference with commendable scalability. The main idea is to exploit the dependency of neuromorphic cores predetermined at compile time. We design a DepAsync scheduler for each core to trace the running state of its dependencies and control the core to safely forward to the next timestep without waiting for other cores to complete their tasks. This approach prevents the necessity for global synchronization, allowing DepAsync to minimize core waiting time facing inherent core and time imbalance in SNN workloads. The comprehensive evaluations using five SNN workloads show that DepAsync achieves 2.47x speedup and 1.55x energy efficiency compared to the state-of-the-art synchronization architectures.**

***Index Terms*—Accelerators, spiking neural networks, neuromorphic computing.**

I. Introduction

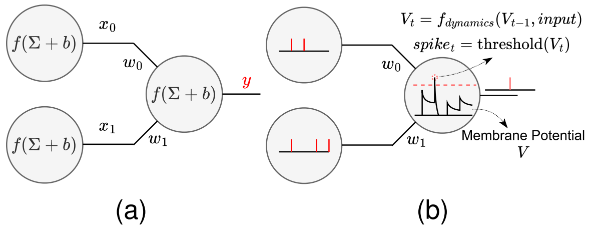
BRAIN-INSPIRED computing, or neuromorphic computing, aims to simulate brain behavior to achieve artificial intelligence energy-efficiently [1], [2], [3], [4]. Spiking neural networks (SNNs) are widely used models for neuromorphic computing and neuroscience research. Rather than conventional artificial neural networks (ANNs) use only rate coding, SNNs are more similar to biological neural networks, using various coding with richer spatial-temporal information. Neurons in SNNs are supposed to model simplified dynamics like biological neurons, mathematically equivalent to discrete-updated differential equations (Fig. 1).

Several neuromorphic accelerators have been proposed to exploit the great potential in low power consumption of SNNs.

Received 2 April 2025; revised 24 September 2025; accepted 19 October 2025. Date of publication 24 October 2025; date of current version 11 December 2025. This work was supported in part by the National Key R&D Program of China under Grant 2025YFG0100700, in part by the Natural Science Foundation of China under Grant 61925603 and Grant 62172361, and in part by the Major Projects of Zhejiang Province under Grant LD24F02001. Recommended for acceptance by Y. Hu. *(Corresponding author: Gang Pan.)*

The authors are with Zhejiang University, Hangzhou 310058, China (e-mail: gpan@zju.edu.cn).

Digital Object Identifier 10.1109/TC.2025.3625156



$f(\Sigma+b)$
x_0
w_0
x_1
w_1
y
(a)
$V_t = f_{dynamics}(V_{t-1}, input)$
$spike_t = \text{threshold}(V_t)$
Membrane Potential V
(b)

Fig. 1. Different neuron models between (a) ANN and (b) SNN.

They can be divided into two categories. The first kind of SNN accelerators [5], [6] adopts an architecture similar to existing ANN accelerators, which composes a 2-D systolic array of neuromorphic cores and a large shared global buffer. Several studies on optimizing datapath [5], SNN pruning [7] are dedicated to improving performance. However, the ANN-accelerator-like architecture separates computation and memory units, which is unsuitable for SNNs. On the other hand, a near-memory many-core architecture can dramatically eliminate redundant data movement, which is adopted by some industrial neuromorphic chips, such as TrueNorth [3], SpiNNaker [8], Loihi [1] and Darwin [9]. They comprise many neuromorphic cores with built-in memories for computation, and an on-chip network (NoC) for spike communication. There are many researches [10], [11] on improving performance, flexibility, and energy efficiency. In this work, we focus on this architecture.

Although recent many-core SNN accelerators can reduce memory access to lower power consumption, they suffer from low utilization caused by periodic all-core synchronization. The synchronization procedure is necessary for the SNN computation mode mentioned before, where each core in an accelerator updates neuron states step-by-step. Each neuromorphic core should wait for all other cores and spike transfers at each timestep to ensure the correctness of the result, wasting considerable time. Many solutions have been proposed to improve SNN accelerator performance by eliminating or relaxing synchronization. Unfortunately, they either use immature technologies [12], support a limited range of neuron dynamics [13], [10], or reduce the synchronization frequency by speculative execution mechanism meanwhile bringing rollback overheads [11].

In this work, we propose that DepAsync tackle the low utilization problem caused by global synchronization. We first

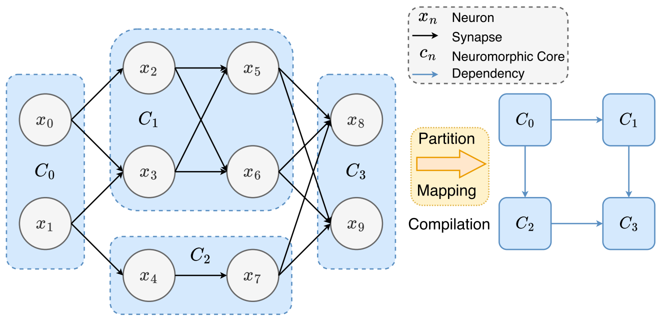


Fig. 2. Deploy pipeline on SNN accelerators.

explain synchronization from the perspective of data dependency among neuromorphic cores in SNN accelerators. Then, we introduce core dependency determined at compilation time (Fig. 2) as prior information into time-driven architectures and design a mechanism to trace running states of dependencies for each core dynamically. With static core dependency and dynamic running states, DepAsync can control cores to safely forward to the next timestep, reducing redundant waiting time. Our asynchronous mechanism eliminates all-core synchronization without speculative execution. Making fine-grained control in core-imbalance and time-imbalance workloads is more flexible, achieving higher speedup and energy efficiency. Our experiments use five different SNNs to evaluate DepAsync. The results show that DepAsync improves performance and energy cost by 2.47x and 1.55x over plain time-driven accelerators, and 1.87x, 1.40x over speculative-execution architectures.

The main contributions of this work are three-fold:

- We identify that the synchronization in time-driven SNN accelerators derives from data dependencies between neuromorphic cores and review time-driven architectures by analyzing core dependencies.
- We introduce core dependencies as prior information to time-driven SNN hardware accelerators and propose DepAsync to eliminate all-core synchronization to solve the low utilization problem.
- We evaluate our architecture with five different SNN workloads and demonstrate its advantages over existing accelerators.

II. Background

A. Time-Driven SNN Inference

Neurons in SNNs, acting as biological neurons, keep updating internal states (i.e., membrane potential) and output a spike when the voltage reaches the preset threshold as illustrated in Fig. 1, which gives SNNs the ability to model richer spatial-temporal information than ANNs. For example, the Leaky Integrate-and-Fire (LIF) model is prevalent nowadays in SNNs because of its good balance between biological plausibility and computing complexity. In mathematics, the internal state

Algorithm 1 SNN inference in the time-driven manner

```
Input: neurons $\{x_i\}_{i=1}^N$, synapse weight $\{w_{ij}\}_{N\times N}$
Input: threshold $v_{th}$, reset potential $v_{rst}$
Input: maximum timestep $t_{max}$
Output: spikes $\{s_{ij}\}_{N\times t_{max}}$
 1: while $t < t_{max}$ do
 2:     for all $x_i \in \{x_i\}_{i=1}^N$ do
 3:         $x_i.states = $ neuron_model$(x_i.states, x_i.acc)$
 4:         $s_{it} = x_i.states > v_{th}$
 5:         if $s_{it}$ then
 6:             $x_i.states = v_{rst}$
 7:             for all $x_j \in x_i.fanout$ do
 8:                 $x_j.acc = x_j.acc + w_{ij}$
 9:             end for
10:         end if
11:     end for
12:     $t = t + 1$
13: end while
```

updating procedure can be represented as a differential equation like Equation 1:

$$\tau_m \frac{dV}{dt} = -(V - V_{rst}) + \frac{I}{g_L} \tag{1}$$

where V is the membrane potential, I is the external input, and V_{rst}, τ_m, g_L are parameters, representing resetting potential, membrane time constant, and leak conductance, respectively.

However, it is complex and inefficient to calculate exact analytical solutions using digital circuits. Thus, to efficiently simulate an SNN on digital circuits, mainstream SNNs [14] use discrete methods to solve differential equations in Equation 1, where the discrete time dt is usually called *timestep*. Fig. 2 shows an example of SNNs, where x_i denotes neurons and the arrows represent connections between them (i.e., synapses) and the overall SNN inference can be written as Algorithm 1.

B. Many-Core Time-Driven SNN Accelerators

Compared with GPU-based acceleration [15], [16] and systolic-array-based SNN accelerators [5], [6], a custom-designed near-memory architecture can leverage more energy-efficient advantages of SNN as it eliminates intense data movement between global and local memory and therefore has been adopted by industrial neuromorphic chips [1], [3], [8], [9]. This architecture usually comprises many individual neuromorphic cores and an on-chip network, as illustrated in Fig. 3. The neuron unit in each neuromorphic core calculates neuron state updating step-by-step and the in-core memory stores neuron states and synapse weights. For simplicity, we still use time-driven accelerators to represent many-core architectures in the subsequent sections without specific clarification.

A compilation step is required to deploy SNNs to time-driven accelerators, as shown in Fig. 2. After compilation, the core dependencies are fixed. Notably, most prevalent forms of synaptic plasticity, such as STDP, focus on changing synapse weights

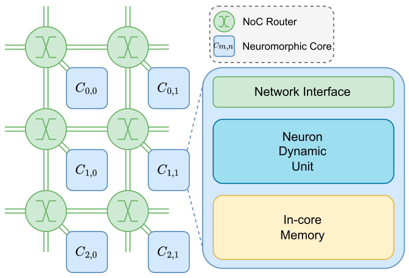


Fig. 3. Block diagram of many-core SNN accelerators.

