Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture

Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk

► To cite this version:

Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk. Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture. International Conference on Neuromorphic Systems (ICONS), Jul 2021, Oak Ridge (virtual), United States. hal-03270518

HAL Id: hal-03270518

<https://hal.science/hal-03270518>

Submitted on 24 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture

Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk

► To cite this version:

Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk. Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture. International Conference on Neuromorphic Systems (ICONS), Jul 2021, Oak Ridge, TN, United States. hal-03270518

HAL Id: hal-03270518

<https://hal.archives-ouvertes.fr/hal-03270518>

Submitted on 24 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture

Mahyar Shahsavari
m.shahsavari@imperial.ac.uk
Imperial College London
London, UK

Andrew Brown
adb@ecs.soton.ac.uk
University of Southampton
Southampton, UK

David Thomas
d.thomas1@imperial.ac.uk
Imperial College London
London, UK

Wayne Luk
w.luk@imperial.ac.uk
Imperial College London
London, UK

ABSTRACT

Neuromorphic computing systems simulate spiking neural networks that are used for research into how biological neural networks function, as well as for applied engineering such as robotics, pattern recognition, and machine learning. In this paper, we present a neuromorphic system based on an asynchronous event-based hardware platform. We represent three algorithms for implementing spiking networks on our asynchronous hardware platform. We also discuss different trade-offs between synchronisation and messaging costs. A reinforcement learning method known as Reward-modulated STDP is presented as an online learning algorithm in the network. We evaluate the system performance in a single box of our designed architecture using 6000 concurrent hardware threads and demonstrate scaling to networks with up to 2 million neurons and 400 million synapses. The performance of our architecture is also compared to existing neuromorphic platforms, showing a 20 times speed-up over the Brian simulator on an x86 machine, and a 16 times speed-up over a 48-chip SpiNNaker node.

KEYWORDS

Neuromorphic system, Spiking neural network simulation, Reinforcement Reward-modulated STDP, Reconfigurable architecture.

ACM Reference Format:

Mahyar Shahsavari, David Thomas, Andrew Brown, and Wayne Luk. 2021. Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture. In *Proceedings of ACM Conference (ICONS 2021)*. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

A neuromorphic platform inspired by the biological brain performs computation via the exchange of spikes between large numbers

of simulated neurons. Information is encoded in the timing of relatively infrequent and unpredictable spiking events, unlike traditional artificial neural networks (ANNs), which pass information as numeric values between neurons on a regular schedule.

Neuromorphic computing has great potential as a computing platform due to the capability for low-power and high-performance computation, particularly in the areas of inference and control. For example, they have been used as an accelerator for cognitive computation algorithms running on robotic brains [23], or future low-power mobile processors [22]. In both application areas, a key concern is the efficiency and performance of simulating large numbers of spiking neurons at high speed. Neuromorphic compute platforms have been constructed from standard multi-core CPUs/GPUs, but there has also been a great deal of research into specialised hardware platforms. These platforms are designed to both exploit the potential for low-power computation from specialised hardware, while also managing the challenge of large-scale spike distribution. Technologies used include reconfigurable digital hardware such as FPGAs [3, 23], custom digital ASIC solutions [7], and even full-custom analogue ASIC systems [12]. In specialised hardware systems such as FPGAs, in addition to the programmability and flexibility, they can exploit the high temporal and spatial sparsity that can be found in the computational data flow of information which is difficult to be exploited by CPU/GPUs [15].

Designing a generic algorithm to manage how information communicates between units is one of the most important challenge for computation on spiking neuromorphic chips. Two main techniques are traditionally used for simulating the spiking neural units on hardware platforms are event-driven or asynchronous and clock-driven or synchronous [2]. An event-triggered system needs time-steps to give a guarantee that neurons have enough time to generate spikes as well as there is enough time for spikes to reach destinations. It means a synchronisation method to be implemented on hardware components is unavoidable even in event-triggered systems. To the best of our knowledge, in this work for the first time, we evaluate different computation algorithms for event-based asynchronous neuromorphic platforms. The key advantage of using event-driven asynchronous is the potential in speeding up the computation and propagation due to not iterating many small update steps for a neuron in which no event arrives.

