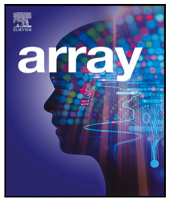
Array 20 (2023) 100323

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array



Advancements in spiking neural network communication and synchronization techniques for event-driven neuromorphic systems

Mahyar Shahsavari [a,*], David Thomas [b], Marcel van Gerven [a], Andrew Brown [b], Wayne Luk [c]

[a] Donders Institute for Brain, Cognition and Behaviour, Radboud University, Thomas van Aquinostraat 4, 6525 GD, Nijmegen, The Netherlands
[b] School of Electronics and Computer Science, University of Southampton, University Road, Southampton SO17 1BJ, UK
[c] Imperial College London, South Kensington Campus, London SW7 2AZ, UK

ARTICLE INFO

Keywords:
Neuromorphic computing
Spiking neural network
Event-driven distributed system
FPGA hardware

ABSTRACT

Neuromorphic event-driven systems emulate the computational mechanisms of the brain through the utilization of spiking neural networks (SNN). Neuromorphic systems serve two primary application domains: simulating neural information processing in neuroscience and acting as accelerators for cognitive computing in engineering applications. A distinguishing characteristic of neuromorphic systems is their asynchronous or event-driven nature, but even event-driven systems require some synchronous time management of the neuron populations to guarantee sufficient time for the proper delivery of spiking messages. In this study, we assess three distinct algorithms proposed for adding a synchronization capability to asynchronous event-driven compute systems. We run these algorithms on *POETS (Partially Ordered Event-Triggered Systems)*, a custom-built FPGA-based hardware platform, as a neuromorphic architecture. This study presents the simulation speed of SNNs of various sizes. We explore essential aspects of event-driven neuromorphic system design that contribute to efficient computation and communication. These aspects include varying degrees of connectivity, routing methods, mapping techniques onto hardware components, and firing rates. The hardware mapping and simulation of up to eight million neurons, where each neuron is connected to up to one thousand other neurons, are presented in this work using 3072 reconfigurable processing cores, each of which has 16 hardware threads. Using the best synchronization and communication methods, our architecture design demonstrates 20-fold and 16-fold speedups over the Brian simulator and one 48-chip SpiNNaker node, respectively. We conclude with a brief comparison between our platform and existing large-scale neuromorphic systems in terms of synchronization, routing, and communication methods, to guide the development of future event-driven neuromorphic systems.

1. Introduction

Neuromorphic systems enable information processing using only a fraction of the resources required by conventional computing machinery. [1]. One significant application is the development of AI solutions designed to operate on the edge, necessitating the efficient processing of substantial data streams. The requirements imposed by data-intensive computing on edge devices influence not only the creation of innovative, parallel, and energy-efficient neuromorphic hardware but also the development of software that effectively utilizes such hardware to enable machine learning applications.

In addition to affording low-power, high-performance and parallel computing, as well as collocating processing and memory units, neuromorphic systems are typically event-driven, meaning that computation occurs only when new events are registered [2]. Event-driven system promote sparsity, thereby allowing extremely low-power computation [3]. Neuromorphic systems have been either used as a large-scale hardware platform for simulation of neural information processing [4,5] or an accelerator in edge computing for inference tasks in autonomous systems and signal processing using event-driven neuromorphic sensors [6–8].

So far, different technologies have been used to construct neuromorphic hardware platforms, from bottom-up analog or mixed-signal designs such as in-memory computing [9] to top-down digital approaches such as standard multicore CPUs/GPUs [4,10]. There has also been a great deal of research into customized hardware platforms. Technologies used in customized systems include reconfigurable digital hardware such as Field Programmable Gate Arrays (FPGAs) [8], custom digital Application-Specific Integrated Circuit (ASIC) solutions and even fully analog ASIC systems [11]. FPGAs, in addition to their

* Corresponding author.
E-mail address: mahyar.shahsavari@ru.nl (M. Shahsavari).

https://doi.org/10.1016/j.array.2023.100323
Received 12 June 2023; Received in revised form 20 September 2023; Accepted 30 September 2023
Available online 5 October 2023

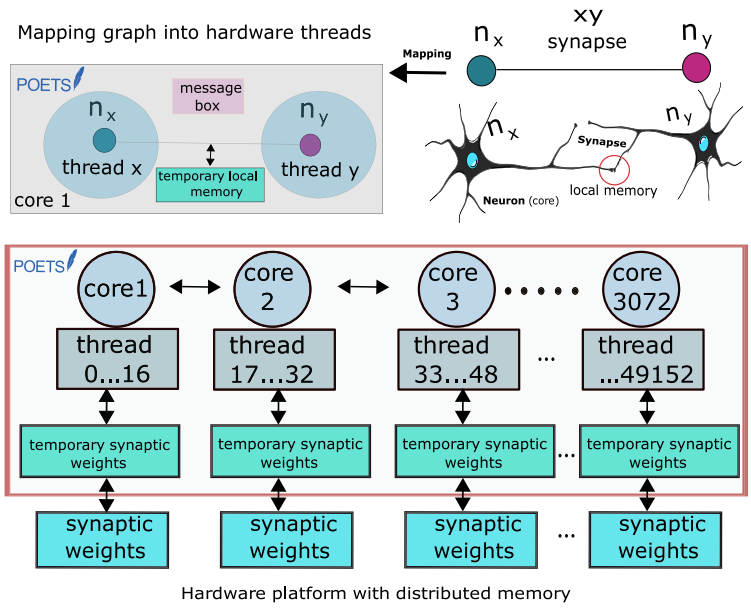


Fig. 1. A general view of mapping neurons into the neuromorphic architecture. Using the massive parallelism of POETS, each neuron can be assigned to a thread to communicate via a message box with other threads, taking the advantages of a local cache and shared memory to store synaptic weights.

programmable nature and low latency are capable of exploiting the temporal and spatial sparsity inherent to a specific problem, which is difficult to obtain using many-core digital systems [12].

A key challenge in neuromorphic design is to efficiently and effectively simulate large numbers of interacting spiking neurons [13,14]. These systems compute via the exchange of events (spikes) between simulated neurons. Information is encoded in the timing of relatively unpredictable and irregular spiking events, unlike traditional artificial neural networks running on Von Neumann architectures, which pass data as numeric values between neurons on a periodic schedule. There are traditionally two main techniques for simulating spiking neurons on hardware platforms, namely, *clock-driven* (synchronous) and *event-driven* (asynchronous) techniques [15].

Clock-driven methods are more compatible with conventional digital systems that compute with periodic timing clocks. Nevertheless, generating clock pulses consistently may not be energy-efficient, especially when events are infrequently distributed across space and time. Additionally, in clock-driven approaches, spikes may introduce a delay in calculations instead of occurring in real-time. Therefore, clock-driven methods will suffer from inaccuracies in spike timing due to the difference between a neuron's firing time and the clock timing.

Event-driven methods are closer in spirit to the spike-generating process of biological neurons. The challenge for event-driven methods, however, is to accurately generate and propagate spiking messages. Therefore, synchronization plays an important role in an event-based system, which operates by processing and responding to events or messages rather than on a fixed schedule. Synchronization algorithms are critical to the proper functioning of event-based systems, as they help ensure that processes operate in a coordinated and efficient manner, minimizing the potential for conflicts or errors. In other words, an event-driven system still needs to guarantee that there is sufficient time for neurons to produce spikes as well as for spikes to arrive at their destinations. Consequently, a certain degree of synchronization appears to be inevitable, even within event-driven systems. To tackle this challenge, our research investigates algorithms designed to establish a synchronization over an event-driven platform.

Implementing a spiking neural network (SNN) on a neuromorphic platform requires designing a system which consists of neurons as computing units, synapses as memories, network of neurons configuration and efficient methods for spike transmission. The development of such systems relies on multiple lines of research, ranging from modeling of individual neurons [16,17] to designing large-scale neuromorphic hardware [4,5,11]. In the present work, we describe the implementation of an asynchronous large-scale Spiking Neural Network (SNN) on the Partial Ordered Event Triggered Systems (POETS) [18] platform, as shown in Fig. 1. POETS has been used in various domains required event-driven and distributed computation such as neuromorphic design including SNN learning [19,20], large-scale Petri net simulations [21], Tsetlin Machines in machine learning domain [22], and Dissipative Particle Dynamics (DPD) [23]. In this research, using POETS as a neuromorphic computing platform, we explore how computational tasks will be distributed between different threads and focus on the question of how effective transmission of spike events can be realized via different synchronization methods. We focus on message passing and communication effectiveness between nodes in neuromorphic systems, as this has not been yet evaluated in depth in the literature. This can be contrasted with our previous work, which focused on exploring learning in SNNs using the same hardware [19].

Our research contributions are as follows. First, we compare three synchronization algorithms for event-driven asynchronous hardware, comparing both speed and spiking accuracy. A recently developed hardware-based synchronization barrier suitable for neuromorphic system will be analyzed using different networks in this research. Second, using the best synchronization algorithm, we evaluate hardware speed including comparison using 1, 2, 4, and 8 boxes of our hardware platform which has more than 3000 customized processors that cover up to 50,000 threads. Using the setup, we explore the communication between hardware components, routing methods, and local adaptive versus fixed firing rates. Third, an empirical speed performance comparison with existing platforms shows that POETS is over 20 times faster than the Brian 2 simulator and over 16 times faster than the SpiNNaker system [4]. Finally, we review the communication properties of existing large-scale neuromorphic platforms and compare these with the methods developed using the POETS framework that could be useful for the future neuromorphic system design.

2. The POETS event-driven platform

Neuromorphic computing hardware platforms generally make use of processors such as multi- and many-core processors, ASIC chips, or GPUs, which offer different degrees of programmability.

Platforms using ASICs have high-performance and high-speed computing capabilities in addition to offering low power consumption. However, their architecture is fixed, which limits their use when adopting different designs. This is particularly important in neural network implementations, which rely on various network and communication parameters.

Many-core microprocessor-based architectures offer user-friendly modeling of neuromorphic systems, yet they suffer from the Von Neumann memory bottleneck, particularly in a large network with high demand for memory–processor communication. Simulating spiking neural network using GPUs provides some speedup using multi-core processing units as well as flexible programmability, but the drawback particularly in implementing scalable SNNs is their large power consumption.

FPGAs have previously been used as an accelerator for neural network simulation [8,24–26]. High flexibility to customize the system design, fine-grained parallelism and high scalability makes FPGAs an appropriate candidate for developing a neural network hardware simulator. In this work, we make use of POETS (Partial Ordered Event Triggered Systems) as a flexible neuromorphic hardware simulator platform to simulate large-scale biological models using spiking neurons [13,19,20,27].