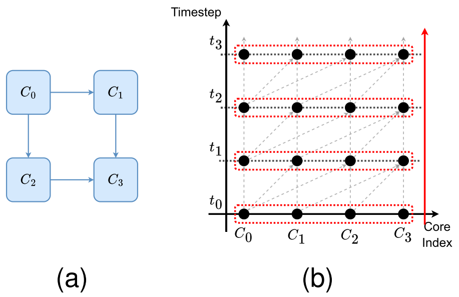


Fig. 4. SNN inference on time-driven SNN accelerators. (a) The topology of core dependencies. (b) SNN inference workflow.

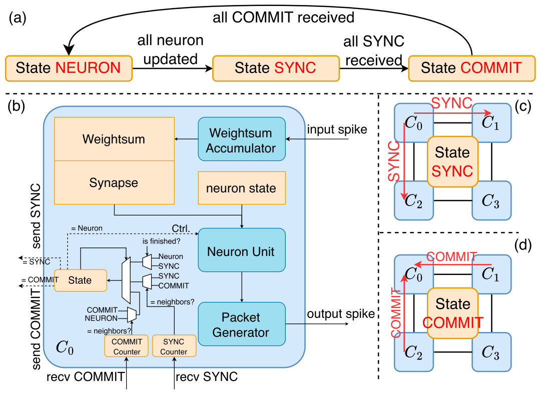


Fig. 5. Workflow of a timestep with adaptive global synchronization.

rather than adding or removing the connection themselves [17], so they do not affect core dependencies after compilation.

The primary contribution to accelerating SNN inference is exploiting inter-core parallelism within the single timestep. Neuromorphic cores can safely parallelize neuron internal state updating since there is no data dependency at this stage. Then, all cores go through a synchronization stage to wait for spikes generated in this timestep to arrive at destination cores via the NoC. Fig. 4(a) illustrates the topology of a typical SNN inference on a 4×4 time-driven accelerator with black arrows representing core dependency, and Fig. 4(b) shows the corresponding SNN inference workflow: cores perform neuromorphic computing (black dots) in parallel before synchronization (in a red rectangle), then work in a timestep-by-timestep manner. The whole system can only simulate SNNs step-by-step, restricted to per-timestep synchronization.

To implement a synchronization mechanism, TrueNorth uses a 1 kHz synchronization trigger signal [3]. When receiving this tick arrival, the neuromorphic core will be triggered to read the spike buffer and forward to the next timestep. SpiNNaker adopts a similar strategy, defining a timer event by a fixed number of clock cycles [8]. It also provides a catch-up mechanism to handle time imbalance in SNN workloads. Loihi [1] realizes synchronization more adaptively. After the neuron state updates, cores in Loihi will send special packets that will drill up spike packets remaining in NoCs. Darwin uses both synchronization methods to give users more options [9].

Fig. 5(a) illustrates a more detailed workflow in each timestep with adaptive global synchronization. In the NEURON state, the core performs neuron updates and weightsum accumulation in parallel. When a neuron firing, it will send a spike packet via the NoC; when a spike packet arriving, it will be consumed by the weightsum accumulator. When the core finished its computational task of a timestep, it turns to a SYNC state, and sends SYNC packets to its neighbors (Fig. 5(c)). If the core has finished and received SYNC from its neighbors, it turns to the COMMIT state and sends COMMIT packets back (Fig. 5(d)). These barrier packets will flush all in-flight spike packets in the NoC. When a core has received all COMMIT packets, all neuron computations and spike transmissions are finished. Then the core can forward to the next timestep. Fig. 5(b) shows the detailed global synchronization mechanism in each core.

C. Under-Utilization Problem

Although time-driven SNN accelerators dispatch neuron-updating computation to parallel neuromorphic cores, the sparsity nature of spiking data in SNN workloads is not adequately considered. Compared with dense multiply-and-accumulate operations in ANNs, highly sparse spikes imbalanced workloads among timesteps and cores, hindering overall hardware acceleration.

1) Imbalance in SNN Workloads: We perform an SNN inference with several workloads to demonstrate workload imbalance. The evaluation details are discussed in Section V-A1. The SNN runs 100 timesteps on a 4×4 accelerator. Fig. 6(a) shows that generated spikes exhibit significant variability across timesteps, and Fig. 6(b) renders 4×4 neuromorphic cores with different colors and numbers in each circle representing normalized firing rates of each core. Timestep imbalance is from temporal sparsity in SNNs, that is, neurons don't fire at all timesteps. On the other hand, the core imbalance is from spatial sparsity, which means not all neurons fire at each timestep. Thus, the total number of spikes is different.

2) Under-Utilization Caused by Synchronization: The under-utilization problem in time-driven accelerators has been observed in [11], [8]. It's difficult for a rigid synchronization mechanism to simulate imbalanced workloads. Fix synchronization signals cannot be changed dynamically, which is unsuitable for temporal-variant spike rates. Synchronization messages after the timestep finish make some cores always wait for the slowest core in the system, which cannot tackle inter-core imbalance. What makes it worse is that spike transfer time

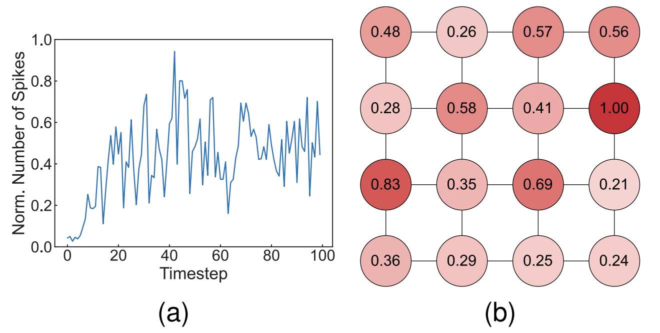


Fig. 6. (a) Time-imbalance and (b) Spatial-imbalance in SNN workloads.

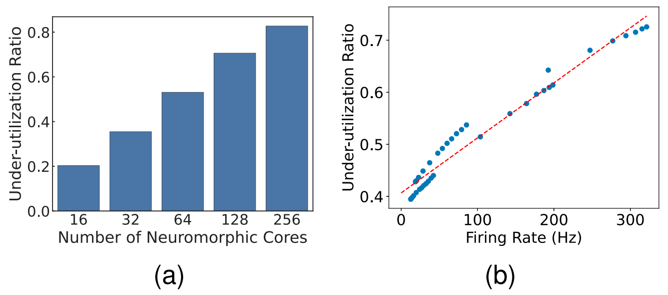


Fig. 7. Under-utilized cycle ratio of time-driven accelerators. The under-utilization increases with (a) scale and (b) firing rate.

in NoC grows as the number of cores scales up, and workloads with higher firing rates generate more spikes, which cause more NoC congestion. Both of these factors deteriorate the under-utilization problem.

Fig. 7(a) illustrates that the utilization ratio decreases with the growth of spike transfer time caused by increasing scales. Then, we adjust the average firing rate of SNN workloads. The results in Fig. 7(b) show that the under-utilization problem becomes worse as the firing rate increases, due to more generated spikes slowing down the transmission ability of the NoC.

D. Related Work

There are several approaches to reduce synchronization.

Firstly, leverage novel devices. Memristors are efficient emulators of biological neurons and synapses, because of the similarity in using ion migration as the fundamental mechanism. Analog circuits with memristors can directly simulate neuron dynamics [18], [19]. Thus, timesteps become unnecessary in such total neuromorphic chips. Recently, Chen et al. [12] used a passive electrochemical memory (ECRAM) array to improve programming accuracy for analog circuits. However, these emerging devices are still in the research stage and are less mature than large-scale integrated circuit technology.

Secondly, use spike event queues. Minitaur [13], PEASE [20] and NeuroEngine [10] use ordered event queues to manage spikes and get rid of global barrier-based synchronization. The event-driven neuromorphic core merges timestep-by-timestep computations and performs computations only at spike arrivals. This behavior skips some computational details in Equation 1, making it unsuitable for some mechanisms such as time-variant leak mechanism, dynamic synaptic conductance and second-order neuron models like the Izhikevich neuron model [21]. For the leak mechanism, Minitaur [13] fuses step-by-step leaks into one, and PEASE [20] simplifies the step-by-step leak by considering the leak at each timestep as a constant. Both lead to different results compared to the timestep-driven counterpart, where the leak at each timestep depends on the current neuron states. For second-order neuron models, an input spike can affect the next several timesteps when the input current (I in Equation 1) decays over timesteps. In this case, updating neurons only at spike arrivals may make neurons generate spikes from the past timesteps, leading to inconsistent results with Algorithm 1.

Thirdly, additional mechanisms are incorporated to mitigate synchronization. NeuroSync [11] proposes a speculative-execution (SE) and rollback-recovery (RR) mechanism to reduce synchronization frequency. However, periodical all-core synchronization is still inevitable, and the rollback-and-recovery overheads grow with the firing rate increases. Fig. 8(a) gives a 4-core example of the SE mechanism: cores run speculatively with rollback and recovery until the period synchronization. Fig. 8(b) shows RR cycles of time-driven accelerators with speculative execution, identifying RR overheads become gradually inevitable when more spikes are generated. Speculative execution mechanisms cannot fully leverage their advantages without the cooperation of predictors. SpikeNC [22] proposes a software agent-based asynchronous scheme. However, it does not leverage core dependencies and is limited at the SNN layer level. It also lacks hardware design. In this work, a novel time-accurate architecture is proposed to eliminate synchronization. We consider inter-cores dependencies as an additional control mechanism to achieve our design goals. Then, we realize DepAsync by adding a scheduler into each core and extending the spike communication mechanism via message passing.