Biological brain's reward/punishment system by injecting dopamine as a neuromodulator, plays a critical role in decision-making and

Figure 1: Mapping the neural nodes and synaptic edges to the different hardware components in POETS.

connection weight modification [21]. For training the network, we use a reinforcement learning (RL) version of spike-timing-dependent plasticity (STDP) known as reward-modulated STDP (R-STDP) [10].

In this work, we use a partially ordered event-triggered system (POETS) platform that computation and communication happen through sending and receiving events occurring in thousands of concurrent hardware threads [16, 18]. Figure 1 shows how neurons are assigned into our hardware threads to communicate via message box and using on-chip memory for temporary synaptic weights, and off-chip memory for calculated weights.

Briefly, our main contributions in this research are:

- mapping spiking neural network (SNN) onto our architecture, using two tool-chains to compile networks into hardware descriptions;
- evaluation of three different synchronisation algorithms for event-driven asynchronous hardware;
- increasing the communication speed using a new version of hardware overlay which supports multicasting;
- implementing reward-modulated STDP as a learning language in the network;
- a qualitative comparison between POETS and other neuromorphic platforms, showing that a 6000 thread in one box is 20 times faster than Brian, and at least 16 times faster than a 48 chips SpiNNaker node.

2 NEUROMORPHIC HARDWARE

From an architectural point of view, we can identify three main approaches to create neuromorphic hardware platforms:

- (1) Many-core microprocessor-based approaches where the system uses software instructions running in a hardware communications fabric to model the behavior of neural systems, such as the SpiNNaker machine [7], Loihi [5].
- (2) Digital custom circuits where the neural system components are modeled as a circuit in an ASIC, such as the IBM TrueNorth machine [12].
- (3) Analogue/digital mixed-signal systems that use analogue components to model neurons directly, such as NeuroGrid [1] and BrainScales [17] projects.

Neuromorphic platforms intend to simulate a spiking neural network (SNN) from thousands to millions of neurons with varying degrees of connectivity. For example, a SpiNNaker system is built out of one or more 48-chip boards can simulate millions of neurons and synapses. Another well-known neuromorphic platform is TrueNorth which supports feed-forward, recurrent, and lateral connections, however, it does not implement any on-chip learning and it only uses pre-trained networks. The speed of network components in SNN architectures is higher compared to biological elements. For instance, BrainScales accelerates the computation by a factor of 10^3 to 10^4 compared to biological NN. NeuroGrid developed at Stanford University uses analogue/digital mixed-signal to model continuous-time for neural network components. There are yet other neuromorphic platforms such as Loihi [5], NeuroFlow [4], and Darwin [11], of which currently Loihi attracts more SNN applications in the community. Loihi is fabricated in Intel’s 14-nm process, and supports up to 130,000 artificial CUBA leaky-integrate-and-fire neurons and 130 million synapses, on up to 4,096 on-chip cores.

2.1 Event-based compute platform

The hardware platform designed in our team can support the event-driven parallel programming model which is capable to be used as a neuromorphic architecture. It uses a large number of simple cores. This platform is developed for parallel computing based on a graph abstraction, so any given problem to be solved must first be decomposed into a graph. Vertices in the graph represent finite-state machines capturing the state and computation, while edges are static paths over which events can flow between state machines. Computation happens whenever a message is sent from or received at a vertex – these messages are called “events”, as they both contain the data being transmitted, as well as coordinating computation within the graph.

This event-driven process is completely asynchronous and concurrent. Each state-machine will be updated independently, and there is no shared state within the system. Only one event will occur at any vertex at any time, which is guaranteed in this system. It means once a message has been sent it will be eventually delivered. However, there are no other guarantees on the ordering of message arrival.

Figure 2: a) One hardware box, each box has 7 FPGAs: 6 are workers, and one is used to communicate with an x86 machine. b) The default configuration of the Tinsel (NoC) on a single FPGA. c) Inside Tinsel, the cores are RISC-V processors supporting 16 threads.

2.2 Hardware architecture

This hardware is designed in terms of “boxes”, which each box containing six FPGAs, and supporting 6144 partially independent RISC-V hardware threads. The six FPGAs in each box are located on DE5-Net boards, with each board having $2 \times 4GB$ DDR3 DRAMs, $4 \times 8MB$ QDRII+ SRAMs, and $4 \times 10G$ SFP+ ports. An additional non-computational FPGA is used as a PCIe to SFP+ bridge board, providing a fast connection between the x86 and remaining six worker FPGAs (Figure 2.a). The system is designed to scale to multiple co-operating boxes but in this paper, we focus on just a single box.