2.1. Programming model

A cluster is a set of computers that work together to perform the same task as a single system. POETS is a reconfigurable cluster-based many-core machine with a high-level API (Application Programming Interface) based on a series of hardware and software design and development suitable for users with no prior knowledge of hardware or FPGA technology to facilitate design space exploration at large scale for neuromorphic systems. The POETS platform is designed to support the event-driven parallel programming model of numerous simple nodes, which is ideally suited as the basis for a neuromorphic architecture. This platform is designed for parallel computing by mapping an application graph to a physical network. In the model, vertices represent finite-state machines (FSMs) capturing state and computation, while edges represent connections between vertices capable of passing messages between vertices. Computation occurs whenever a message, referred to as an "'event" is delivered from or received at a vertex. This event-driven process is entirely asynchronous and concurrent. Each FSM will be updated independently, and there is no shared state within the system. The system guarantees that only one event can happen at a vertex at any time and once an event is sent it will eventually be delivered. However, there is no guarantee on the ordering of message arrival.

2.2. Hardware architecture

The current POETS hardware system is designed in terms of "boxes", with each box containing six FPGAs located on DE5-Net boards, providing 3072 RISC-V processing cores and, 49,152 threads when using all eight existing boxes. Each DE5-Net supports 2×4 GB DDR3 DRAMs, 4×8 MB QDRII+ SRAMs, and $4 \times 10G$ SFP$^+$ (Small Form-factor Pluggable) ports. One FPGA is used as a PCIe to SFP+ bridge board, providing a fast connection between the x86 and the remaining six worker FPGAs (Fig. 2.a). The system supports both asynchronous and synchronous message exchange transferring between the cores. The current configuration of hardware is represented as a *Tinsel overlay* [28], which is a highly-parameterized design taking advantage of the capability of adjusting design parameters per application. However, to simplify the presentation, we will assume a default configuration of the overlay on top of our current cluster. The overlay contains a highly regular structure consisting of a scalable grid of tiles connected by a reliable communication fabric that extends throughout the cluster. In the default Tinsel architecture, each FPGA DE5-Net board has 16 tiles (Fig. 2.b). A tile, depicted in Fig. 2.c, consists of the following components.

1. Core: Each tile contains 4 RISC-v cores and the number of threads per cores is 16. In total, each FPGA board has 1024 threads.
2. FPU: Each tile shares a 32-bit FPU (floating-point unit) between four cores. Latency in IP core is the number of cycles it takes for an accumulation to propagate through the block from input to output. The FPU operates using Altera IP blocks that have 14 cycles latencies at 250MHz.
3. Partitioned data cache: The data cache part is an 8-way set-associative write-back used to access off-chip memory. The threads are non-coherent, hence, there is no support for communication via shared memory.
4. Mailbox: Each mailbox sends and receives messages between threads, supporting up to 1 KB of space. Mailboxes are connected together to establish a distributed network supporting unicast and multicast routing methods.

2.3. Software toolchain overview

Real-world applications such as World Wide Web, medical informatics, social network, and machine learning libraries make use of graph-based algorithms [29]. POETS with the capability of fast and parallel processing at large scale uses graphs to map applications to the hardware. To perform graph processing, our framework provides high-level programming models for the users to easily create the graphs and edges without dealing with low-level FPGA coding. Users must think about how to break down their application components into vertices and edges. For instance, in the SNN case, the neurons in the spiking neural network are vertices and the edges are synaptic weights between the neurons. Currently, there are two different toolchains for mapping the simulations and applications to hardware, namely, Graph Schema and POLite [28].

Graph Schema. This toolchain transforms the abstract representation of graphs into an XML specification containing a GraphType and a GraphInstance section. In the GraphType section, the user defines the types of vertices and edge. Subsequently, vertices and edges are instantiated in the GraphInstance section to create a graph network, determining which device is connected to which (weighted) edges. A vertex device processes an event via a message handler to be delivered over edges. This event-driven system works with an FSM, which controls the states of devices synchronously. If a message arrives or is sent via an edge, the FSM is in *OnReceive* or *OnSend* respectively. Firstly, the graph configuration including the vertices and the connection map (edges) will be provided as an XML input file. In the second phase, the XML input configuration map will be read by a Python programmed generator, which can add the device state and properties to the graph in addition to run simulations and create the weighted edges to produce the XML output map. A configuration system, namely 'Orchestrator', handles the intermediate translation from high-level specification to machine-executable code to be run on FPGAs. More details could be found in [13]. The disadvantages of using Graph Schema together with Orchestrator is using different languages e.g., Python, XML, and C++. Therefore, to provide less headache for users to run applications on POETS, another toolchain has been developed that we used it in this research.

POLite. Another high-level programming environment toolchain is POLite [28], which is similar to the vertex-centric paradigm [29,30] but supports both synchronous and asynchronous messaging. POLite is a layer of abstraction that manage mapping arbitrary graphs onto the Tinsel overlay. POLite is a C++ light software layer on top of the Tinsel hardware designed to implement a graph-based event-driven abstraction, while it is able to hide architectural details from the user. Similar to Graph Schema, the vertices receive the events and if the state of the vertex is changed, it will send a message via the edge to the other connected vertices. The event messages are treated by the following event handlers:

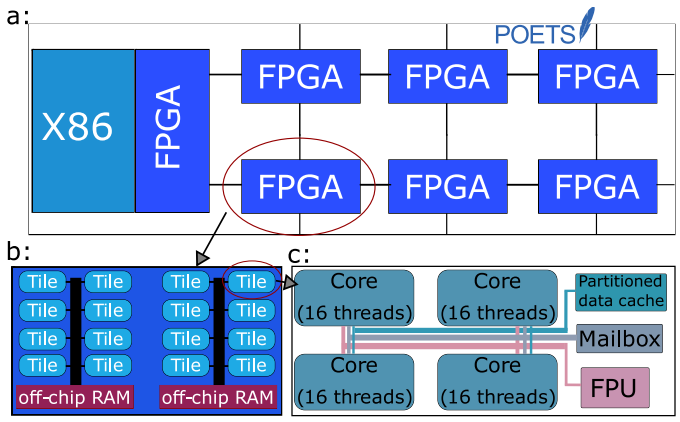


Fig. 2. **(a)** One POETS box. Each box has 7 FPGAs, 6 workers connected to an X86 GPP (General Purpose Processor) via one intermediate FPGA. **(b)** A default configuration of the Tinsel Network on Chip (NoC) on a single FPGA. **(c)** Default structure of a Tinsel tile. The cores are highly reconfigurable and extendable softcore processors.

- *Send handler:* Send handler provides a message buffer, to which the outgoing message should be delivered.
- *Receive handler:* If a message arrives, the receive handler of the vertex will be called with a pointer to the message and a pointer to the weight associated with the edge along with the incoming event.
- *Step handler:* The step handler is called whenever no vertex in the graph wishes to send, and there are no messages in-flight. It returns a flag, indicating whether the vertex wishes to compute. Typically, an asynchronous application returns a false, while a synchronous one will compute for the time-step, and requesting to send again. It returns true if it wishes to start a new time-step.
- *Finish handler:* If the previous call of the step handler returned false from every vertex, then the finish handler is called. The finish handler can only be invoked when all vertices in the entire graph have no demand to continue.

3. SNN simulation and synchronization in event-driven systems

Machine learning algorithms have been used to solve tasks in automation, recognition, classification, and prediction without being explicitly programmed [31]. In other words, they are capable of learning from data. Artificial neural networks are a class of machine learning models that are loosely inspired by their biological counterparts [32]. Conventional neural networks continuously transmit real-valued signals between neurons that can be interpreted as firing rates. In contrast, spiking neural networks (SNNs) operate via the transmission of discrete events, referred to as spikes, mimicking action potential generation in the brain. SNNs offer huge energy savings due to their sparse event-driven nature, however, training and simulating such models remains challenging [33].

3.1. Modeling and simulating SNNs

A SNN is a mathematical model of biological neural network which mimics the behavior of a biological neuron, spiking communication and synaptic plasticity. At each time-step, each neuron is updated using its current state and the weighted sum of input spikes from the previous time step to produce a new neuron state. We define the SNN in terms of a set of N neuron states $s_1,\ldots,s_N$, a $N \times N$ weight matrix W and a neuron update function ξ. The update function computes for each neuron the new state s_i^{t+1} and spiking event f_i^{t+1} from the current state s_i^t and the weighted sum of spikes $I_i^t = \sum_{j=1}^{N} W[i,j] f_j^t$. That is,

$$\left(s_i^{t+1}, f_i^{t+1}\right) = \xi\left(s_i^t, I_i^t\right) . \tag{1}$$

Synapse delay refers to the time it takes for a spike to travel from the presynaptic neuron to the postsynaptic neuron. Biological networks have variable synapse delays, and there is some work showing configurable synaptic delays might also have a positive impact on learning [34]. However, in this work we focus only on the zero synapse delay case, where spikes generated at time t are always delivered at time $t+1$. Our reasoning is that delivering spikes with no delay is the hardest problem to solve for synchronization and event delivery, as we cannot use buffering to improve throughput. We also observe that longer delays can always be added to a zero synapse delay systems using local memory buffers at neurons, and as long as the underlying zero synapse delay synchronization is correct, the system with delays will also be correct.

In case of on-chip learning with weight modification, we generalize this notation to

$$\left(s_i^{t+1}, f_i^{t+1}, W_i^{t+1}\right) = \xi\left(s_i^t, I_i^t, W^t\right) \tag{2}$$

with W_i^{t+1} the vector of input weights for the ith neuron. This system could be analyzed as a dense matrix–vector multiplication for small N, or as a sparse matrix–vector multiplication for larger N, however, eventually the sparsity and unpredictable nature of the firing vector f makes the computation inefficient. We assign each neuron n_i into its own independent state machine in hardware. To this end, we define $S_i = (s_i, I_i)$ as the corresponding hardware state, which may include any extra information related to synchronization.

In event-driven systems, we need a synchronization barrier to manage when to start a new calculation to separate the current and the next computation steps. This barrier could be a software or hardware barrier. Table 1 lists the response to synchronization barriers. In case of a software barrier no *Idle* event happens and for a hardware barrier there is no action in the *Send* event. That is, in the hardware barrier, the neuron accumulates the received spikes and will send when an *Idle* event happens. The weights W are encoded as edge weights on a static graph, and can be delivered very efficiently. Another feature allows us to disable the outgoing message from a send event, so if $msg.f$ is false, the run-time will not send that particular message. In the hardware barrier version, the system only checks the vector f and there is no message related to the firing information.