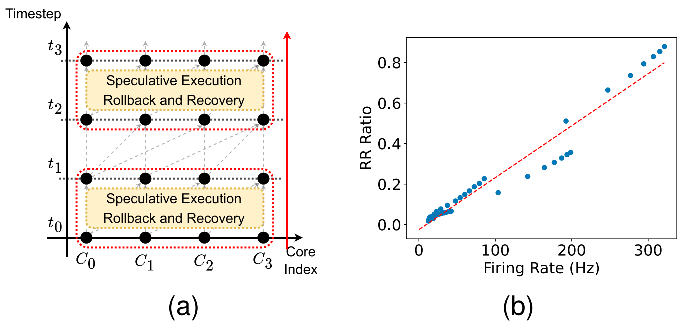


Fig. 8. Synchronous SNN accelerators with speculative execution. (a) SNN workflows with SE mechanism. (b) RR overheads.

The similar idea has been exploited for code generation with less barriers in compilers for distributed shared memory systems [23], task scheduling in real-time systems [24], etc. They assume that the execution time or computation cost of a computation task or operator is piror information, so that some algorithms such as

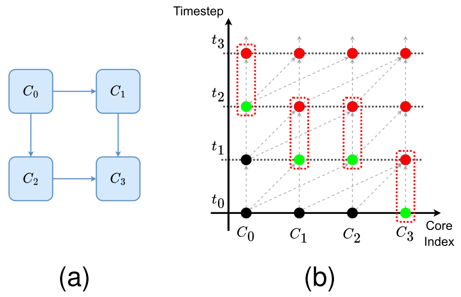


Fig. 9. Dependencies diagram of DepAsync. (a) Topology of neuromorphic cores. (b) SNN inference workflow on DepAsync.

DAG-based analysis or dynamic programming can be applied to obtain a static schedule. However, the execution time of each timestep in neuromorphic computing is dynamic. Thus, we need DepAsync to realize a dynamic schedule mechanism. This paper is the first to propose a dependency-aware, message passing mechanism to eliminate global synchronization in many-core neuromorphic architectures.

III. DepAsync Design

A. Ignored Inter-Core Dependencies

Line 6-9 in algorithm 1 demonstrates that synapses connect data dependencies between neurons. At timestep t, post-synapse neurons update their internal states, requiring spikes generated by pre-synapse neurons at timestep $t-1$. When we dispatch neurons to many-core SNN accelerators, dependencies between neurons are inherited by core dependencies. Similarly, we call core A a *pre-dependency* of core B if A sends spikes to B; versus the *post-dependency*.

Although these dependencies are Read-After-Writes (RAWs), synchronization becomes necessary in time-driven architectures for the correctness of results. In most SNN accelerators, synchronization is rigid, since it requires all cores to wait together, like a barrier primitive. However, each core is supposed to wait for only part of the cores on which it depends. When all pre-dependencies of a core finish timestep t and all spikes are received, it can safely forward to the next timestep without waiting for other unfinished cores.

Fig. 9 illustrates a 4-core example, where grey arrows denote data dependencies like the topology at the left. Meanwhile, black, green, and red dots represent core workloads at finished, running, and unfinished timesteps, respectively. At this moment, core 0 is running at timestep t_2, core 1-2 are at t_1, and core 3 is at t_0. For core 0, no pre-dependency means it can keep forwarding to the next, while cores 1 and 2 rely on spikes from core 0. Thus, they can only safely proceed t_2 after the current timestep finishes without waiting for the slowest core 3. Finally, core 3 only has a forward window $[t_0, t_1]$, since core 1 and 2 are its pre-dependencies.

On the other hand, post-dependencies limit the size of the *forward window* (red rectangles in Fig. 9(b)), which means how many timesteps a core can forward without waiting for its post-dependencies. In neuromorphic cores, the memory size assigned for buffering input spikes is finite, which means there may be no more empty buffer for spikes generated at far future timesteps. Here, we set the size of the forward window is 2 for visualization. For example, with core 1 and 2 constraints to buffering input spikes from 2 more timesteps, the forward window of core 0 will be $[t_2, t_3]$. In this case, if core 0 runs over t_3 fast enough that core 1 or 2 is still working on t_1, it has to wait for them to release some spike buffers when starting t_2.

B. Tracing Dependencies and Running States

The dependency analysis above gives neuromorphic cores the ability to eliminate whole-system synchronization. To implement such a mechanism, the main challenge is that each core has to trace dependencies and their running states.

Storing dependencies. For most SNNs, connections (not weights) between neurons are determined once the network is defined. When deployed to accelerators, SNNs are first partitioned into several groups, each corresponding to a logic neuromorphic core. Then, these logic cores are mapped to physical cores in real hardware by mapping algorithms [25], [26] designed for reducing spike transfer latency and on-chip network communication congestion. After mapping, fixed real core dependencies can be stored at in-core memories as other neuron parameters.

Tracing running states. Unlike static dependencies, running states dynamically change during the whole SNN inference, which is traced by NoCs in our proposed architecture. NoCs in most SNN accelerators are mainly used to transfer generated spikes. As the communication component of the system, they are also responsible for transferring meta information, such as input parameters [9], core statistics [8], and synchronization messages [1]. Naturally, we allow neuromorphic cores to transmit running states to their dependencies.

Generally, packets in NoCs compose packet headers, including meta information like packet type and routing information and packet bodies containing different message payloads. As shown in Fig. 10, we add a new kind of packet, DEP, for transmitting running states. The DEP packet body has three components. The *identifier* entry denotes the dependency the packet belongs to. The *timestep* entry records the running progress of dependencies. The *flag* entry is for distinguishing running states. In this work, *flag* is only 1 bit, with 0 for a timestep finish (FINISH packet), and 1 for a timestep start (START packet). Then, the core procedure in DepAsync is changed to Algorithm 2, where red lines highlight the differences.

In each neuromorphic core, two tables are prepared for running states of dependencies, which will be updated at the arrival of START/FINISH packets. Assume a core can accept spikes from m future timesteps, its safe forwarding conditions at timestep t is two-fold: 1) all its pre-dependencies finish t; 2) all its post-dependencies start $t-m+2$. If any condition is unsatisfied, the core turns to a wait state and keeps receiving and monitoring new packets. Once new DEP packets arrive and carry relevant information, the core is triggered to forward to the next timestep. When $m=1$, cores holding the output layer of the SNN, which have no post-dependencies, will first

header	*flit type* 2 bits	*VC* 2 bits	*dst port* 3 bits	*dst x* 4bits	*dst y* 4 bits	*src x* 4 bits	*src y* 4 bits

payload						
type 2 bits	*synapse id* 18 bits			*delay* 4 bits	*timestep* 9 bits	SPIKE packet
	reserved	*flag* 1 bit	*dep id* 8 bits	*delay* 4 bits	*timestep* 9 bits	DEP packet

Fig. 10. Packet structure in NoC. VC: out virtual channel; $dst\ port$: next hop destination; $dst/src\ x/y$: destination/source position; $synapse\ id$: activated synapse id in post-dependencies; $delay$: synapse delay; $timestep$: spike timestep; $flag$: START/FINISH flag; $dep\ id$: dependency id.

Algorithm 2 Neuromorphic core procedure in DepAsync

```
 1: while $t < t_{max}$ do
 2:     if not allowed to forward to $t$ then
 3:         waiting
 4:     end if
 5:     send START packets to its pre-dependencies
 6:     calculate weight sum, apply learning rules
 7:     for all $x_i$ in this core do
 8:         update $x_i.states$
 9:         if $x_i$ fires a spike then
10:             send spike packets to post-dependencies
11:         end if
12:     end for
        ~~waiting synchronization~~
13:     send FINISH packets to its post-dependencies
14:     $t = t + 1$
15: end while
```

forward to $t+1$, then wake up other cores in a cascading way. In this case, the whole system falls back to the synchronization mechanism. Otherwise, if $m > 1$, DepAsync can provide fine-grained control of neuromorphic cores with output correctness at every timestep.

Formally, the safe forwarding condition at Line 2 in Algorithm 2 is calculated by Equation 2:

$$\begin{aligned}\text{cond} &= \text{pre-cond} \wedge \text{post-cond} \\ &= \left(\bigwedge_{i=1}^{N_{pre}} t_i^{pre} >= t^{cur} \right) \wedge \left(\bigwedge_{i=1}^{N_{post}} t_i^{post} > t^{cur} - m + 1 \right)\end{aligned} \tag{2}$$

where N_{pre}, N_{post} are the number of pre- and post-dependencies, t_i^{pre}, t_i^{post} denote the timestep from received FINISH/START packets, and t^{cur} represents the current timestep of each neuromorphic core.

Fig. 11 gives an example of the DepAsync workflow. The topology is the same as Fig. 9, and the spike buffer can hold spikes from 2 timesteps (i.e., $m = 2$). Table headers *pre*, *post* and *cur* denote tables for pre-dependencies, post-dependencies, and the current timestep, respectively. The c_i within the parentheses at the beginning represents core dependency topology corresponding to Fig. 9. Solid arrows represent safe forwarding and dash arrows are DEP packets sent by neuromorphic cores.