The computation and communication functions are implemented using a bespoke CPU and network overlay called *Tinsel* [14]. Tinsel is a highly-parameterised design that allows for different configurations to balance compute versus network capability, but here we will assume the default configuration of the overlay running on our current cluster. Tinsel uses a 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 contains 16 tiles (Figure 2.b). A tile, depicted in Figure 2.c, contains the following components.

Core. Each tile contains 4 RISC-V cores, with 16 threads per core, so in total each FPGA has 1024 threads.

FPU. Each tile shares a 32-bit FPU (floating-point unit) between four cores, which provides a balance between floating-point throughput and area consumed.

Mailbox. The Mailbox is used to send and receive messages between threads, providing a message space of 1KB. Mailboxes are connected together to establish a distributed network, through which a thread can send a uni-cast message to any other thread recently multicasting is supported too.

partitioned data cache. The data cache is an 8-way set-associative write-back used to access off-chip memory on each DE5. There is no support for communication via shared memory, so the threads are non-coherent.

2.3 Software API overview

POETS is designed to support a wide variety of applications, with the current portfolio including dissipative particle dynamics, earthquake modelling, biological network analysis, and genetic analysis. To avoid application developers needing to deal with the low-level aspects of programming a bare-metal RISC-V system, a shared graph abstraction is used to describe applications. Developers express their problems in terms of state machines (vertices) and topology information (edges) at a high-level, and then a shared tool-chain is used to map it into the hardware. So in our current target application, the developer needs to work out how to express a spiking neural network as state machines, then it will get compiled and mapped automatically.

Currently, there are two different tool-chains for mapping the models and applications onto hardware, Graph Schema and POLite.

Graph Schema. This tool transforms the abstract representation of graphs into an XML specification containing a GraphType and a GraphInstance sections. The GraphType section defines the types of messages and the functional behaviour of each state machine, while the GraphInstance section instantiates state machines and connects them together.

POLite. Another high-level programming environment that our team has developed is POLite. It is similar to the vertex-centric paradigm [25], but supports both synchronous and asynchronous messaging. POLite is a lightweight C++ software layer built on top of the low-level Tinsel hardware API, designed to support the exploration of a graph-based event-driven abstraction, while also hiding architectural details from users. Figure 3 shows the steps from high-level modeling down to hardware mapping when using POLite toolchain.

3 SPIKING NEURAL NETWORKS ON EVENT-BASED HARDWARE

POETS supports applications that are described as graphs of event-based state machines, but does not provide any specific application support. As a consequence, while SNNs should be a good fit for implementation on this platform, there is no direct support for them. For example, POETS provides no notion of system-wide time or real-time clocks, nor does it provides ordering or synchronisation primitives, both of which are needed for an SNN.

3.1 Spiking neural network in POETS

The form of SNN we consider is the discrete-time evolution of a network of neurons through a set of time-steps. At each time-step, each neuron is updated, using its current state and the weighted sum of input spikes from the previous time-step as input, and producing a new neuron state. We can capture the neuron model and topology as follows:

- A set of n neuron states : $s_1 \dots s_n$
- An $n \times n$ weight matrix : W
- A neuron update function : $u(s, I) \rightarrow (s', f')$

Figure 3: Software-hardware co-design from high-level user programming to low-level hardware FPGA implementation of SNN.

The neuron update function u consumes the current neuron state plus the weighted sum of spikes from the previous time-step:

$$(s_i^{t+1}, f_i^{t+1}) = u \left(s_i^t, \sum_{j=1}^n W[i, j] f_j^t \right). \quad (1)$$

This system is easy to describe as a dense matrix-vector multiplication for small n , or even as a sparse matrix-vector multiplication for medium n , but eventually the sparsity and unpredictable nature of the firing vector f makes this inefficient.

In Hardware, we map each neuron n_i into its own independent state machine. When spikes arrive at the neuron, they cause a receive event, which is used to accumulate the spike into the running stimulus total. If s_i is the neuronal state of neuron i , we use S_i to describe the corresponding hardware state, which will also contain extra information related to synchronisation. At the simplest, we could define $S_i = (s_i, I_i)$, where the new variable I_i represents the sum of the incoming stimulus.