3.2. Different algorithms for SNN in event-driven platforms

In neuromorphic systems based on parallel computing platforms, the method of message-passing between processes and threads is a significant challenge to address. The key advantage of using event-driven asynchronous communication is the potential of speeding up the computation and propagation compared to clock-driven systems since processing is only needed in case an event happens. This also leads to

Table 1
Response of neuron to synchronization barriers.

Barrier	Send	Receive	Idle
Software	$(s_i, msg.f) = \xi(s_i, I_i)$; $msg.src = i$; $I_i = 0$	$I_i = I_i + W[i, msg.src] \times msg.f$	N/A
Hardware	N/A	$I_i = I_i + W[i, src] \times f$	$(s_i, f) = \xi(s_i, I_i)$; $src = i$; $I_i = 0$

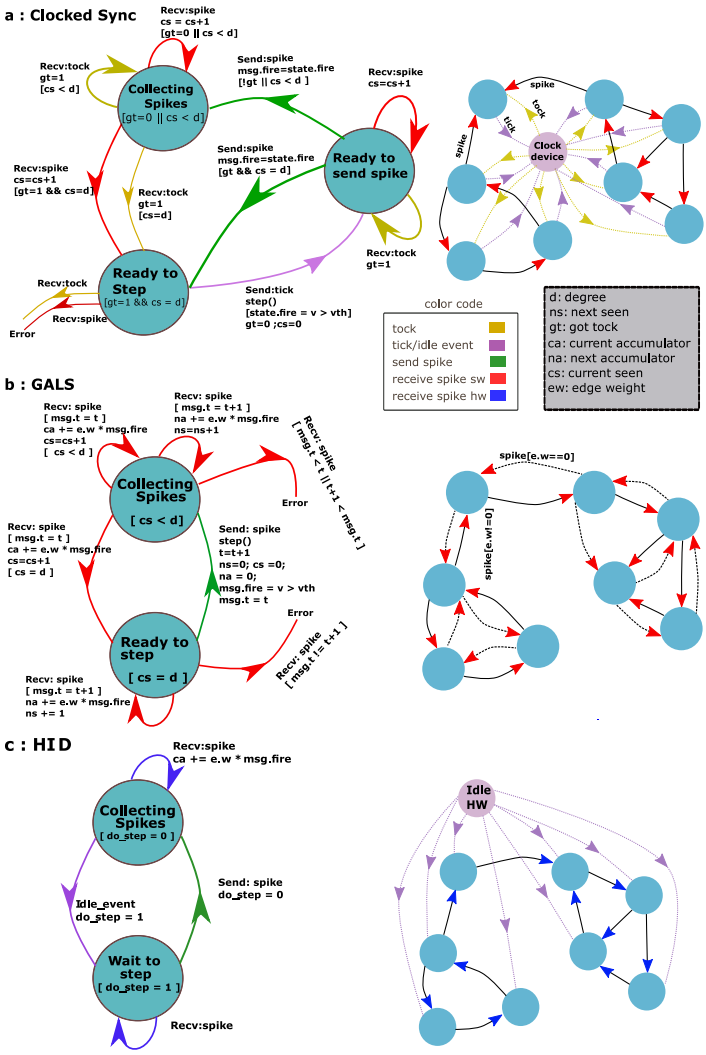


Fig. 3. A finite state machine that can be applied for three different synchronization methods in event-driven networks. **(a)** Clocked synchronization (CS) method, **(b)** globally asynchronous locally synchronous (GALS) communication, **(c)** Hardware idle detection (HID). The left side of the figure shows the Finite-state machine (FSM) and the right side shows the network configurations in respect to the synchronization method illustrations.

a more precise representation of spike times. Furthermore, in event-driven systems, spikes can be produced by a neuron only at times of incoming spikes, which makes event-driven simulation relatively easier than clock-driven simulation [15].

Although spikes propagate between the nodes asynchronously, we do need to ensure that a *Send* message is invoked only when all incoming messages from the previous time step have been collected. This is particularly important in POETS, which guarantees the arrival of messages, but not their order of arrival. We may address this problem using different algorithms for synchronization, as depicted in Fig. 3. First, a software barrier known as *clocked synchronization (CS)*, second, *globally asynchronous, locally synchronous (GALS) communication* and, third, a hardware barrier known as *hardware idle detection (HID)*. In the following, we describe these approaches in more detail.

Clocked synchronization (CS): In a computing system, a clock device is employed to not only internally alter the states of neurons but also synchronize messaging with neighboring neurons. We define three states for this method: *Collecting spikes*, *Ready to step*, and *Ready to send spike* in a state machine (See Fig. 3.a). In the CS method, the clock device in each state-machine has to send a message to every neuron in each time step, regardless of whether it spikes or not. Hence, the basic considerations of the algorithm are as follows:

- Two messages are used between the clock-device and neurons named 'tick and tock' that carry only the internal neuron information, not any spike.
- An internal counter is defined to compare the number of messages from different inputs (seen) with the degree of connections.
- After neuron firing, the counter and internal state of the neuron will be reset.

For a transition from the *collecting spike* state, if a tock message arrives and the number of times that neuron has been seen by connected neurons is equal or bigger than the degree of connections in each neuron, the state will change to *ready to step*. This method is known as a voting mechanism, which means that all connected nodes agree on delivering their sending spikes in the previous time step after tick-tock messaging. In the *ready to step* state, neurons announce their membrane potential firing state and unconditionally the state will change to *ready to spike* and another tick will be sent to inform connected neurons. In the *ready to spike* state, the neuron will send the firing state to its connected neurons.

In CS, whether the neuron fires or not, it sends out a firing message to inform other connected neurons about its firing state. The number of messages in one time-step in this method is $n \cdot (d + 2)$, where n and d are the number of neurons and the degree of connections, respectively. For asynchronous hardware like our platform, this method is not recommended for large networks, as it has the drawbacks of full synchronization. However, this method is recommended for small-to-medium-sized networks in which accuracy is of crucial importance.

GALS: Globally asynchronous, locally synchronous (GALS) communication is a method that enables the synchronization of a group of locally clocked modules to facilitate effective communication among [35]. GALS has more degree of freedom compared to the CS method by eliminating the need for a global clock. GALS systems, with their localized clocks, provide better fault isolation. If a module fails, it primarily affects its neighboring modules, minimizing the impact on the entire system. GALS systems are highly scalable as they can easily accommodate the addition or removal of modules without affecting the overall system timing. This scalability is particularly beneficial in large-scale systems where CS becomes more challenging. By allowing individual components to operate at their own clock frequencies, power can be saved by avoiding the need for high-frequency operation throughout the entire system [36]. The following is a list of fundamental considerations for GALS methods in our system:

- When a neuron receives an incoming spike, it sends a message to all of its local neighbors to inform them whether it spikes or not.
- The neuron could then only progress to the next time-step if it has received a number of messages equal to the number of input connections.
- There are two sets of counters, in which one set tracks the number of spikes and total stimulus accumulated in the current time-step, while the other set has been dedicated to the next time-step.

As shown in Fig. 3.b, the machine state will change from *Collecting spike* to *Ready to step* where in each transition the edge weight will be modified in case of on-chip learning. Messages have to be returned from the neighboring neurons at every time-step. The total number of messages in one time-step in this method is $2 \cdot n \cdot d$. This implies that in larger networks, the number of messages surpasses that of clocked synchronization (CS). Nevertheless, removing global synchronization will accelerate computation, especially in non-fully connected networks running on distributed, federated or heterogeneous systems can facilitate parallelization. Running this method on our platform, in addition to reviewed benefits of GALS, we predict more capability to speed up the computation and communication due to the lack of global clock connections.

Hardware idle detection (HID): In this research, we introduce HID for the neuromorphic event-driven system synchronization based on termination detection [28]. A hardware barrier synchronization will synchronize the event-based neurons in the HID method. It means a neuron does not wait for any other neurons to receive messages. The HID method is designed for globally-asynchronous applications, and it uses a signal to ensure there is no undelivered message in the system. HID identifies an event when there is no thread in the system with pending send, receive or computation tasks, known as *hardware idle*. This hardware idle state keep the neurons synchronized. The state machine is shown in Fig. 3.c which represents *Collecting spike* and *Wait to step* states. HID implements the following in our neuromorphic system:

- When a message reaches a neuron, the handler will establish two pointers, one pointing to the message itself and the other pointing to the weight linked to the incoming edge through which the message has arrived.
- When idle detected, there is no send or receive message in-flight.
- In the step state, the neuron decides to fire or not, but always starts at a new time-step.

In the HID method, the number of messages in one time-step is $n \cdot d \cdot f_r$, where n, d, and $f_r \in \{0, 1\}$ represent the number of neurons, the degree of connections and the spiking state, respectively. Therefore, in those time steps in which a neuron generates no firing spike ($f_r = 0$), it propagates no message to the system. This version of message communication is simple and generates a smaller number of messages in the network compared to CS and GALS. Furthermore, there is no synchronization except an idle detected hardware signal, which makes this method faster compared to the other two methods. However, we need to provide a handler which is called for all nodes. Hence, any node which takes longer to reach the barrier due to having more messages to handle, will enforce other nodes to sit idle. It takes more execution time, particularly if the tasks to be executed are Heterogeneous. Consequently, as the synchronization barrier will happen in the hardware, there will be less control and debugging facilities if there is an error in the system. Moreover, the advantage of employing HID is substantial when dealing with small graphs. However, this advantage diminishes as the computational load per time step increases, whether by assigning more vertices to each thread or by augmenting the fan-out and consequently the quantity of messages handled by each thread [37]. Conversely, if the computational load per time step decreases, such as through the addition of more cores, we can anticipate an enhanced benefit from HID.

4. Mapping the Izhikevich model on POETS

POETS is designed to efficiently simulate highly scalable event-based models. We will focus on simulating spiking neural networks that implement parallel distributed processing at large scale. In the previous section, we presented a generic model that could be used for any neuron model. Here, we demonstrate how the Izhekevich model can be mapped on hardware. The balance of excitatory to inhibitory neurons ratio in the spiking network is 80% to 20% respectively.

4.1. Izhikevich spiking neuron model

The Izhikevich model combines the physiological plausibility of the Hodgkin–Huxley model with the computational efficiency of integrate-and-fire models of a neuron [16].

$$\frac{dv(t)}{dt} = 0.04v^2 + 5v + 140 - u + I(t) \tag{3}$$

$$\frac{du(t)}{dt} = a(bv - u) \tag{4}$$

If a neuron reaches the threshold voltage of its membrane potential, it fires and the post-spike reset is given by

$$v(v > v_{th}) = c \text{ and } u(v > v_{th}) = u + d \tag{5}$$

The states and the parameters used in the Izhikevich model are presented in Table 2.

Table 2
Izhikevich variables modeled in POETS.