At the start moment ❶, we also assume running states of cores align to Fig. 9 and timestep t_3 is the end. In detail, core c_0 has finished timestep t_2, c_1 is running at t_1, while c_2 is at the end of t_1, finally, c_3 is still working at t_0. In this case, the contents of the tables for pre- and post-dependencies are determined. c_0 is stopped after t_2 (filled in the red cell) because of the post-condition (illustrated as a red font), and c_2 is stuck at t_1 for the same reason. At moment ❷, c_3 safely forwards to its next timestep t_1. Since c_1 and c_2 are pre-dependencies of c_3, START packets at t_1 from c_3 are generated and sent to them. After a transmit latency in the NoC, c_1 and c_2 receive the packets and update their post-tables. In particular, the update in c_2 satisfies the post-condition, thus c_2 is wakened up to work on t_2, which also sends a START@2 packet to c_0. When it comes to moment ❸, c_3 surpasses the progress of c_1, which means it has to wait for pre-dependency c_1 to finish t_1 first. Then, at the moment ❹, c_1 accomplishes t_1 and successfully goes to the next timestep, therefore it sends a FINISH@1 to c_3 and a START@2 to c_0. Simultaneously, c_2 ends t_2 and waits c_3 again. The FINISH@1 from c_1 and FINISH@2 from c_2 arrive c_3 at time ❺, then replace the timesteps recorded in the pre-table and trigger c_3 to proceed to t_2. Then c_3 triggers c_2 after a while. Finally, at the moment ❻, c_3 goes to the last timestep, sending FINISH@2, which satisfies pre-cond of c_3. Overall, neuromorphic cores in DepAsync alternately proceed until the simulation end, and leverage dependencies to ensure correct results without all-core synchronization.

IV. DepAsync Implementation

In this section, we implement DepAsync by three critical components in neuromorphic cores. Fig. 12 gives an overview of our architecture. We first introduce the novel DepAsync Scheduler, which plays a core role in dependency analysis. Then, we design our spike buffers used for storing future spikes, making it possible to proceed neuromorphic core without whole-system waiting. Next, we enhance the packet generator to support new DEP packets that cooperate with the DepAsync scheduler. Finally, routers in NoCs are modified, ensuring correctness during the DEP packet transmission. It is worth noting that the DepAsync mechanism is suitable for a wide range of neuron and synapse models, and we adopt the computation (including neuron update and synapse accumulation) and firing control components similar to current accelerators [1], [5]. For example, the LIF model needs a multiplier to implement the leak mechanism, an adder to add on the accumulation that is accumulated after spike arrivals, and a comparator to detect whether the membrane voltage exceeds the threshold (i.e. firing a spike).

A. DepAsync Scheduler

The main component for core dependency is the DepAsync Scheduler with a working mechanism illustrated as Fig. 13. The scheduler is responsible for tracing dependency running states and controlling core forwarding.

Tracing dependency. The scheduler keeps monitoring income packets from the network interface. When DEP packets

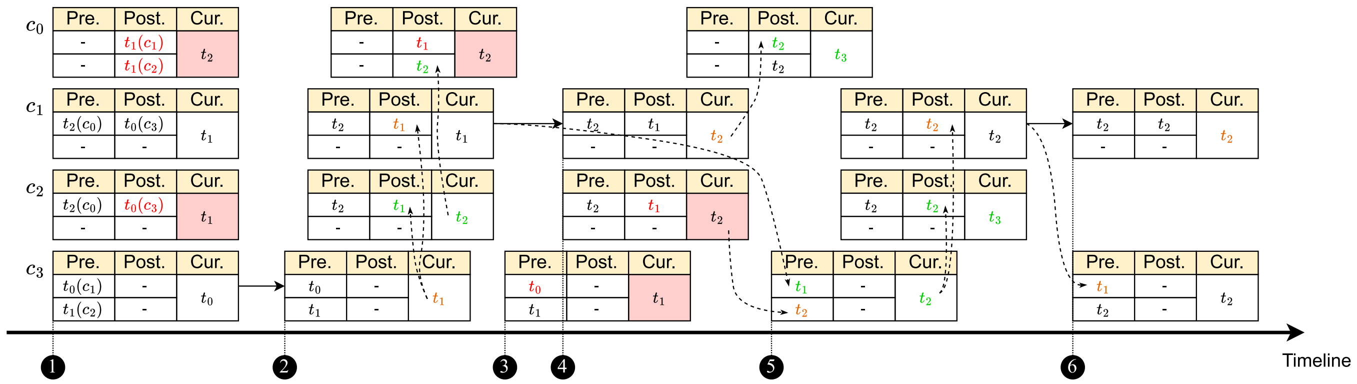


Fig. 11. A 4-core example of the DepAsync workflow with two spike buffers.

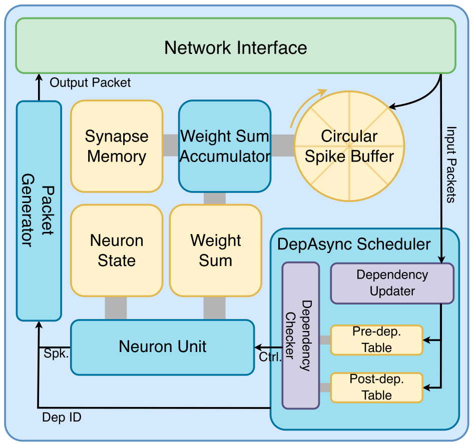


Fig. 12. Overview of DepAsync architecture.

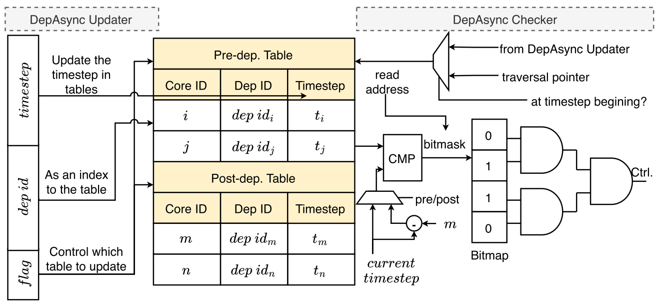


Fig. 13. Datapath in the DepAsync scheduler.

arrive, the DepAsync updater will store the carried *timestep* into the pre- or post-dependency table decided by the *flag* entry, with the address given by *dep id*.

Controlling. The DepAsync checker in the scheduler generates control signals to direct the neuromorphic core, whether forwarding to the next. Firstly, the timesteps stored in the pre-dependency table are compared with the current timestep t, checking whether the next timestep requires potential unarrived spikes. This condition guarantees the data dependencies. On the other hand, the memory assigned for buffering input spikes is limited, which means there may be no more empty buffer for spikes generated at far future timesteps. Hence, post-dependencies limit the size of the *forward window* (m). All timesteps stored in the post-dependency table are compared with $t - m$, ensuring all post-dependencies have enough space for buffering spikes. These two conditions can be formularized as Eq. 2. The DepAsync checker has a bitmap for all dependencies and a comparator to determine the conditions.

The scheduler is parallel to the neuron unit, and is activated when DEP packets arrive or a new timestep begins. The core receives one packet per cycle, therefore only one entry will be updated and checked at a time. When starting a new timestep, the bitmap is updated by iterating through the dependency tables. This traversal always completes before the new timestep ends, as the number of neurons within a core is generally much greater than the core dependencies.

B. Circular Spike Buffer

The spike buffer in DepAsync is circular, storing incoming spikes for subsequent processing. At the end of each timestep, buffers are rotated one slot. Circular spike buffers are generally adopted for supporting synapse delay [8], which means spikes received at t are used at $t + delay$. In this work, other slots store future spikes from pre-dependencies, which are logically the same as spikes with some delay. DepAsync works with synapse delay as well, subject to:

$$N_{slot} = max\_delay + m - 1 \quad (3)$$

Fig. 14 shows how to store a spike in the circular buffer. The *timestep* entry added by the *delay* determines which slot to access. As constrained by the scheduler, the slot destination never conflicts with the used slot. The occurrence of conflicts is due to excessive core forwarding, which causes the slots to loop back. However, in DepAsync, we track how many slots are available for dependencies through post-dependency tables, and if none are available, the core will pause and wait for new START packets. Thus, the destination slot in the circular spike buffer never conflicts.

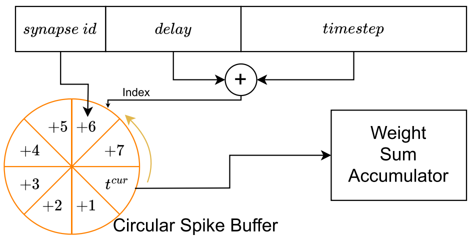
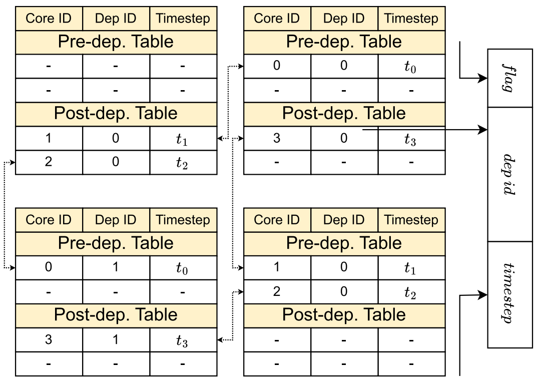


Fig. 14. Datapath of the circular spike buffer.



Core ID	Dep ID	Timestep
Pre-dep. Table		
-	-	-
-	-	-
Post-dep. Table		
1	0	t_1
2	0	t_2

Core ID	Dep ID	Timestep
Pre-dep. Table		
0	0	t_0
-	-	-
Post-dep. Table		
3	0	t_3
-	-	-

Core ID	Dep ID	Timestep
Pre-dep. Table		
0	1	t_0
-	-	-
Post-dep. Table		
3	1	t_3
-	-	-

Core ID	Dep ID	Timestep
Pre-dep. Table		
1	0	t_1
2	0	t_2
Post-dep. Table		
-	-	-
-	-	-

Fig. 15. Datapath for generating DEP packets.

C. Packet Generator

To work with dependency analysis performed by the scheduler, the packet generator is supposed to correctly send packets of DEP type.

As illustrated in Fig. 15, entry $flag$ and $timestep$ in the DEP packet are trivial, directly determined by the DEP packet type and the current timestep. On the other hand, we need an additional mechanism to generate the correct $dep\ id$. As mentioned before, $dep\ id$ is for addressing dependency tables. Thus, a neuromorphic core ought to be aware of its reversed index in pre- and post-dependencies, which is static and determined along with dependency relationships. In DepAsync, we store them at dependency tables. Fig. 15 gives an example corresponding to the topology in Fig. 9. The entry DEP ID stored in one core points to the index in its dependencies. Then the destination core of a DEP packet can decide which line in the dependency table to write.