A node unit sends and receives events to implement the SNN behaviour as:

- Send : $(s_i, msg.f) = u(s_i, I_i); msg.src = i; I_i = 0$
- Recv : $I_i = I_i + W[i, msg.src] \times msg.f$

where msg represents the message being sent or received, contains a flag indicating whether it fired or not and the source neuron. The weight $W[i, msg.src]$ is actually encoded as an edge weight on this static graph, and so 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.

Figure 4: A finite state machine that can be applied for three different synchronisation methods in event-driven networks.

The actual neuronal model in this work is the Izhikevich model, which combines the physiological plausibility of Hodgkin–Huxley model and the computational efficiency of integrate-and-fire neuron [8]. In this model, the state of each neuron is split into two real variables $u(t)$ and $v(t)$, with the following update function:

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

$$\frac{du(t)}{dt} = a(bv - u) \quad (3)$$

where the parameters a to d remain the same as were presented in [8]. If a neuron reaches its voltage threshold of membrane potential, it fires and then resets:

$$v(v > v_{th}) = c \text{ and } u(v > v_{th}) = u + d. \quad (4)$$

3.2 Different algorithms for SNN in Event-based platforms

Now we need to response to the important problem of synchronisation: *how do we ensure do not execute the Send handler on a neuron until all messages from the previous time-step have been processed?* This is particularly important in our hardware, as it does not guarantee the arrival order of messages. We address this issue using three various algorithms which attempt to build a sort of synchronisation using state machines.

Clocked synchronous algorithm In this method, a clock device is defined for each neuron that not only internally changes the states of the neuron (receive, compute, fire/reset) but also synchronises messaging with other neighbor neurons in the network. This method avoids the issue where a neuron to stay behind another neuron. In each state-machine clock device has to send a message to every other neurons in each time-step, regardless of whether

it spikes or not. This is the disadvantage of using a clocked synchronous algorithm. There is a trade-off here between accuracy versus speed, and in this case, we choose accuracy. For asynchronous hardware like our platform, this method is not recommended as it returns the full synchronisation drawback to the systems. Thus this method is recommended for small to medium size of networks (See Figure 4).

GALS The globally asynchronous, locally synchronous method allows a collection of locally clocked modules to communicate with each other and ensures those neurons that are directly connected cannot get more than one step ahead or behind. A neuron can only progress to the next time-step if it has received messages equal to the number of input connections. In the GALS technique, there is no requirement of a global clock nor bi-directional connections, unlike the clocked sync method, however, incoming spikes are stored in a list. There are two sets of counters and stimulus accumulators at each neuron: one set tracks the number of spikes and total stimulus accumulated in the *current* time-step, while another tracks the spikes and stimulus for the *next* time-step. As there is no global synchronisation between the neurons in the system, we predict increasing in the speed of computation and communication in the neuromorphic system. The state machine and a simple GALS-based network are shown in the middle of Figure 4.

Hardware idle detection (HID) In this method, no neuron waits for all other neurons to receive messages but it should meet hardware barrier synchronisation. 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 keeps the neurons synchronised. This version of message communication is simple as it will be detected by hardware which makes it faster than clocked versions. However, we need to provide a handler that is called for all nodes. Therefore, obviously for any node it takes longer time to reach the barrier if there are more messages to send or receive. It will enforce other nodes to sit idle. The state machine of HID method and network configuration is depicted in Figure 4.

3.3 Unicast versus Multicast threads communication

Threads in this hardware communicate with each other via mailboxes. However, two different methods have been used for message communication between threads namely Unicast and multicast. In the unicast method, there is one-to-one communication between two threads while in multicast one thread could communicate to several other threads. Tinsel provides a programmable router on each FPGA board to support global multicasting. Programmable routers automatically communicate to destination threads distributed throughout the cluster, minimising interconnection bandwidth usage while offloading work from the processing cores. We predict using multicasting the network speeds up messaging communication.

4 ONLINE LEARNING IN NEUROMORPHIC HARDWARE

One of the advantages of using neuromorphic systems over conventional computing systems such as CPUs, GPUs is the real-time inference capability [24]. Online on-chip learning requires to be

realized based on asynchronous inputs and event-based data flow which are the properties of our hardware as a neuromorphic architecture. Implementing on-chip learning in spiking neural networks to interact in real-time with the real-world via analyzing sensory data which is meaningful to bridge the gap between neuroscience and machine learning. Using state-of-the-art inference methods such as CNN (Convolution Neural Network) [9] or DBN (Deep Belief Network) [6] on spiking neuromorphic platforms are not very promising due to the non-local learning nature of gradient descent back-propagation as well as lack of suitable spiking data resources. In addition, if we could even conquer the non-locality learning problem to solve big matrices using gradient descent, the accuracy in the spiking version is yet less than learning algorithms in ANN working with continued data. Furthermore, using the back-propagation algorithm is not enough fast method for online learning in real-time systems which the latency is a very critical item.