Variable	Meaning	Type
v	Membrane potential	`float`
u	Membrane recovery variable	`float`
v_{th}	Threshold voltage	`float`
I	Synaptic currents	`float`
t	Simulation time	`uint32_t`
dt	Time steps	`uint32_t`
a	Time scale of recovery variable u	`float`
b	Sensitivity of recovery variable u	`float`
c	After-spike reset value of v	`float`
d	After-spike reset value of u	`float`
$spikeCount$	Spike counter	`uint32_t`

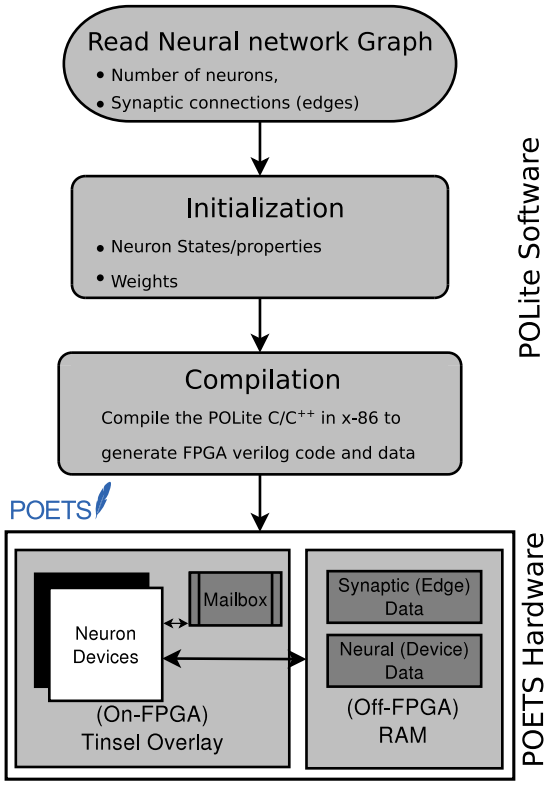


Fig. 4. Software and hardware design flow from high-level user programming to low-level hardware FPGA implementation of SNNs.

4.2. Mapping network components on POETS

The SNN is mapped as a graph on POETS where the neurons represent the device vertices and the edges of the graph represent plastic synaptic connections between neurons. The POETS ecosystem supports mapping different feed-forward, including local circuitry similar to convolutional neural network (CNN) or recurrent network topologies. After mapping the network topology graph onto the hardware, the connections remain fixed during the running time of SNN on hardware. Firstly, we define the number of neurons and the connection between the neurons. The neuron model will be defined, and the parameters will be initialized subsequently. In the next step, the high-level software design will be compiled using Graph Schema or POLite to be transferred to FPGA as Verilog code and data. Finally, the code will be mapped onto the POETS hardware. Fig. 4 shows the steps from high-level modeling to the hardware mapping using the POLite application to simulate a neural network on POETS. The same steps are presented using a Graph Schema API in previous work for simulation of SNNs on POETS [20].

Table 3
Characteristics of one DE5-NET FPGA board.

FPGA Model	Stratix V GX (5SGXEA7N2F45C2)
Core (Tinsel)	64
Threads (Tinsel)	1024
Logic Elements(LEs)	622,000
DRAM	2 × 4 GB DDR3
SRAM	4 × 8 MB QDRII+
FLASH	256 MB
FPGA Clock Frequency	250 MHz
DSP blocks	256 (27 × 27 bit)
Power	<50 W

4.3. Routing methods

Routing algorithms serve the purpose of directing data packets, while each algorithm being a software tasked with determining the most efficient path for transmitting a packet. Achieving effectiveness and efficiency in routing stands as a paramount aspect within NoC-based (Network on Chip) neuromorphic systems [38]. Message delivery in POETS system is guaranteed by the hardware provided that all threads eventually execute the messages available to them. Threads communicate with each other via mailboxes. Two different methods have been used for message communication between threads, namely unicast and multicast. In the unicast method, there is a point-to-point communication between two threads in which a single packet is sent to a single destination. The aim of multicasting is to send the same message to multiple destinations while minimizing messaging traffic in the system. Messages first will be delivered to the programmable routers, which automatically propagate messages to any number of destination threads distributed throughout the cluster. If a router supports multicast routing, then neurons with a high fan-out can be communicate efficiently with minimizing inter-FPGA bandwidth while offloading work from the processing cores. Tinsel provides both unicast and multicast communication by having a programmable router on each FPGA board to support global multicasting.

5. Experimental validation

In this work, the FPGA-hardware implementation for up to two million neurons on one cluster (box) and up to 8 million neurons on 8 clusters is performed while each neuron is connected to 100 to 1000 other neurons. Most of the SNN mapping is implemented using one POETS box which includes six FPGA boards. By employing an extensive setup, we aimed to push the system to its limits, testing its performance and scalability across 8 interconnected POETS boxes using all 48 FPGAs and 49,152 threads uncovering the full potential of the platform. It provides valuable insights into capabilities of system in tackling complex computational tasks and addressing large-scale problems. We have also compared the speed performance of the network with a different number of neurons using 1, 2, 4, and 8 of POETS boxes. The properties of one FPGA board are listed in Table 3.

Each box is hosted by an x-86 machine with 28 cores Intel(R) i9-7940X CPU@3.10 GHz. Although in previous work, we have used MNIST data set as an output due to establishing a learning algorithm on the hardware [19]. In this research, the input spiking data is generated randomly using a normal distribution for different networks to verify the network capability running different number of neurons. We implemented a random recurrent neural network with sizes from 100 to 8 million nodes and placed on one to 8 hardware boxes. The testing benchmark is similar to [13] but in more scalable sizes. The critical check points for the number of nodes are 100, 200, 500, 1k, 10k, 50k, 100k, 500k, 1000k, 2000k, 8000k. The number of synapses per neuron has been defined to verify a normal and extreme number of connections, 100 and 1000 synapses per neuron. Due to more robustness of results while using the random seeds inputs, we average

out the effects with considering statistical average of 10 times running of each implementation. The weights are initialized randomly using Gaussian random initialization: Each weight is randomly assigned a value from a Gaussian (normal) distribution with a mean of zero and a standard deviation that corresponds to the chosen weight range. For connection weight modification, The system supports STDP (Spike Timing-Dependent Plasticity) and reward-based STDP which have been presented in our previous works running on the same platform [19,20]. In this work, we focus on neuron activities and message communication while STDP rule is used for weight modification without decoding any input data pattern.

5.1. Messaging results

Here we compare the speed and spiking activity accuracy of three different synchronization algorithms. In addition, we observe the distribution of communication between neurons assigned to different hardware components. We compare synchronization and routing methods as well as the impact of firing rate on simulation time.

5.1.1. Comparing speed performance of different synchronization algorithms

We evaluate the speed performance and spiking activity accuracy of three different SNN algorithms for synchronization in communication between the neuronal nodes running on POETS. The maximum number of neurons implemented on one box using CS and GALS is limited to 500 thousand for each algorithm, while using hardware idle detection (HID) the system is capable of implementing two million neurons. The HDI algorithm for synchronization not only is able to run larger networks on the same hardware platform but also is faster than the two other algorithms, as demonstrated in Fig. 5. In the clocked synchronous algorithm, each neuron sends a message to all neurons it is connected to, regardless of whether that neuron fires or not. This property facilitates debugging and reporting, but also increasing the latency of execution and compilation. In the GALS version, the system is more flexible in checking and verifying the neuron updates, getting rid of global clock. To observe the effect of routing methods, we compared the speed performance of network choosing one of the synchronization methods. The impact of multicast routing methods on the speed performance of the network has been compared to the same synchronization algorithms using unicast routing, shown in Fig. 5. The results demonstrate the recent advancements of routing method by applying multicast routing in the system, improve the speed of simulation particularly for larger networks.

5.1.2. Comparing spiking activity accuracy of different synchronization algorithms

POETS is designed to simulate large-scale event-driven systems but also allows reporting the spiking behavior of a single neuron as well as a population of neurons. The spike activity accuracy of a single neuron depends on simulation of the membrane potential dynamics. To evaluate how accurate a neuron fires, we compare the neuron's spiking activities in the Brian simulator [39] as a reference. We run the hardware implementation model 10 times for each method to obtain a reliable accuracy estimation. For the CS method, the spiking activity accuracy is 96.2%. For the GALS method, accuracy is 93.8% and the overlap of the spike timing compared to Brian is less than the other two models. The HID method has an accuracy of 96.1% with a better overlap of the spike timing over the Brian model. The raster plot of Fig. 6 shows the overlap of spiking activity for a group of 100 neurons running on POETS compared to the Brian software simulation.

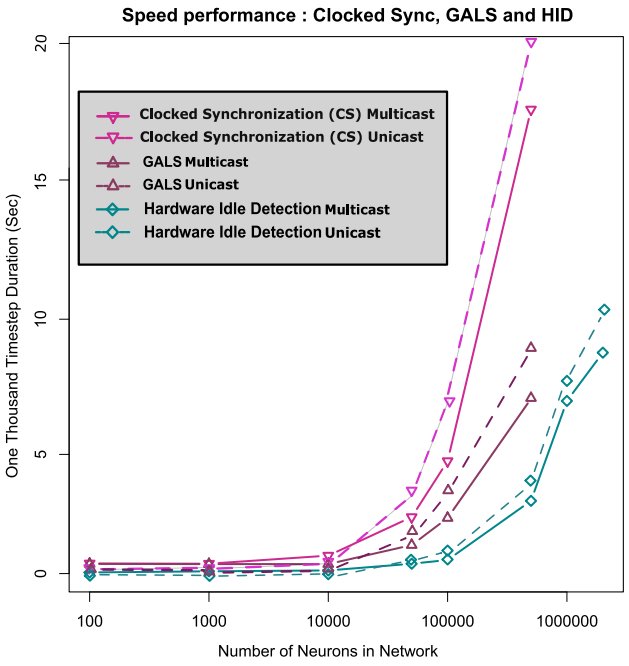


Fig. 5. Comparing the clocked synchronization (CS), hardware idle detection (HDI) and GALS synchronization algorithms. For observing the impact of routing methods on speed performance for each synchronization, both multicast and unicast results are depicted.