D. Multi-VC NoC Router

The basic assumption in DepAsync is that START/FINISH packets precisely represent the start and finish of a timestep of neuromorphic cores. This implies that the START/FINISH packets must arrive at destination cores in order, and the FINISH packet at timestep t must be later than all spike packets. It is naturally satisfied if the on-chip network orderly transfers packets.

Recently, prevalent NoCs have adopted the virtual channel technique to improve network performance. Each router port has more than one physical channel and concurrently transmits several packets by multiple channels. Each input port in a router has an arbiter to grant a request from multiple virtual channels at each cycle in a round-robin strategy [27], which makes the packet transmission out of the original order and violates the assumption in DepAsync.

To cooperate DepAsync with virtual channels, the spike packets should be prior to the FINISH packets with the same source, destination, and timestep. Fortunately, routers with virtual channels generally mask out invalid channels before arbitrating one to transmit [27]. Thus, an additional mask is attached as shown in Fig. 16. In this case, FINISH packets always wait for spikes transfer first, and START packets are orderly transmitted as well. Noticing that the out-order of spikes within the same timestep is insignificant for correctness, DepAsync is capable of benefiting from performance improvement due to virtual channels.

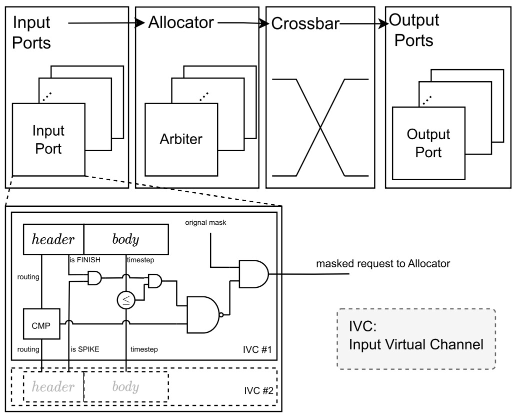


Fig. 16. Datapath of the additional mask in input virtual channels.

The NoC is responsible for both synchronization packets and spike packets. Upon receiving a SPIKE packet, the core stores it in the spike buffer, where it awaits processing by the weightsum accumulator. Conversely, DEP packets are handled directly by the DepAsync scheduler. From the perspective of the NoC, the only difference between these packets during transmission is their payload (Fig. 10). In a typical Network-on-Chip (NoC), the transmission of a packet involves several stages: route computation, virtual channel allocation, switch arbitration, and link traversal [27]. Throughout these steps, the NoC only utilizes information from the packet's header flit. For instance, it uses the src and dst entries (Fig. 10) for routing, and the VC entry to determine the channel for the next hop. During this process, the payload information is transparent to the NoC. Building on this foundation, we have extended a priority arbitration mechanism for two specific packet types: DEP and SPIKE.

Aside from adding a priority level to DEP packets, our NoC uses 2-cycle pipeline with lookahead XY routing as the router microarchitecture. And we use the flit-based flow control with virtual channels to mitigate head-of-line blocking and maximize bandwidth utilization. For buffers in the NoC, we choose the credit-based buffer backpressure mechanism. The NoC protocol in this paper adopts a classic NoC design [27] that is widely used in neuromorphic cores [1], [2], [8].

TABLE I
AREA (mm^2)

Component	Sync	DepAsync
Neuron	3.91	3.91
Synapse	13.36	13.36
Communication	3.22	4.08
Spike Buffer	0.91	1.82
Scheduler	-	0.31
Total	21.39	23.48

V. EVALUATION

A. Experimental Setup

1) Experimental Platform: To model performance, we develop a cycle-accurate simulator written in C++ to obtain workload latency. Neuromorphic cores execute instructions in Algorithm 2 cycle by cycle, and the topology of the NoC is 2D-mesh, and the routing algorithm is XY routing. This work primarily focuses on the synchronization mechanism within many-core architectures. It is important to note that DepAsync does not enhance the computational capacity (such as the maximum throughput); instead, it improves the utilization. Thus, each core in our simulator updates one neuron and sends/receives one packet per cycle for DepAsync and its baselines. The NoC component is implemented with reference to gem5 [28] in order to provide cycle-accurate packet communication. The logic in each core is implemented as pipelines to align as closely as possible with the hardware.

The energy and area model used in this work originates from NeuroSim [29]. We extend the NeuroSim to support our SNN workloads, where the energy cost of a single operation (such as neuron updates, buffer reads/writes, and NoC hops) is first synthesized and modeled, and then the overall energy consumption is estimated by multiplying it with the number of operations. The energy and area models of NeuroSim are calibrated with the Predictive Technology Model (PTM) [30], which is well-suited for early-stage design space exploration due to its public availability and broad technology node coverage, in contrast to industry transistor models. In this work, DepAsync is evaluated using 22nm PTM technology.

Each core of DepAsync can accommodate up to 4,096 neurons, 524,288 synapses, and 512 core dependencies. The limitation of the number of dependencies leads to a new constraint in compilation. Fortunately, 512 core dependencies are enough for our workloads (Sec V-C2). The size of the spike buffer is dependent on the parameter m. In our current design, if the buffer is full, we drop subsequent spikes. Considering that the spike buffer is much smaller than the neuron and synapse memories, we set a sufficient spike buffer size (2,048) to avoid dropping spikes. Thus, the drop never occurs in our evaluation. In the following experiments, m is 2 unless otherwise specified. Additionally, we use a 2-cycle router with 4 virtual channels for communication. For a 128-core setting with $m = 2$ and 4 virtual channels, the area of each component is listed in Table I. The area overhead caused by DepAsync is 9.8%.

TABLE II
WORKLOADS FOR EVALUATION. HERE 64C3 REFERS TO 64 CONVOLUTION KERNELS WITH KERNEL SIZE 3; AP2 REFERS TO AN AVERAGE POOLING WITH SIZE 2; AND FC REFERS TO A FULL CONNECTED LAYER

Workload (#Timestep)	**#Neuron**	**#Synapse**	**#Core**	**Structure**
MNIST (500)	7,298	566,864	16	Input-16C5-32C3-AP2-8C3-10FC
N-MNIST (500)	25,726	1,169,280	16	Input-16C5-AP2-32C3-AP2-10FC
DVS-Gesture (500)	130,923	33,917,568	128	Input-16C5-AP2-32C3-AP2-64C3-AP2-512FC-11FC
CIFAR10DVS (500)	174,218	51,254,400	128	Input-16C5-AP2-32C3-32C3-AP2-64C3-64C3-AP2-512FC-10FC
CIFAR10 (500)	753,664	632,332,288	1,024	Resnet18
Synthetic-1M (500)	1M	100M	256	-

2) Baselines: We compare DepAsync with two time-driven architectures, *Sync* and *SE*. The *Sync* is based on a global synchronization mechanism like Loihi [1]. On the other hand, the *SE* adopts the speculative-execution and rollback-and-recovery mechanisms in NeuroSync [11]. For fair comparisons, parameters such as core count, spike buffer size, and the number of virtual channels are the same across the three architectures. We also validate the output spike results of the two baselines to guarantee the SNN outputs are exact same as DepAsync.

We do not compare against time-driven accelerators with fixed synchronization signals such as TrueNorth [3] and SpiNNaker [8], because their synchronization frequency relies on user settings (such as 1 kHz). It is inconvenient to choose a suitable frequency for all SNN workloads which have different spike counts. The *Sync* architecture provides a more adaptive synchronization way to make sure all in-flight spikes are received.

3) Workloads: Firstly, we select five different real-data SNN workloads to evaluate DepAsync as shown in Table II: two small-scale workloads on MNIST [31] and N-MNIST [32] datasets, and two larger-scale worklods on DVS-Gesture [33], CIFAR10DVS [34], and CIFAR10 [35] datasets. For MNIST, dense images from the dataset are encoded by a Poisson process. For others, they contain real spike trains generated from digital images or dynamic vision sensor (DVS) cameras. We use these real-world datasets to train SNNs with a SNN framework Spikingjelly [16]. The accuracy of the models used in this paper is 98.7%, 91.8%, 91.3%, 67.7% and 80.4%, respectively. Besides, a larger synthetic SNN workload, consisting of 1 million neurons and 100 million synapses, is incorporated in our evaluation. All workloads use the LIF model and run 500 timesteps.

We convert different workloads to a common configuration format containing SNN structures and sample data. The SNNs are partitioned layer-by-layer where each layer takes several

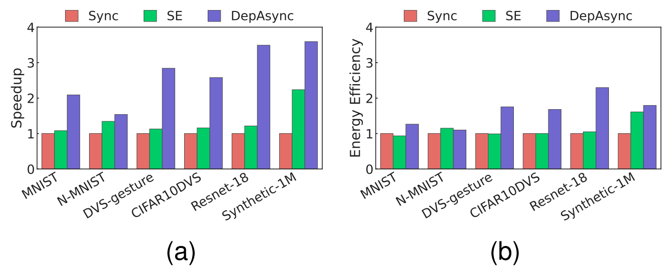


Fig. 17. Overall results. (a) Speedup. (b) Energy efficiency.

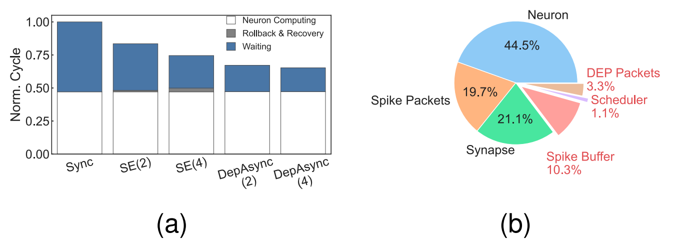


Fig. 18. (a) Latency breakdown and (b) energy breakdown of DepAsync.