Local learning methods such as Spike-timing dependent plasticity (STDP) not only is suitable for SNN using local synaptic weight plasticity, but also it is fast enough to be adapted to real-time computation systems. However, the drawback of using STDP in SNN is low accuracy compared to the ANN learning algorithm. Therefore mixing the plasticity-based learning with reinforcement learning (RL), we offer Reward-Modulated STDP (R-STDP) to improve the accuracy yet take the advantages of the local learning algorithm. This method is named semi-supervised as the answer for classification will be checked with the fired neuron in the last layer of the neural network. If the answer is correct then the reward signal will be applied for the STDP to improve the connection weights to the fired neuron. Consequently, if the answer of the fired neuron is not as same as the real answer, the punishment signal will be used. In other words, the anti-STDP is used, that persuade the neuron to learn something else. To realize the Reward-Modulated STDP, the winner-take-all (WTA) competition method will be applied for neurons at the last layer to choose the fired neuron to compare to the real answer. To formulate R-STDP learning rule, we use a similar method represented in [13], if a reward signal is received, then

$$\Delta W_{ji} = \begin{cases} \alpha_{rw}^p \times W_{ji} \times (1 - W_{ji}) & \text{if } \Delta t \geq 0 \\ \alpha_{rw}^n \times W_{ji} \times (1 - W_{ji}) & \text{if } \Delta t < 0 \end{cases} \quad (5)$$

and in case of receiving a punishment signal, we have

$$\Delta W_{ji} = \begin{cases} \alpha_{pu}^p \times W_{ji} \times (1 - W_{ji}) & \text{if } \Delta t < 0 \\ \alpha_{pu}^n \times W_{ji} \times (1 - W_{ji}) & \text{if } \Delta t \geq 0 \end{cases} \quad (6)$$

where i and j refer to the pre- and postsynaptic neurons, respectively, ΔW_{ji} presents the changing in weight between neuron i and j , α_{rw}^p , α_{rw}^n , α_{pu}^p and α_{pu}^n are factor values of reward or punishment for increasing or decreasing the amount of ΔW_{ji} . In fact, in the R-STDP learning rule, the reward signal supports more learning in both positive and negative directions, and the punishment signal reverses the polarity of STDP known as anti-STDP encouraging to learn something else (see Figure 5.a).

Figure 5: a) Changing the weights between pre-and postsynaptic neurons in R-STDP learning in the condition of existing reward or punishment signal, b) preparation of MNIST dataset converting the pixel intensity to the input spikes.

5 EXPERIMENTAL RESULTS

In this work, we consider the capabilities of the system in a single hardware box, containing 6 FPGAs and 6144 threads, allowing us to implement up to 2 million neurons. FPGAs contain DE5-NET boards (Stratix-5SGXEA7N2F45C2) running at 250 MHz, and connected to an Intel(R) i9-7940X CPU@3.10GHz with 28 cores. We set up two various experimental validation tests to verify first, the inference and second, the scalability of simulation of large SNNs.

Inference setup: To demonstrate the inference capability of our platform, we used the MNIST dataset of handwritten digits to train the spiking neural networks. The training set consists of 60000 digits between 0 to 9, and each handwritten number is a 28×28 pixels image. The asymptotic representation in Figure 5.b shows each pixel is connected to one input buffer neuron. We transferred the pixel intensity into spikes. The intensity between 0 to 255 for each pixel is transferred to 0 to 22 spikes while presenting the inputs using a 350 ms presentation window based on our previous SNN simulation experiences [19, 20]. To train the network with MNIST, we used Restricted Boltzmann Machine (RBM) network topology.

Scalability setup: MNIST is a relatively small dataset to demonstrate the capability of our platform in the simulation of a large network (in the range of millions of neurons). Therefore, we set up the second experimental validation by generating a large number of spiking inputs randomly in a randomly connected network of millions of neurons. Twenty percent of neurons in this network are inhibitory neurons. Time-step is 1 ms for all experimental runs, and the simulated time span is 1000 ms.