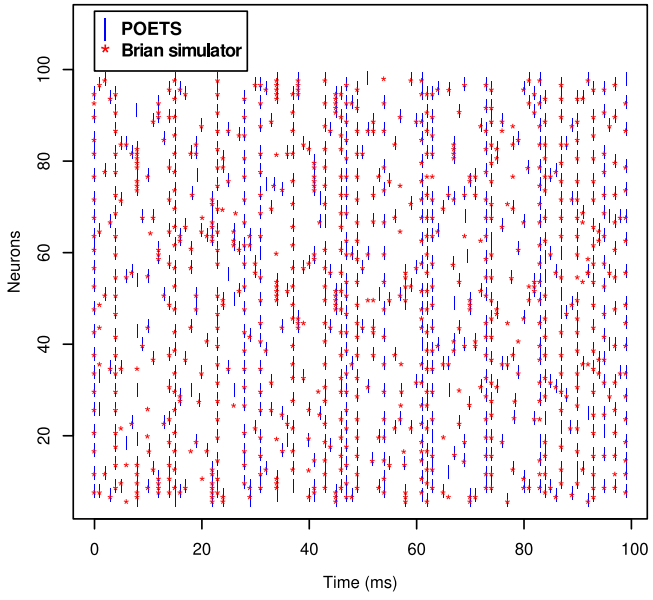


Fig. 6. Raster plot of spiking activity of 100 neurons, POETS versus Brian simulator, the activity accuracy could be observed in overlapping points referencing the Brian simulator.

5.1.3. Neuron mapping distribution on different hardware components

The POETS ecosystem mapping strategy works efficiently depends on the number of vertices and edges in the mapped graph. If the vertices could be assigned parallelly into the threads of one physical board, it will not use two boards to address high-speed communication and energy saving challenges. The threads on neighboring mailboxes communicate faster compared to threads on further mailboxes, causing these to spend more time and consume more bandwidth in the system. Therefore, it is important how neurons communicate with each other on the same thread, or different threads while still sharing

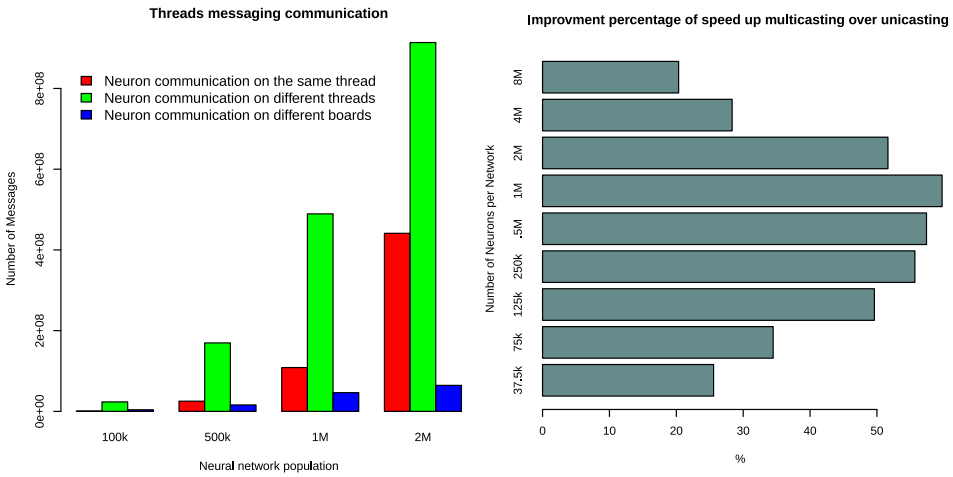


Fig. 7. Message distribution between neurons on the same thread, different threads on the same FPGA boards, and using different FPGA boards (left). The percentage of improving the multicast over unicast routing methods (right).

the same mailbox, or different threads sharing different mailboxes on one or more FPGA boards. The system supports efficient parallel communication and computation if each neuron is assigned to one thread, instead of several neurons to one thread. However, for large-scale networks in which the number of neurons is larger than the number of threads, several neurons will be mapped on the same thread. We can map neurons on threads in different boards, but in this case we need to consider inefficient communication due to distant threads and mailboxes. Hence, there will be a trade-off between parallelism and communication distance. Fig. 7 (left) shows for networks with a different number of spiking neurons that neurons on different threads have the highest communication rate.

5.1.4. Routing method message-passing comparison

There is a trade-off between offloading work from the cores and overloading the programmable routers. Messages which are closer to the destinations consume the less space and energy on the network. As shown in Fig. 7 (right), the effectiveness of using the multicast method compared to the unicast method is most pronounced for medium-sized networks. In our experience, a mix of local sending in unicast methods and offloading via the programmable router is an efficient solution. For very large number of neurons that occupy all eight boxes and a few neurons that could be run on single FPGA boards using the programmable router is less advantageous comparing to the average number of neurons between the lowest number of neurons and the highest system scalability limitation. For more information regarding the routing and message-passing, we refer to [40].

5.1.5. Impact of firing rate on simulation time

If the firing rate of a network is higher, the system has to carry more messages and consumes more energy. By changing the firing threshold parameter, it is possible to vary the firing rate of a neuron. In larger networks, the firing rate might be quite different from one neuron to another neuron due to the number of input spikes to each neuron, as well as the degree of spiking sparsity in the network. This variability leads to the instability in the firing patterns of neurons. Consequently, it may happen frequently that some neurons generate spikes very frequently while other neurons remain silent for a long period of time. In the large networks, defining a local adaptive threshold is useful to restrict the variability of the firing rate of neurons, which is known as homeostasis [41]. Fig. 8 (left) shows how the adaptive threshold could limit the firing variability in a network for a larger number of neurons. To observe the impact of threshold on the speed of simulation, we varied the firing rate from zero to 100 per time step. As the results depict in Fig. 8 (right), increasing the firing rates leads to increasing the simulation time, but the slope of the increment is supra-linear.

5.2. Hardware simulation speed

In large-scale systems, the speed of hardware simulation is a key parameter to determine system performance. To evaluate POETS, we simulated networks with different numbers of neurons, starting from 100 up to 1 million. We used all possible variations of different POETS boxes (i.e., 1, 2, 4, and 8 boxes). Two different degrees of connections are used in the simulation of 100 and 1000 number of synapses for each neuron. The results are shown in Fig. 9. For a small number of neurons, the simulation time is close to constant. It shows that as long as the number of neurons is less or equal to the number of threads, the computation is parallel and the running time does not change dramatically. Running time strongly increases as the number of neurons increases from 10,000 to 50,000 neurons. In this case, there is a transition from parallel processing to the partially parallel processing regime where each thread has to handle hundreds of neurons for computing and messaging. This dramatic change in simulation time is most obvious in networks with 1000 synapses as the number of messages increases to a billion scales, for example in 8 boxes with 50 thousand neurons (close to the number of threads).

To assess the speed performance of POETS, we need to compare it with other simulators and platforms. We compare the speed of running the network simulation on POETS with the Brian simulator and the SpiNNaker [4] platform using the same number of nodes starting from 100 up to 2 million neurons implemented on one box and 8 million neurons on eight boxes. As demonstrated in Fig. 10, the simulation time for a network consisting of two million neurons is 8.16 and 4.67 s using one box and eight POETS boxes respectively. We used the same networks with the same number of connections for Brian simulation. Results show that the hardware implementation on POETS is more than twenty times faster than the Brian simulator. Additionally, comparing the system with one 48-chip SpiNNaker node shows that POETS is at least 16 times faster. This speed comparison is made using optimal communication parameters such as HID synchronization, multicast routing, and considering homeostasis for POETS as discussed previously.

6. Existing neuromorphic hardware platforms

One of the main goals of this research is to support engineers in designing new neuromorphic platforms. We briefly review other well-known neuromorphic systems with a focus on communication methods. From the technological point of view, we recognize three approaches to create a large-scale neuromorphic platform:

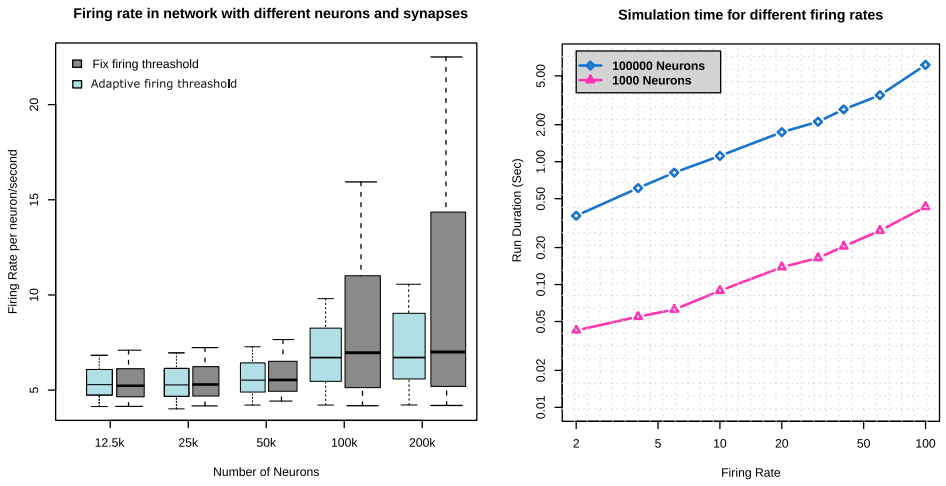


Fig. 8. The left side shows a comparison of the variety of firing rates in networks of neurons with fixed firing threshold and local adaptive firing threshold. The right side shows the impact of increasing the firing rate on the run duration of the network.

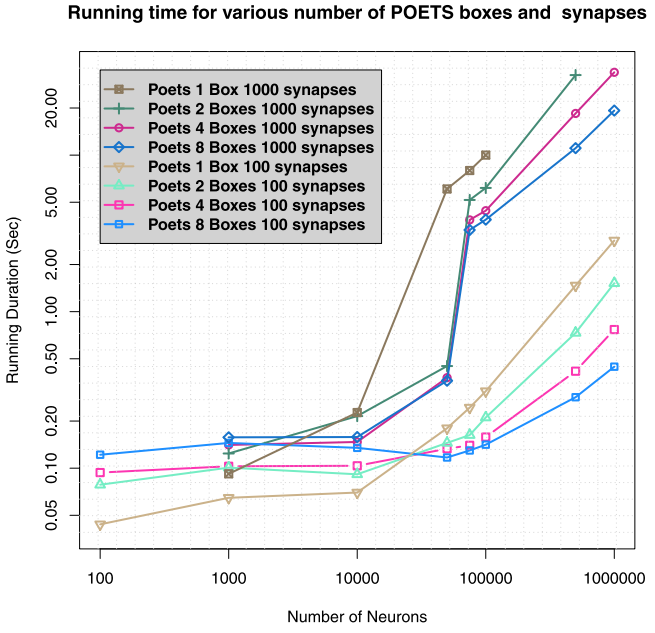


Fig. 9. A comparison of speed of simulation for different number of boxes and two different degree of connections.