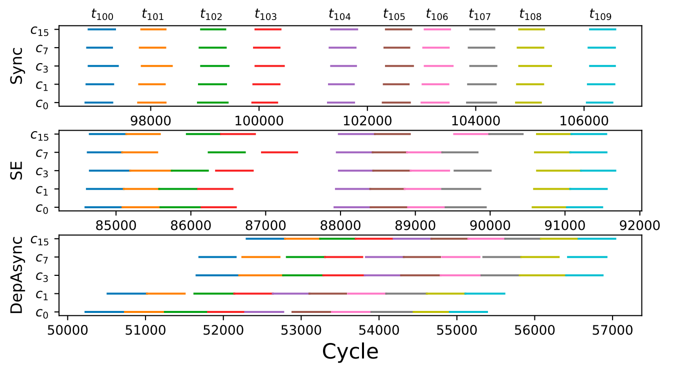


Fig. 19. A slice when three architectures running from timestep 100 to timestep 109.

logic cores. Then, we adopt an efficient mapping algorithm [25] to map high-level SNN logic cores to low-level hardware neuromorphic cores for future simulation.

B. Overall Results

Compared with the two baselines, we show the overall speedup and energy efficiency of DepAsync. Fig. 17 illustrates the improvement of SE and DepAsync over Sync on five workloads. In summary, DepAsync achieves 2.47x and 1.55x harmonic mean speedup and energy efficiency over plain synchronous time-driven accelerators, meanwhile 1.87x, and 1.40x over speculative-execution architectures. Exploiting more core parallelism than Sync speeds up DepAsync. DepAsync performs better than SE because it schedules cores by core dependencies, avoiding misspeculation and rollback overheads. The DepAsync approach demonstrates a more modest improvement in energy consumption compared to its speedup gains, primarily because DepAsync primarily optimizes waiting time rather than reducing actual neuron computations.

DepAsync leverages core dependencies to eliminate all-core synchronization. In the worst case, if a core with many post-dependencies is always the slowest one, these post-cores still wait at every timestep, formally equivalent to a many-core synchronization. However, this situation is quite unlikely to occur in a real data SNN workload.

Fig. 18(a) shows the latency breakdown of Sync, SE, and DepAsync. The waiting time in both SE and DepAsync decreases as the level of relaxation to synchronization increases. Although the idle time of SE cores is less than DepAsync, the cost of RR operations remains and even slightly grows due to the more aggressive speculative execution. For fair comparison between SE and DepAsync, the rest experiments set the synchronization period the same as the number of spike buffer slots m. Fig. 18(b) shows the DepAsync scheduler and additional spike buffers take about 11.4% of total energy consumption, and the DEP packets communication takes 3.3%. As for the memory cost, the additional spike buffers and dependency tables take 24 KB and 2 KB SRAM per core, occupying 4.3% and 0.4% more memory, respectively.

At circuit level, most many-core neuromorphic chips adopt a globally-asynchronous-locally-synchronous (GALS) technology [1], [2], [3], [8], [9]. The computational circuits use synchronous design and the clock signals are generated locally in each core. On the other hand, the NoC uses asynchronous design. With GALS, the global high-frequency clock is unnecessary, tremendously reducing the clock energy consumption. However, the asynchronous handshake protocol takes 30%-50% energy cost of the NoC part, because of its frequently signal flips.

Fig. 19 captures a slice in the MNIST workload, showing the difference between DepAsync and synchronous architectures, where colored lines represent neuron computation in a single timestep. Periodic synchronization can be observed from blanks lying in two workload lines. Meanwhile, DepAsync makes neuromorphic cores consecutively execute computations.

C. Sensitivity Analysis

We analyze the impact of spike buffer size, dependency density, firing rate, and NoC virtual channels on DepAsync. Furthermore, we also analyze the performance of DepAsync in the presence of cyclic connections and its sensitivity to different mapping algorithms in the supplementary materials.

1) Spike Buffer Size: Fig. 20 shows the impact of the spike buffer size on DepAsync performance. When the spike buffer size becomes large, more future timestep spikes can be stored, leading to a larger forwarding window and better performance. Different workloads have different performance converge speeds. For instance, DepAsync gains up to 3.79x speedup than baseline on DVS-gesture workloads. However, larger spike buffers lead to more energy consumptions. Thus, it is a trade-off in real applications. Across all workloads, the best choice of spike buffer size is 2 or 4, leading us to select $m = 2$ in the experiment settings.

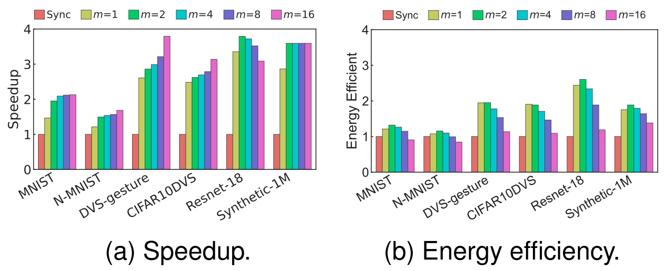


Fig. 20. The impact of the spike buffer size. The number in the legends denotes the number of spike buffer slots (i.e. m).

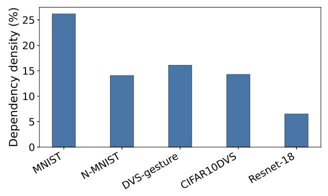


Fig. 21. Dependency density of real-data workloads.

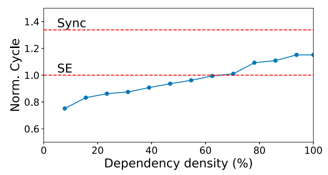


Fig. 22. Performance with dependency density.

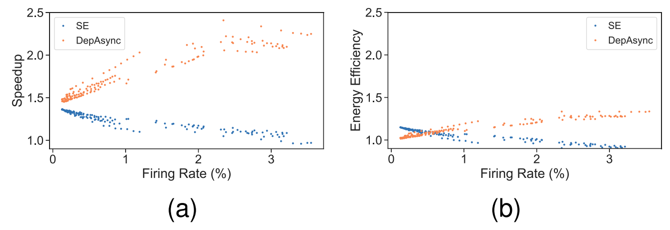


Fig. 23. The impact of the firing rate. (a) Speedup. (b) Energy efficiency.

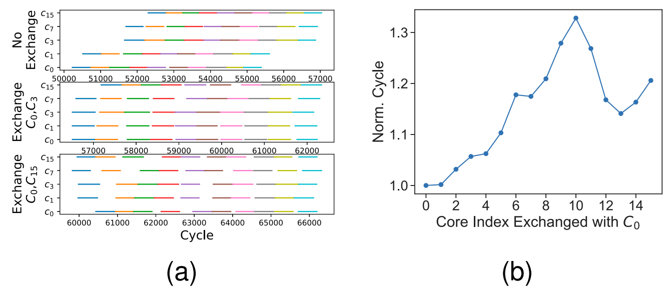


Fig. 24. The impact of cyclic connections. (a) a slice and (b) normalized execution time of DepAsync with different cyclic connections.

2) Dependency Density: The core dependencies in current workloads are not very dense because the partition algorithm aggregates neuron connections into neuromorphic cores. We manually add the density of core dependency (may not occur in real-data workloads) and evaluate DepAsync at different density levels. The results in Fig. 22 show that the performance of DepAsync decreased near linearly when core dependencies become denser and that DepAsync has advantages until the density reaches 70%, which is much higher than the density of normal workloads as shown in Fig. 21. These results suggest that a well-designed partitioning algorithm should aim to place connected neurons on the same core as much as possible to reduce core dependencies.

3) Firing Rate: Furthermore, we compare SE and DepAsync at different levels of firing rate. As mentioned, the more spikes are generated, the more likely rollback and recovery (RR) caused by misspeculations occur, which hints at overall performance. We change neuron parameters τ_m, V_{rst} in Equation 1 to control the average firing rate in SNN workloads. Fig. 23 demonstrates that the more spikes are generated, the more DepAsync exceeds SE, as the performance of SE is limited by the massive RR procedure at a high firing rate. It is worth noting that practical high-performance SNNs, either ANN-converted or trained by learning rules behave quite differently from real biological neural networks since the sparse coding and learning mechanism in the human brain is still an open problem. Thus, recent SNNs generally have a higher firing rate, in which case DepAsync provides faster SNN inference performance.

4) Cyclic Connections: DepAsync analyzes dependencies to control core behavior. When there is a cyclic connection, for example, two cores point at each other, both cores are pre-dependencies to one another. In this case, there are deadlocks when $m = 1$. However, when $m > 1$, the post-conditions are always satisfied, and the pre-conditions are broken once all cores finish the current timestep, which means the slowest core will trigger others at its end. Eventually, the whole system is fallback to a synchronous architecture. We exchange part of neurons between different neuromorphic cores and core C_0 to simulate the cyclic connections because C_0 holds neurons in the input layer and has no pre-dependencies in the default partition. As shown in Fig. 24(a), C_0 has to wait for other cores when there are cyclic connections. Fig. 24(b) shows that the performance decreases by 30% at worst with such cyclic connections. Fortunately, some cyclic connections can be reduced in software compilers with an additional restriction.

5) Mapping Algorithms: The mapping algorithm in SNN compilers is responsible for mapping the logic cores to real physical cores, which affects the distance between a core and its dependencies. Three mapping algorithms are involved in our experiments: Plain means logical cores are orderly mapped to hardware cores; HSC leverages the spatial locality of Hilbert space-filling curves; HSC+FD finetunes HSCs to exploit more locality [25]. We calculate the average physical distance (measured by NoC hops) between a core and its dependencies. The results are 2.37, 2.54, and 2.40 hops when using HSC+FD, HSC, and plain mapping algorithms, respectively. The difference in physical distance affects the overall performance.