5.1 Comparing different synchronisation algorithms

We evaluate the speed and accuracy performance of the system using clocked synchronous, GALS, and hardware idle detection (HID) as synchronous methods running on POETS as an event-based neuromorphic platform.

5.1.1 Speed performance results. The maximum number of neurons can be implemented on one box using clocked synchronisation, and GALS is 500 thousand for each, and using HID is two million neurons. HID algorithm for synchronisation not only has the potential of running the more scalable network on a similar hardware but also it is faster than two other algorithms as it is demonstrated in Figure 6. The limitation in implementing the number of neurons on one box for different synchronisation methods is due to respect to the message transferring guarantee in each time-step. In a clocked synchronous algorithm, each neuron sends a message to all connected neurons fired or not. These properties increase the debugging ability and better reporting but also increases the latency of running and compilation. In the GALS version, the system is more relaxed to being checked regularly. In the HID method, a neuron sends a message only if it fires while it happens infrequently comparing to the GALS and clocked approach.

To compare unicast messaging speed with multicasting communication, the results prove that multicasting increases the speed of communication. However, how much it speeds up the system is a trade-off between offloading work from the cores and overloading the programmable routers. To make the comparison more visible, we used all eight boxes of hardware for this experimental validation. Figure 6 (right) shows multicasting speed up the simulation particularly for a medium to a large number of neurons in the network.

5.1.2 Accuracy evaluation. We checked two different accuracy rates. One using the MNIST dataset to verify the network recognition rate and the other one using the scalability setup to check the single-neuron spiking activity in a large network of neurons compared to Brian as a reference. The accuracy using the 10000 MNIST test digit known as recognition rate is 89.7% using pure STDP learning and 91.4% using R-STDP learning methods. The recognition accuracy rate is not comparable with state-of-the-art

Figure 6: Speed comparison between different synchronisation algorithms (left). Comparing the running speed of different networks using unicasting versus multicasting communication (right).

DNN, however; we need to consider the results as a first inference experimental implementation on the POETS platform without tuning the learning parameters and using shallow SNN.

The second accuracy evaluation is the spike activity accuracy of a single neuron depends on membrane potential evaluation. To evaluate how accurate a neuron fires, we compare it to a neuron in Brian simulator as a reference. We run the hardware implementation model 10 times for each method to reach reliable accuracy. The accuracy for each model compared to Brian reference for the last run is shown in Figure 7.a1 to a3. For the clocked synchronisation method the electrophysiological activity accuracy is 96.3%. 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 96.1% with a better overlap of the spike timing over the Brian model. The raster plot of Figure 7.b shows a good overlap of the spike timing for a group of 100 neurons running on our hardware comparing to the Brian software simulation.

5.2 Comparison with existing approaches

Here we briefly compare the performance of POETS against a single SpiNNaker board, and Brian simulator using the same network. It is difficult to make completely like-for-like comparisons, but we have tried to compare based on one “box” worth of computing, i.e. whatever would occupy one slot in a server rack.

Figure 8 shows the total execution time as the number of neurons varies from 100 up to 2 million neurons. We can see that POETS is around 20x faster than Brian for 1M nodes, around 16x faster than SpiNNaker for 500K nodes. In order to show a broader comparison, we have also included a more qualitative comparison between different neuromorphic platforms in Table 1. Note that the SpiNNaker properties are for a 48-chip SpiNNaker node, not the whole system. Table 1 shows that while POETS cannot compete with analogue platforms on scale or performance, it compares favourably with the digital solutions, while still offering substantial flexibility.

6 CONCLUSION

This paper presents a neuromorphic platform implemented on the POETS system. This platform is an asynchronous event-driven compute engine, so to map SNNs into the system we had to convert them

Figure 7: a1) Clocked synchronisation, a2) GALS method, a3) and HID accuracy running on POETS compared to the Brian simulator, b) raster plot of spiking activity, Brian versus POETS (HID method).

Figure 8: Speed comparison among Brian simulator, SpiNNaker, and POETS as well as number of neurons per hardware thread.

into graphs of asynchronous state machines. To manage the computation and communication of event-driven systems, three methods for performing the algorithmic mapping are presented, along with speed and accuracy comparison. A reinforcement learning algorithm known as Reward-Modulated STDP (R-STDP) is implemented on POETS architecture. We presented an empirical performance

Table 1: A comparison of the large-scale neuromorphic systems.