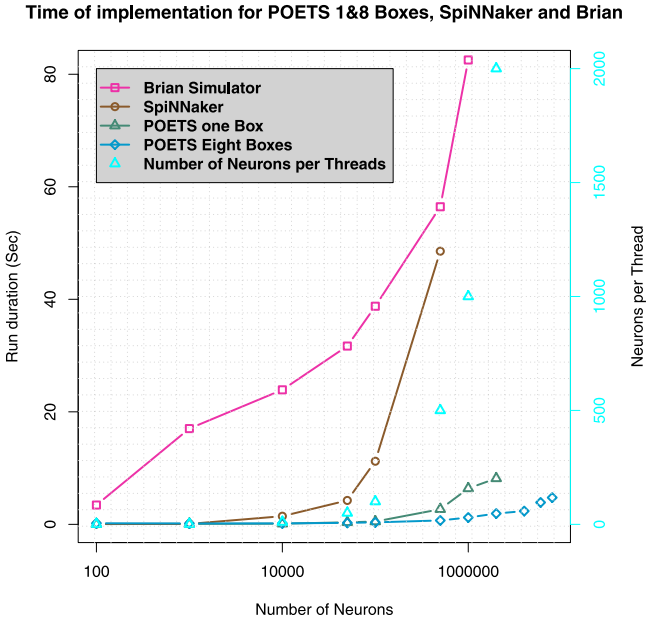


Fig. 10. Speed comparison of implementation among Brian simulator, POETS machine and SpiNNaker.

1. Many-core microprocessor-based approaches where the system reads the instruction codes to model the behavior of neural systems such as SpiNNaker [4] and Loihi [10].
2. Fully digital customized circuits where the neural system components are modeled in circuits using state-of-the-art CMOS technology e.g., IBM TrueNorth machine [11], and POETS.
3. Analog/digital mixed-signal systems that model the behavior of biological neural systems, e.g., the NeuroGrid [5] and BrainScaleS [42] projects.

In the following, we provide some details about well-known large neuromorphic systems.

6.1. SpiNNaker

SpiNNaker is a processor-based system targeting large-scale SNN simulations in real-time. The systems have various sizes consisting of one or more out of 48 chips. Each chip contains 18 fixed-point advanced RISC ARM968 processing cores next to custom routing infrastructure circuits which dedicate 96 kB of local memory besides 128 MB of shared DRAM as depicted in Fig. 11.a. It has a remarkable flexibility, capable of simulating millions of neurons with biologically realistic connectivity supporting learning algorithms. However, the system still uses a von Neumann architecture with a large memory hierarchy as found in conventional computers. Although it uses low-power ARM processors dedicated to power-efficient platforms, the largest machine incorporating over a million ARM processor cores, still requires up to 75 kW of electrical power. The current version has no floating-point support. The improved version named SpiNNaker2 is a 10-million core machine which uses 22 nm scale fabrication technology and supports floating-point operations [43].

6.2. TrueNorth

TrueNorth is a flexible, non-von Neumann fully customized neuromorphic system. TrueNorth uses transistors as digital gates in an

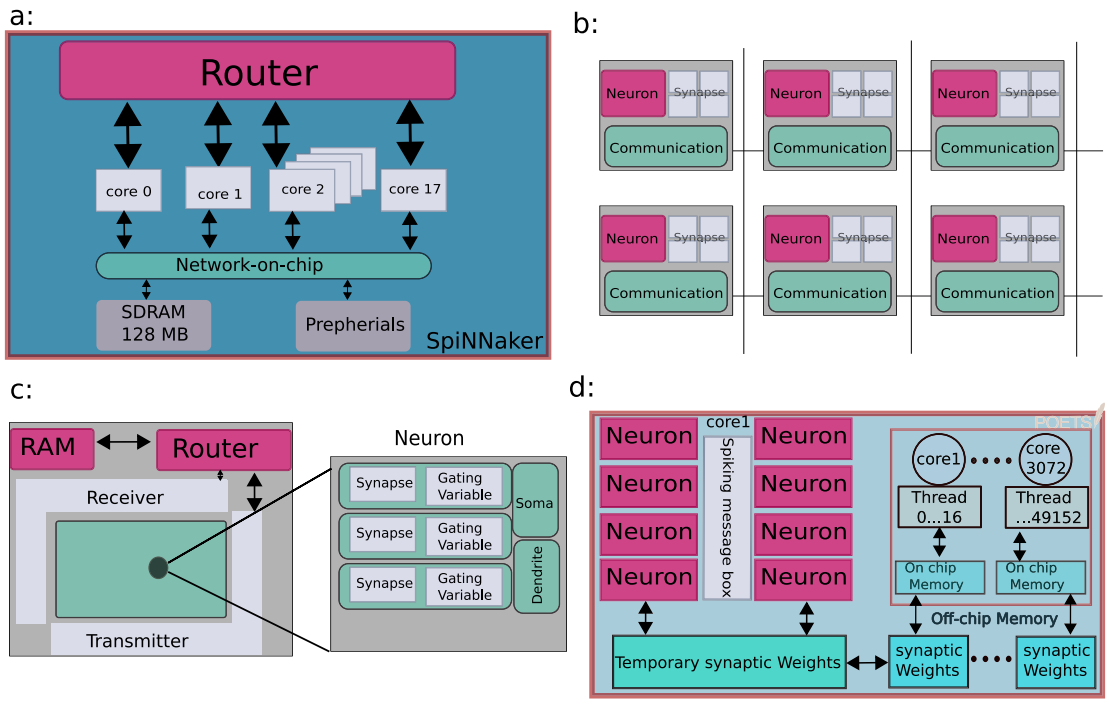


Fig. 11. The way of computation, connection, and communication between neurons assigning to the hardware components in different large-scale spiking neural network systems, **(a)** Principal architectural parts of one node in SpiNNaker, **(b)** TrueNorth, **(c)** NeuroGrid, **(d)** POETS.

event-driven method to communicate in an asynchronous manner. The structure of TrueNorth consists of 5.4 billion transistors to build 4096 neurosynaptic cores. A core includes 256 digital LIF neurons, 256×256 binary synapses, and asynchronous encoding/decoding and routing circuits. Regarding the synaptic matrix, each neuron can be connected to 256 other neurons. The routing in TrueNorth is less flexible than SpiNNaker, however, TrueNorth can distribute the system memory with core synaptic matrix and routing table entries (Fig. 11.b). The power consumption is 20 mW/cm^2, which implements a very power efficient system. In this platform the synapses do not implement any plasticity mechanism, therefore the system has no support for on-chip learning.

6.3. NeuroGrid

NeuroGrid uses analog/digital mixed-signals for modeling neural network components. Similar to TrueNorth, Neurogrid has a non-von Neumann architecture. Neurogrid emulates four neural network components, namely: axon, dendrite, soma and synapse. Axons are modeled by a digital circuit and the other components are modeled in using analog circuits. NeuroGrid consists of 16 standard CMOS "NeuroCores" integrated on a board that works using 3 W of power. The synaptic circuits are shared among the neurons, while different spikes can be assigned to the same synapse (Fig. 11.c). The long time constant limitation (tens of milliseconds) causes difficulties in using typical VLSI for design and implementation.

6.4. BrainScaleS

The BrainScaleS project aims to produce a wafer-scale neural simulation platform, in which each 8 inch silicon wafer integrates 50×10^6 plastic synapses and 200,000 biologically realistic neuron circuits [42]. In order to have a maximal number of processors on the wafer, relatively small capacitors have been used for modeling the synapses and neurons. The speed of network component operations compared to their biological counterparts is accelerated by a factor of 10^3–10^4 which can reduce simulation time dramatically. It needs large bandwidth and fast switching, and still demands a high-power circuit for propagating spikes across the network [44]. Neurogrid and BrainScaleS similarly use the temporal dynamics of memory elements to store the state of the network, with the capability of local learning. The second version of BrainScaleS is developed as BrainScaleS-2. This version is a multi-chip system building upon existing BrainScaleS wafer-scale system components. The architecture is implemented in a single-chip ASIC with an analog core including 512 neurons and 2^{17} synapses, as well as two embedded plasticity and control processors [45]. In contrast to Loihi, which only supports micro-coded operations per synapse or other designs with fixed plasticity, BrainScaleS-2 support plasticity programs with complex control and data dependencies.

6.5. Loihi

Loihi [10] is a recent neuromorphic platform developed by Intel [46]. Loihi is an event-driven neuromorphic chip fabricated in Intel's 14-nm process that uses a discrete-time model for computation distributed over 128 cores that are integrated into an asynchronous mesh. This neuromorphic platform supports variable precision synaptic weights. Each core has 128 kB synaptic state, and 20 kB of routing tables that can be assigned to its 1024 neurons The implementation of 131,072 leaky-integrate-and-fire neurons and more than 130 million synapses has been reported in [10]. Different Loihi systems have been designed by using a multichip mesh architecture, ranging from two chips in a USB device to a rack-mounted system enclosing 768 Loihi chips capable of implementing more than 100 million neurons. Loihi 2 [47] is the next generation of Loihi neuromorphic system which supports simulation of 1 million neuron per chip, programmability of model of neuron, optimizing the energy, latency, and model sizes of intelligent signal processing applications.

6.6. NeuroFlow

NeuroFlow [48] represents another customizable neuromorphic platform capable to simulate scalable SNN. NeuroFlow is an FPGA-based system similar to POETS. This application-specific integrated circuits architecture can be redesigned and reconfigured to suit a

particular SNN simulation to deliver optimized performance. A 6-FPGA Neuroflow system can simulate a network of 600,000 neurons. One FPGA Neuroflow outperforms a speedup of up to 33.6 times the speed of an 8-core processor, as well as 2.83 times the speed of GPU-based platforms. Similar to BrainScaleS and SpiNNaker, it uses PyNN, a simulator-independent neural network description language for the user interface. More details about NeuroFlow are presented in Table 4.

There exist yet other neuromorphic platforms such as Darwin [49], Tianjic [50,51] and Dynap-SEL [52] but we consider their discussion out of the scope of this research.

6.7. Comparing POETS with existing neuromorphic platforms

Neuromorphic platforms can be compared based on various aspects, although delving into such comparisons may exceed the scope of this article. Each neuromorphic platform may have a specific target behavior, either replicating aspects of the brain or catering to AI cognitive computing systems. For instance, SpiNNaker aims to simulate a biological brain or certain parts of it, enabling real-time operation. On the other hand, platforms like Loihi or Darwin, while not designed for real-time processing, are more suitable for applications requiring low-latency and high-speed capabilities. In the context of real-time processing, it is essential to note that POETS, is not specifically designed for real-time operations. Therefore, it would be inappropriate to make direct comparisons to determine one platform's superiority over another, given their distinct purposes for various applications. This article aims to address this gap by conducting a comprehensive comparison of event-driven neuromorphic systems from different perspectives, which has not been explored in existing literature. Specifically, we examine how units or ensembles of units communicate asynchronously within a synchronous (clock-based) architecture and the methods employed for distributing messages among multiple computing units in neuromorphic systems. Fig. 11.d shows POETS hardware architecture next to other neuromorphic architecture to facilitate the comparison.