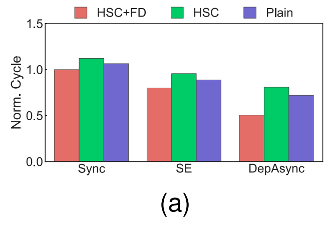
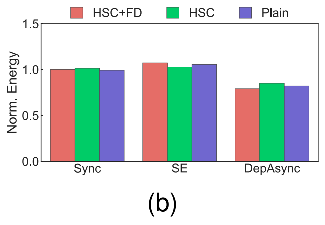



Fig. 25. The impact of mapping algorithms. (a) Latency. (b) Energy efficiency.

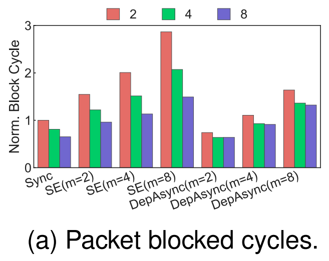
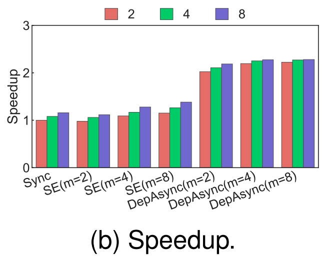



Fig. 26. The impact of NoC virtual channels.

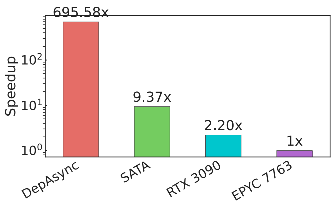


Fig. 27. Comparison with other hardware baselines.

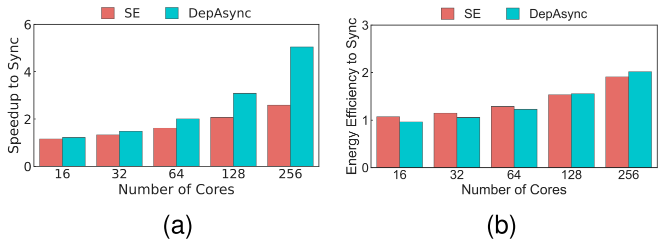


Fig. 28. Scalability. (a) Speedup. (b) Energy efficiency.

Fig. 25 shows the normalized cycle cost of Sync, SE, and DepAsync with three mapping algorithms. The results demonstrate that DepAsync performs worse than two baselines with poor mapping, which means it is more sensitive to the mapping result. Thus, a better algorithm is necessary for DepAsync. Nevertheless, DepAsync with poor mapping strategies still runs faster and costs less energy than the baseline synchronization architecture.

6) NoC Virtual Channels: Fig. 26(a) shows the average cycles of spike packets blocked in NoC routers. The block cycle decreases when there are more channels, meaning a larger NoC bandwidth exists. The block cycle grows when we increase m in both SE and DepAsync. The more timesteps we allow neuromorphic cores to process, the more spikes will be in-flight simultaneously. It is worth noting that the block cycle in DepAsync is smaller than that in Sync when $m = 2$. This is because additional packets in DepAsync($m = 2$) are from only one future timestep, thus are less than additional packets for global synchronization in Sync. Moreover, block occurs less in DepAsync than in SE with the same m. That is because when cores are allowed to process future timesteps, the NoC in DepAsync is only supposed to transfer future spikes, while the NoC in SE has to transfer packets generated by Rollback-and-Recovery operations as well. As for overall performance, Fig. 26(b) demonstrates speedup due to additional virtual channels, where DepAsync overwhelms Sync and SE in all settings.

D. Comparison to Other Hardware Baselines

The ANN-accelerator-like architectures differentiate from our many-core architectures in SNN inference manners. In DepAsync, layers in SNN networks are deployed to different neuromorphic cores and computed simultaneously, as mentioned in Section II-B. However, the ANN-accelerator-like architectures act as a layer-sequential mode where a layer will be processed after computing all timesteps of the previous layer, which is not suitable for SNN networks with no clear layer structure. There are several studies on ANN-accelerator-like architectures [5], [6]. Fig. 27 shows that DepAsync achieves 74.2x speedup than the state-of-the-art ANN-accelerator-like architecture SATA [6] which optimized SNN dataflow to reduce data movement energy overhead. In addition, the performance of DepAsync is significantly superior to that of CPUs and GPUs, with a ratio of 695.58x and 315.64x, respectively.

E. Scalability of DepAsync

Finally, we evaluate the scalability of DepAsync. To manually control the workload scale, a synthetic workload is generated like Izhikevich et al. [21], with both excitatory and inhibitory neurons involved. Synapses and connection weights are randomly set. We gradually increase the number of neurons and synapses in the synthetic workload from 16 cores to 256 cores, as shown in Table IV, then evaluate speedup and energy efficiency of DepAsync. The results shown in Fig. 28 identify the excellent scalability of DepAsync compared to Sync. As the system scales up, the synchronization cost becomes more expensive, thus DepAsync accelerates SNN inference faster. The energy efficiency grows as static power consumption decreases due to higher speed. The fact that speedup grows faster than energy efficiency is mainly because spike packets hop in the NoC grows as $O(\sqrt{N_{core}})$ while computation and memory units in neuromorphic cores grow as $O(N_{core})$. DepAsync gains 5.05x speedup and 2.01x energy efficiency at the 256-core scale, which is better than SE.

Moreover, we consider the bandwidth hierarchy in real-world many-core systems. Generally, the whole system consists of multiple chips, and each chip contains several neuromorphic cores. The intra-chip NoC is multiple times faster than inter-chip (about 4 times in Darwin [9]). To simulate the bandwidth hierarchy in a large-scale system, we divide 16×16 cores into

TABLE III
HARDWARE PARAMETERS OF DIFFERENT NEUROMORPHIC CHIPS

	TrueNorth [3]	SpiNNaker2 [36]	Loihi [1]	Loihi2 [37]	Darwin3 [2]
Implementation	Digital	Digital	Digital	Digital	Digital
Technology	28 nm	22 nm	14 nm	7 nm	22 nm
neuron models	LIF	Programmable	LIF	Programmable	Programmable
max. neurons per core	256	10^3	1K	8K	4K
max. synapses per core	64K	10^6	1M	1M	2M
Energy per synaptic operation	26 pJ	10 pJ	23.6 pJ	-	5.47 pJ

TABLE IV
SYNTHETIC WORKLOADS WITH DIFFERENT SCALES

#Cores	#Neurons	#Synapse
16 (4 × 4)	10,240	903,718
32 (8 × 4)	14,481	2,027,922
64 (8 × 8)	20,480	4,048,000
128 (16 × 8)	28,962	8,043,888
256 (16 × 16)	40,960	16,096,000

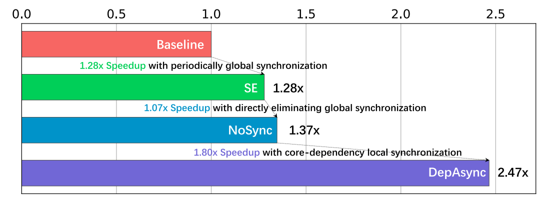


Fig. 30. The performance of different synchronization mechanisms.

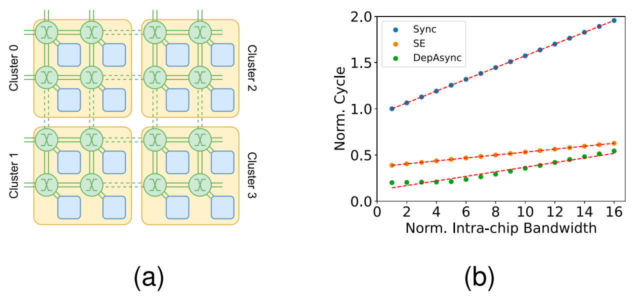


Fig. 29. (a) Topology and (b) Performance of large scale system with bandwidth hierarchy.

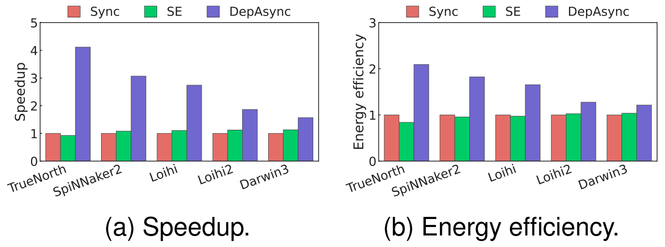


Fig. 31. Performance on different hardware parameters.

8 × 8 4-core clusters and then decrease the bandwidth between adjacent clusters. Fig. 29(a) gives an example topology of 4 clusters. Fig. 29(b) shows a linear relationship between bandwidth difference and the overall performance.

F. Comparison to Directly Removing Global Synchronization

To better illustrate the performance improvement brought by DepAsync, we also compare it with an ideal synchronization mechanism, which directly eliminates the global synchronization itself by assuming a global signal in the system that immediately advances to the next timestep once all cores have completed their work for the current step (*NoSync* in Fig. 30). We assume that there is an ideal global signal which is set when all cores have completed their computation tasks and all spikes have been successfully delivered. Once the signal is set, all cores immediately begin the next time step, thus eliminating the intrinsic overhead of the global synchronization mechanism. As shown in the Fig. 30, DepAsync achieves a 1.80x speedup, which demonstrates that on one hand, it reduces synchronization overhead, and on the other, the safe forwarding of each cores reduces waiting time.