Synchronization. One of the main features of the event-driven computing system is its asynchronous nature. In a physiological brain, feedback processes provides synchronization and coherent information processing over various neurons populations. In an asynchronous system, to ensure that the message transfers from one time-step to the next one, to be able to debug the system, and check the timing accuracy of the system, we need a degree of synchronization. We have proposed three different synchronization algorithms in this paper. Therefore, POETS is flexible for providing a different level of synchronization, either software or hardware barrier. Loihi uses mesh-level periodic wavefronts of barrier messages signifies if all the messages have reached destinations and the time-step can be advanced to the next cycle. This barrier synchronization process gives permission to chips and cores in a multichip mesh to do computation independently yet, via barrier-mediated handshaking. TrueNorth uses two phases for operation, one phase to propagate the messages and update the neuron state, and the second phase is a synchronization that occurs every millisecond for all cores. SpiNNaker machines lack any form of global synchronization, necessitating individual cores to independently update the state of the neurons they are responsible for simulating. Additionally, these cores must process any incoming spikes they have received within a predetermined simulation time step, usually set to 1 ms. SpiNNaker employs the GALS synchronization algorithm, featuring 18 ARM968 processor nodes organized into synchronous islands. These islands are interconnected by a light-weight, packet-switched asynchronous communications infrastructure. In the TrueNorth architecture, all spikes are processed within the respective neurons, transmitted through their destination axons, and queued within that axon during a predefined time interval known as a 'tick' which serves as a synchronization point, ensuring adherence to a common time reference by all components within a core and across all cores on the chip [53]. In the BrainScaleS, the multi-FPGA orchestration provides parallel configuration and synchronized experiments, facilitating wafer-scale experiments [45]. Neurogrid uses recurrent inhibitory connectivity patterns are expected to give rise to globally synchronous spike activity [5].

Messaging delivery. One of the important characteristics of POETS is a guaranty of message into the destination, which for example SpiNNaker suffers from lack of this messaging guaranty. It means there is no lost message on the fly. SpiNNaker has emergency packet re-routing, which allows packets to be sent along an alternative route when a link is detected as failed. In TrueNorth the chip communicates and processes data packets called spikes. Each core can be connected to any other core on the chip by a two-dimensional mesh network, and therefore messages are delivered from each neuron on the crossbar to any axon on the chip. BrainScaleS uses a cyclic redundancy verification to find corrupted messages. In Loihi, neurons communicate via spike events, 32-bit messages containing destination addressing, source addressing and graded-value payloads to distribute between cores. Each core sends generated spikes to down stream fan-out neurons based on configured routing information that allows adapting to the pipeline activities delay. Neurogrid uses address-event bus to build networks with thousands of neurons with a few hundred synaptic connections that shared wires communicating addresses that signal arrival of an action potential, or spike.

Routing. A distributed system could use source-based routing or destination-based routing. In source-based routing, the message will be routed based on the source of the fire event. In destination-based, the messages will be routed based on the destination to which the fired event is transferring. Source routing is advantageous because multicast routing can be implemented by allowing each router to duplicate a packet based on its own routing table. SpiNNaker, BrainScaleS, Neurogrid, and Darwin all use source-based routing, which for each router is stored in off-chip memory. The disadvantage of source-based routing is that a large off-chip memory is required to restore the routing data for each source. The problem is that accessing to the external memory is slow, as frequent repetition reduced the speed at which packets can be routed. In destination-based routing, the source neuron knows the destination. As there is a limitation for the packet size to send, a limited number of destination elements could be encoded as part of the packet. However, as the message packet has the destination address, there is no need for a large off-chip memory. Both TrueNorth and Loihi use destination-based routing to route packets. POETS uses a combination of destination-based routing and "key" based routing. The "key" is not a single destination, but a pointer to a set of destinations that is expanded by the routing fabric.

Other general characteristics of the large-scale neuromorphic systems, such as scalability, interconnections, feature size, and power consumption, are shown in Table 4.

7. Conclusion

Synchronization is one of the most important methods in designing neuromorphic event-driven systems in simulating SNNs. Three synchronization methods including clocked sync, GALS (Globally Asynchronous Locally Synchronous), and Hardware Idle Detection (HID) have been discussed and analyzed in this work that the HID is introduced in this work as a novel synchronization method for neuromorphic system. To implement these algorithms on hardware, we introduced POETS as a new large-scale neuromorphic system which is flexible using FPGA clusters, easily scalable by adding more FPGA boards, reliable with a guaranty of receiving messages, and fast due to the parallel processing of data and high-speed interconnection bandwidth. Running the SNN on POETS hardware, we demonstrated that HID is the best synchronization approach considering speed and spiking accuracy.

The methods of mapping spiking neurons into different hardware components is another challenge in designing SNN hardware simulation. We pointed that POETS uses efficient communication method with

Table 4
A comparison of the large-scale neuromorphic systems.

Properties	TrueNorth	Neurogrid	BrainScaleS	SpiNNaker	NeuroFlow	Loihi	POETS
Technology	Digital	Analog/digital	Analog/digital	Digital	Digital	Digital	Digital (FPGA)
Feature size	28 nm	180 nm	180 nm	130 nm	28 nm	14 nm	28 nm
Chips	16	16	325	48	6	1	48
Power	3.2 W	3 W	500 W	80 W	40 W	0.1 W	42.8 W
Routing	2D mesh-unicast	Tree-multicast	Hierarchical	2D mesh-unicast	MultiTrack	2D-mesh	2D-mesh-Multicast
Neuron model	Configured LIF	Adaptive IF	Adaptive IF	Programmable	LIF/Izhik	LIF	LIF/Izhik
Neurons	16 M	1 M	200 K	768 K	600 K	130 K	8 M
Synapses	4 G	4 G	40 M	768 M	600 M	130 M	4 G
Interconnect	5.44 Gbit/s	43.4 Mspike/s	32 Gbit/s	7.4 Gbit/s	10 Gbit/s	3.44 Gspike/s	10 Gbit/s
Conn to host	AXI/PCI	USB	Ethernet	Ethernet	Ethernet	USB/Ethernet	Ethernet/FPGA

more communication loads on thread to thread which is faster and more parallel comparing to either board to board or neurons on the same thread.

The routing methods for spiking message-passing has been studied in this work too. Results demonstrate the effectiveness of using multicast method of routing (using the programmable router) over the unicast (point-to-point communication) for moderate number of neurons is more visible rather than small or very large number of neurons. The adaptive threshold has been used in spiking neurons, to make the system more stable and showing better results rather than fixed-firing threshold. Connecting this adaptive threshold to the firing rate, we showed that increasing firing rate has less than linear-slope impact on the speed of simulation.

Using the best parameters e.g., synchronization, communication, and routing, we run SNN on POETS hardware in the range of up to two million spiking neurons on one cluster and up to 8 million spiking neurons on eight clusters. Regarding the number of synaptic connections in SNN modeling, up to one billion on one cluster and up to 4 billion synaptic connections on all eight clusters are the maximum numbers that have been modeled in this work. A speed comparison demonstrates that POETS simulates the SNN 20 time faster than Brian simulator and 16 times faster than SpiNNaker using the best approaches for synchronization and communication. This paper provides an architecture overview of current large-scale neuromorphic systems, with a primary focus on investigating synchronization, communication, and routing methods employed in these systems. The future work could be a next generation of this neuromorphic hardware named POETS-II that will use more recent FPGAs including the advancement of hardware overlay beyond Tinsel and APIs that could be available online for different learning algorithms and facilitating data streaming for training.

CRediT authorship contribution statement

Mahyar Shahsavari: Writing – original draft, Methodology, Software. **David Thomas:** Writing – original draft, Revision, Software. **Marcel van Gerven:** Revision, Writing – review & editing. **Andrew Brown:** Revision, Supervision. **Wayne Luk:** Revision, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported partially by EPSRC/UK as POETS project EP/N031768/1, and M.S. is supported by Spikeference project, Human Brain Project SGA3 (ID: 945539).

References

[1] Mead C. Neuromorphic electronic systems. Proc IEEE 1990;78:1629–36.
[2] Liu S-C, Delbruck T, Indiveri G, Whatley ARD. Event-based neuromorphic systems. Wiley; 2014.
[3] Mostafa H, Müller LK, Indiveri G. An event-based architecture for solving constraint satisfaction problems. Nature Commun 2015-12-08;6(1):8941. http://dx.doi.org/10.1038/ncomms9941.
[4] Furber SB, Galluppi F, Temple S, Plana LA. The SpiNNaker project. Proc IEEE 2014;102(5):652–65. http://dx.doi.org/10.1109/JPROC.2014.2304638.
[5] Benjamin BV, Gao P, McQuinn E, Choudhary S, Chandrasekaran AR, Bussat JM, et al. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. Proc IEEE 2014;102(5):699–716. http://dx.doi.org/10.1109/JPROC.2014.2313565.
[6] Tavanaei A, Ghodrati M, Kheradpisheh SR, Masquelier T, Maida A. Deep learning in spiking neural networks. Neural Netw 2019;111:47–63. http://dx.doi.org/10.1016/j.neunet.2018.12.002.
[7] Schuman CD, Kulkarni SR, Parsa M, Mitchell JP, Date P, Kay B. Opportunities for neuromorphic computing algorithms and applications. Nature Comput Sci 2022;2(1):10–9. http://dx.doi.org/10.1038/s43588-021-00184-y.
[8] Walravens M, Verreyken E, Steckel J. Spiking neural network implementation on FPGA for robotic behaviour. In: Lecture notes in networks and systems, vol. 96, 2020, p. 694–703. http://dx.doi.org/10.1007/978-3-030-33509-0_65.
[9] Zhu Y, Zhu Y, Mao H, He Y, Jiang S, Zhu L, et al. Recent advances in emerging neuromorphic computing and perception devices. J Phys D: Appl Phys 2021;55(5):053002. http://dx.doi.org/10.1088/1361-6463/ac2868.
[10] Davies M, Srinivasa N, Lin T-H, Chinya G, Cao Y, Choday SH, et al. Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro 2018;38(1):82–99. http://dx.doi.org/10.1109/MM.2018.112130359, Conference Name: IEEE Micro.
[11] Merolla PA, Arthur JV, Alvarez-Icaza R, Cassidy AS, Sawada J, Akopyan F, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 2014;345(6197):668–73. http://dx.doi.org/10.1126/science.1254642.
[12] Pfeiffer M, Pfeil T. Deep learning with spiking neurons: Opportunities and challenges. Front Neurosci 2018;12:774. http://dx.doi.org/10.3389/fnins.2018.00774.
[13] Rast A, Shahsavari M, Bragg GM, Vousden ML, Thomas D, Brown A. A hardware/application overlay model for large-scale neuromorphic simulation. In: 2020 international joint conference on neural networks. 2020, p. 1–9. http://dx.doi.org/10.1109/IJCNN48605.2020.9206708, ISSN: 2161-4407.
[14] Shahsavari M, Devienne P, Boulet P. N2S3, a Simulator for the architecture exploration of neuromorphic accelerators. 2015, URL https://hal.archives-ouvertes.fr/hal-01240444.
[15] Brette R, et al. Simulation of networks of spiking neurons: A review of tools and strategies. J Comput Neurosci 2007;23(3):349–98, 2007. http://dx.doi.org/10.1007/s10827-007-0038-6.
[16] Izhikevich EM. Simple model of spiking neurons. Trans Neur Netw 2003;14(6):1569–72. http://dx.doi.org/10.1109/TNN.2003.820440.
[17] Maass W, Bishop CM, editors. Pulsed neural networks. Cambridge, MA, USA: MIT Press; 1999.
[18] Brown A, Vousden M, Rast A, Bragg G, Thomas D, Beaumont J, et al. POETS: Distributed event-based computing - scaling behaviour. Adv Parallel Comput 2020;36:487–96. http://dx.doi.org/10.3233/APC200076, cited By 0.
[19] Shahsavari M, Thomas D, Brown A, Luk W. Neuromorphic design using reward-based STDP learning on event-based reconfigurable cluster architecture. In: International conference on neuromorphic systems 2021. ICONS 2021, New York, NY, USA: Association for Computing Machinery; 2021, p. 1–8. http://dx.doi.org/10.1145/3477145.3477151.
[20] Shahsavari M, Beaumont J, Thomas D, Brown AD. POETS: A parallel cluster architecture for spiking neural network. Int J Mach Learn Comput 2021;11(4):281–5. http://dx.doi.org/10.18178/ijmlc.2021.11.4.1048.

[21] Rafiev A, Morris J, Xia F, Yakovlev A, Naylor M, Moore S, et al. Practical distributed implementation of very large scale Petri net simulations. In: Koutny M, Kordon F, Moldt D, editors. Transactions on petri nets and other models of concurrency XVI. Berlin, Heidelberg: Springer Berlin Heidelberg; 2022, p. 112–39. http://dx.doi.org/10.1007/978-3-662-65303-6_6.

[22] Morris J, Rafiev A, Xia F, Shafik R, Yakovlev A, Brown A. An alternate feedback mechanism for tsetlin machines on parallel architectures. In: 2022 international symposium on the tsetlin machine. 2022, p. 53–6. http://dx.doi.org/10.1109/ISTM54910.2022.00018.

[23] Brown AD, Beaumont JR, Thomas DB, Shillcock JC, Naylor MF, Bragg GM, et al. POETS: An event-driven approach to dissipative particle dynamics: Implementing a massively compute-intensive problem on a novel hard/software architecture. ACM Trans Parallel Comput 2023;10(2). http://dx.doi.org/10.1145/3580372.

[24] Cheung K, Schultz SR, Luk W. A large-scale spiking neural network accelerator for FPGA systems. In: Villa AEP, Duch W, Érdi P, Masulli F, Palm G, editors. Artificial neural networks and machine learning. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012, p. 113–20.

[25] Thomas D, Luk W. FPGA accelerated simulation of biologically plausible spiking neural networks. In: Proceedings of the 2009 17th IEEE symposium on field programmable custom computing machines. USA: IEEE Computer Society; 2009, p. 45–52. http://dx.doi.org/10.1109/FCCM.2009.46.

[26] Pani D, Meloni P, Tuveri G, Palumbo F, Massobrio P, Raffo L. An FPGA platform for real-time simulation of spiking neuronal networks. Front Neurosci 2017;11. http://dx.doi.org/10.3389/fnins.2017.00090.

[27] Vousden M, Morris J, McLachlan Bragg G, Beaumont J, Rafiev A, Luk W, et al. Event-based high throughput computing: A series of case studies on a massively parallel softcore machine. IET Comput Digit Tech 2023;17(1):29–42. http://dx.doi.org/10.1049/cdt2.12051.

[28] Naylor M, Moore SW, Thomas D. Tinsel: A manythread overlay for FPGA clusters. In: 2019 29th international conference on field programmable logic and applications. 2019, p. 375–83. http://dx.doi.org/10.1109/FPL.2019.00066.

[29] Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, et al. Pregel: A system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. New York, NY, USA: ACM; 2010, p. 135–46. http://dx.doi.org/10.1145/1807167.1807184.

[30] Zhou S, Kannan R, Zeng H, Prasanna VK. An FPGA framework for edge-centric graph processing. In: Proceedings of the 15th ACM international conference on computing frontiers. New York, NY, USA: Association for Computing Machinery; 2018, p. 69–77. http://dx.doi.org/10.1145/3203217.3203233.

[31] Jordan MI, Mitchell TM. Machine learning: Trends, perspectives, and prospects. Science 2015;349:255–60. http://dx.doi.org/10.1126/science.aaa8415.

[32] Lecun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521:436–44. http://dx.doi.org/10.1038/nature14539.

[33] Zenke F, Neftci EO. Brain-inspired learning on neuromorphic substrates. Proc IEEE 2021;109(5):935–50. http://dx.doi.org/10.1109/JPROC.2020.3045625.

[34] Wang X, Lin X, Dang X. A delay learning algorithm based on spike train kernels for spiking neurons. Front Neurosci 2019;13:252. http://dx.doi.org/10.3389/fnins.2019.00252.

[35] Muttersbach J, Villiger T, Kaeslin H, Felber N, Fichtner W. Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems. In: Twelfth Annual IEEE international ASIC/SOC conference (Cat. No.99TH8454). 1999, p. 317–21. http://dx.doi.org/10.1109/ASIC.1999.806526.

[36] Gagne R, Belzile J, Thibeault C. Asynchronous component implementation methodology for GALS design in FPGAs. In: 2009 Joint IEEE North-east workshop on circuits and systems and TAISA conference. 2009, p. 1–4. http://dx.doi.org/10.1109/NEWCAS.2009.5290499.

[37] Naylor M, Moore SW, Mokhov A, Thomas D, Beaumont JR, Fleming S, et al. Termination detection for fine-grained message-passing architectures. In: 2020 IEEE 31st international conference on application-specific systems, architectures and processors. 2020, p. 17–24. http://dx.doi.org/10.1109/ASAP49362.2020.00012.

[38] Trik M, Mozaffari SP, Bidgoli AM. Providing an adaptive routing along with a hybrid selection strategy to increase efficiency in noc-based neuromorphic systems. Comput Intell Neurosci 2021;2021. URL https://api.semanticscholar.org/CorpusID:237614646.

[39] Stimberg M, Brette R, Goodman DF. Brian 2, an intuitive and efficient neural simulator. In: Skinner FK, editor. eLife 2019;8:e47314. http://dx.doi.org/10.7554/eLife.47314.

[40] Naylor M, Moore SW, Thomas D, Beaumont JR, Fleming S, Vousden M, et al. General hardware multicasting for fine-grained message-passing architectures. In: 2021 29th Euromicro international conference on parallel, distributed and network-based processing. 2021, p. 126–33. http://dx.doi.org/10.1109/PDP52278.2021.00028.

[41] Marder E, Goaillard J-M. Variability, compensation and homeostasis in neuron and network function. Nat Rev Neurosci 2006;7(7):563–74. http://dx.doi.org/10.1038/nrn1949, URL http://www.nature.com/nrn/journal/v7/n7/full/nrn1949.html.

[42] Schemmel J, Briiderle D, Griibl A, Hock M, Meier K, Millner S. A wafer-scale neuromorphic hardware system for large-scale neural modeling. IEEE; 2010, p. 1947–50. http://dx.doi.org/10.1109/ISCAS.2010.5536970.

[43] Mayr C, Höppner S, Furber S. SpiNNaker 2: A 10 million core processor system for brain simulation and machine learning. Concurr Syst Eng Ser 2019;70:277–80. http://dx.doi.org/10.3233/978-1-61499-949-2-277, cited By 0.

[44] Indiveri G, Liu S-C. Memory and information processing in neuromorphic systems. Proc IEEE 2015;103(8):1379–97. http://dx.doi.org/10.1109/JPROC.2015.2444094, arXiv: 1506.03264.

[45] Pehle C, Billaudelle S, Cramer B, Kaiser J, Schreiber K, Stradmann Y, et al. The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity. Front Neurosci 2022;16. http://dx.doi.org/10.3389/fnins.2022.795876, URL https://www.frontiersin.org/articles/10.3389/fnins.2022.795876.

[46] Davies M, Wild A, Orchard G, Sandamirskaya Y, Guerra GAF, Joshi P, et al. Advancing neuromorphic computing with loihi: A survey of results and outlook. Proc IEEE 2021;109(5):911–34. http://dx.doi.org/10.1109/JPROC.2021.3067593.

[47] Orchard G, Frady EP, Rubin DBD, Sanborn S, Shrestha SB, Sommer FT, et al. Efficient neuromorphic signal processing with loihi 2. 2021, arXiv:2111.03746.

[48] Cheung K, Schultz SR, Luk W. NeuroFlow: A general purpose spiking neural network simulation platform using customizable processors. Front Neurosci 2016;9:516. http://dx.doi.org/10.3389/fnins.2015.00516.

[49] Ma D, Shen J, Gu Z, Zhang M, Zhu X, Xu X, et al. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. J Syst Archit 2017;77:43–51. http://dx.doi.org/10.1016/j.sysarc.2017.01.003.

[50] Pei J, Deng L, Song S, Zhao M, Zhang Y, Wu S, et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. Nature 2019;572(7767):106–11. http://dx.doi.org/10.1038/s41586-019-1424-8.

[51] Wu Y, Zhao R, Zhu J, Chen F, Xu M, Li G, et al. Brain-inspired global-local learning incorporated with neuromorphic computing. Nature Commun 2022;13(1):65. http://dx.doi.org/10.1038/s41467-021-27653-2.

[52] Moradi S, Qiao N, Stefanini F, Indiveri G. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). IEEE Trans Biomed Circuits Syst 2018;12(1):106–22. http://dx.doi.org/10.1109/TBCAS.2017.2759700, Conference Name: IEEE Transactions on Biomedical Circuits and Systems.

[53] Carney R, Bouchard K, Calafiura P, Clark D, Donofrio D, Garcia-Sciveres M, et al. Neuromorphic Kalman filter implementation in IBM's TrueNorth. J Phys Conf Ser 2017;898(4):042021. http://dx.doi.org/10.1088/1742-6596/898/4/042021.