G. Performance on Real-World Neuromorphic Chip Parameters

As shown in the Table III, different real-world neuromorphic chips have different hardware parameters and specifications. They all support LIF neuron model used in our experiments. Therefore, to demonstrate the high efficiency of DepAsync under various hardware parameters, we conducted tests on the parameters of these five platforms. As the Fig. 31 illustrates, DepAsync achieves 2.37x speedup and 1.54x improvement in energy efficiency across different hardware. Furthermore, under the same workload, the more limited the capability of a single core, the more cores are required, thus leading to a larger mesh scale. As a result, the acceleration effect of DepAsync becomes more significant, which is consistent with our results.

VI. CONCLUSION

In this paper, we propose a novel asynchronous architecture to time-accurately accelerate SNN inference on hardware inspired by dependency relationships on neuromorphic cores. We first observe the imbalance in SNN workloads deployed on many-core accelerators and identify under-utilization caused by all-core synchronization. Then, we leverage core dependencies determined at compilation time to control the timestep proceeding behavior of neuromorphic cores, implemented by adding a scheduler in each core worked with a new type NoC packet. Finally, we use five SNN workloads to evaluate our DepAsync architecture. Compared to conventional synchronous architectures, DepAsync achieves an average speedup of 2.47x and an energy efficiency improvement of 1.55x. It also outperforms state-of-the-art speculative-execution-based architectures with 1.87x higher performance and 1.40x better energy efficiency. Furthermore, our experiments at different scales show that DepAsync exhibits excellent scalability as the system size increases.

REFERENCES

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

[2] D. Ma et al., "Darwin3: A large-scale neuromorphic chip with a novel ISA and on-chip learning," *Nat. Sci. Rev.*, vol. 11, no. 5, 2024, Art. no. nwae102. 03.

[3] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[4] A. M. Zyarah, K. Gomez, and D. Kudithipudi, "Neuromorphic system for spatial and temporal information processing," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1099–1112, Aug. 2020.

[5] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, "SpinalFlow: An architecture and dataflow tailored for spiking neural networks," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA),* 2020, pp. 349–362.

[6] R. Yin, A. Moitra, A. Bhattacharjee, Y. Kim, and P. Panda, "SATA: Sparsity-aware training accelerator for spiking neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 1926–1938, Jun. 2023.

[7] R. Yin et al., "Workload-balanced pruning for sparse spiking neural networks," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 8, no. 4, pp. 2897–2907, Aug. 2024.

[8] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[9] D. Ma et al., "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017.

[10] H. Lee, C. Kim, Y. Chung, and J. Kim, "NeuroEngine: A hardware-based event-driven simulation system for advanced brain-inspired computing," in *Proc. 26th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, New York, NY, USA: ACM, 2021, pp. 975–989.

[11] H. Lee, C. Kim, M. Kim, Y. Chung, and J. Kim, "NeuroSync: A scalable and accurate brain simulator using safe and efficient speculation," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA),* 2022, pp. 633–647.

[12] P. Chen et al., "Open-loop analog programmable electrochemical memory array," *Nature Commun.*, vol. 14, no. 1, Oct. 2023, Art. no. 6184.

[13] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[14] Y. Hu, H. Tang, and G. Pan, "Spiking deep residual networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 8, pp. 5200–5205, Aug. 2023.

[15] C. Hong et al., "SPAIC: A spike-based artificial intelligence computing framework," *IEEE Comput. Intell. Mag.*, vol. 19, no. 1, pp. 51–65, Feb. 2024.

[16] W. Fang et al., "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Sci. Adv.*, vol. 9, no. 40, 2023, Art. no. eadi1480.

[17] J. C. Magee and C. Grienberger, "Synaptic plasticity forms and functions," *Annu. Rev. Neurosci.*, vol. 43, pp. 95–117, Jul. 2020.

[18] M. Soltiz, D. Kudithipudi, C. Merkel, G. S. Rose, and R. E. Pino, "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1597–1606, Aug. 2013.

[19] Y. Xiao et al., "Bio-plausible reconfigurable spiking neuron for neuromorphic computing," *Sci. Adv.*, vol. 11, no. 6, 2025, Art. no. eadr6733.

[20] A. Roy, S. Venkataramani, N. Gala, S. Sen, K. Veezhinathan, and A. Raghunathan, "A programmable event-driven architecture for evaluating spiking neural networks," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des. (ISLPED),* 2017, pp. 1–6.

[21] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[22] L. Xie et al., "SpikeNC: An accurate and scalable simulator for spiking neural network on multi-core neuromorphic hardware," in *Proc. IEEE 30th Int. Conf. High Perform. Comput., Data, Analytics (HiPC),* 2023, pp. 357–366.

[23] M. O'Boyle and E. Stohr, "Compile time barrier synchronization minimization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 6, pp. 529–543, Jun. 2002.

[24] A. Norollah, Z. Kazemi, N. Sayadi, H. Beitollahi, M. Fazeli, and D. Hely, "Efficient scheduling of dependent tasks in many-core real-time system using a hardware scheduler," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, 2021, pp. 1–7.

[25] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "Mapping very large scale spiking neuron network to neuromorphic hardware," in *Proc. 28th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, New York, NY, USA: ACM, 2023, vol. 3, pp. 419–432.

[26] O. Jin et al., "Mapping large-scale spiking neural network on arbitrary meshed neuromorphic hardware," *IEEE Trans. Parallel Distrib. Syst.*, vol. 36, no. 11, pp. 2325–2340, Nov. 2025.

[27] N. E. Jerger, T. Krishna, and L.-S. Peh, *On-Chip Networks*, 2nd ed. San Rafael, CA, USA: Morgan & Claypool, 2017.

[28] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[29] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.

[30] Y. Cao, *Predictive Technology Model for Robust Nanoelectronic Design.* New York, NY, USA: Springer-Verlag, 2013.

[31] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs*, vol. 2, pp. 1–10, Jan. 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[32] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, pp. 1–11, Nov. 2015.

[33] A. Amir et al., "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 7388–7397.

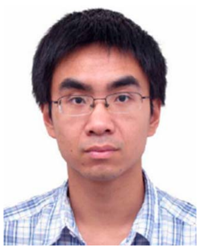
[34] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "CIFAR10-DVS: An event-stream dataset for object classification," *Front. Neurosci.*, vol. 11, pp. 1–10, May 2017.

[35] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[36] H. A. Gonzalez et al., "SpiNNaker2: A large-scale neuromorphic system for event-based and asynchronous machine learning," in *Proc. Workshop Mach. Learn. New Compute Paradigms (NeurIPS)*, 2023, pp. 1–10.

[37] G. Orchard et al., "Efficient neuromorphic signal processing with Loihi 2," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, 2021, pp. 254–259.

Zhuo Chen is currently working toward the Ph.D. degree with the School of Computer Science from Zhejiang University. His research interests include neuromorphic computing systems, with particular emphasis on the design, implementation, and application of neuromorphic chips. His work aims to bridge the gap between biological neural processing and artificial intelligence hardware through brain-inspired computing architectures.



De Ma received the Ph.D. degree in electronic science and technology from Zhejiang University, Hangzhou, China, in 2013. He is an Associate Professor with the College of Computer Science and Technology, Zhejiang University. He began his academic career in 2013 as a Faculty Member with Hangzhou Dianzi University and joined Zhejiang University, in 2018. He was an Intern with VERIMAG Laboratory, Grenoble, France, in 2010, and a Visiting Scholar with IMEC, Leuven, Belgium, in 2013. His research interests include neuromorphic hardware, VLSI design, and SoC architecture.

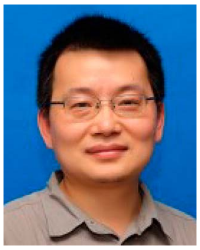
Xiaofei Jin received the M.S. degree from Zhejiang University, in 2015. He currently serves as a Senior Engineer with the Zhejiang Laboratory, while working toward the Ph.D. degree with Zhejiang University on a part-time basis. With extensive experience in multicore and many-core processor architecture design, he has recently shifted his focus to the research and development of neuromorphic brain-inspired computing chips.

Qinghui Xing is currently working toward the Ph.D. degree with the School of Computer Science and Technology, Zhejiang University. His research interests focus on neuromorphic computing and computer architecture.

Ouwen Jin is currently working toward the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University. His research interests include brain-inspired computing and neuromorphic computing hardware architectures. His work focuses on optimizing the compilation, deployment, and execution efficiency of spiking neural networks on neuromorphic hardware. He has published as the first author at ASPLOS.

Xin Du received the Ph.D. degree in computer science from Fudan University, in 2024. He is an Assistant Professor with the School of Software Technology, Zhejiang University, China. His research interests include service computing, distributed system and brain-inspired computing. He has published several papers in flagship conferences and journals including IEEE ICWS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, etc. He has received the Best Student Paper Award of IEEE ICWS 2020 and IEEE ICWS 2023.

Shuibing He is a Professor with the College of Computer Science and Technology, Zhejiang University (ZJU), China, where he leads the Intelligent Storage and Computing Systems (ISCS) Laboratory. He is also a Vice President of Zhejiang Laboratory and Deputy Director of the Zhejiang Key Laboratory of Big Data Intelligent Computing. His research interests include storage systems, intelligent computing, computer architecture, and high-performance computing. He serves as an Associate Editor for IEEE TRANSACTIONS ON COMPUTERS (TC) and previously for IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS, 2018–2022). He has served as a Program Chair of NAS 2024, General Chair of ChinaSys 2024, and Program Committee Member for ICDCS, SRDS, ICPP, IPDPS, and CLUSTER.



Gang Pan (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 100 refereed papers and was a Visiting Scholar with the University of California, Los Angeles, from 2007 to 2008. His research interests include pervasive computing, computer vision, and pattern recognition.