Properties	TrueNorth	Neurogrid	BrainScales	SpiNNaker	NeuroFlow	POETS
Technology	Digital	Analog	Analog	Digital	Digital (FPGA)	Digital (FPGA)
Feature size	28 nm	180 nm	180 nm	130 nm	28 nm	28 nm
Chips	16	16	325	48	6	48
Power	3.2 W/system	3 W/system	500 W/system	80 W/board	40 W/board	42.8 W/board
Interconnect	2D mesh-unicast	Tree-multicast	Hierarchical	2D mesh-unicast	Multitrack	2D-mesh-uni/multicast
Neuron model	Configurable LIF	Adaptive IF	Adaptive IF	Programmable	LIF/Izhikevich	LIF/Izhikevich
Neurons	16 M	1 M	200 k	768 k	600 K	2 M
Synapses	4 G	4 G	40 M	768 M	600 M	400 M

evaluation of the SNN variants in a hardware “box” containing 6 FPGAs and 6144 threads, demonstrating that the system can support up to 2M neurons and 400M synapses. Compared with the Brian simulator, we show a speed-up of around 20x, and 16x versus a single 48-chip SpiNNaker node.

Briefly, in this paper based on our experimental design and validation, we offer three key take-home messages:

- (1) FPGA-based platform is a promising hardware solution for the future neuromorphic system design considering the high speed and parallel computation and communication properties of FPGAs. The hardware soft-core overlay similar to what we used in this paper, facilitates the programming bottleneck for users of FPGA-based system.
- (2) Using appropriate synchronisation method for an event-based system has a significant impact on the speed of neuromorphic system.
- (3) The reinforcement Reward-Modulated STDP (R-STDP) learning has a great potential for online (on-chip) learning in SNN systems.

Current work is focusing on better understanding the performance effects of different synchronisation methods, and balancing accuracy of simulation versus performance. We are also evaluating scaling of the system up to our latest 8-box systems, containing 48 FPGAs and 48000 threads for the next work.

7 ACKNOWLEDGMENTS

Thanks to Jonathan Beaumont and Matthew Naylor for their technical development supports. This work is sponsored by UK EPSRC grant EP/N031768/1 (POETS project).

REFERENCES

- [1] B. V. Benjamin and et. al. 2014. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proc. IEEE* 102, 5 (May 2014), 699–716. <https://doi.org/10.1109/JPROC.2014.2313565>
- [2] Romain Brette and et. al. 2007. Simulation of networks of spiking neurons: A review of tools and strategies. 23, 3 (2007), 349–398, 2007. <https://doi.org/10.1007/s10827-007-0038-6>
- [3] Kit Cheung, Simon R. Schultz, and Wayne Luk. 2012. A Large-Scale Spiking Neural Network Accelerator for FPGA Systems. In *ICANN*. Springer, 113–120.
- [4] Kit Cheung, Simon R. Schultz, and Wayne Luk. 2016. NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors. *Frontiers in Neuroscience* 9 (2016), 516. <https://doi.org/10.3389/fnins.2015.00516>
- [5] Mike Davies and et al. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. 38, 1 (2018), 82–99. <https://doi.org/10.1109/MM.2018.112130359> IEEE Micro, 2018.
- [6] Mazdak Fatahi, Mahyar Shahsavari, Mahmood Ahmadi, Arash Ahmadi, Pierre Boulet, and Philippe Devienne. 2018. Rate-coded DBN: An online strategy for spike-based deep belief networks. *Biologically Inspired Cognitive Architectures* 24 (April 2018), 59–69. <https://doi.org/10.1016/j.bica.2018.04.009>
- [7] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. 2014. The SpiNNaker Project. *Proc. IEEE* 102, 5 (2014), 652–665. <https://doi.org/10.1109/JPROC.2014.2304638>
- [8] E. M. Izhikevich. 2003. Simple Model of Spiking Neurons. *Trans. Neur. Netw.* 14, 6 (Nov. 2003), 1569–1572. <https://doi.org/10.1109/TNN.2003.820440>
- [9] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. 2016. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience* 10 (2016). <https://doi.org/10.3389/fnins.2016.00508> Publisher: Frontiers.
- [10] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. 2008. A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLOS Computational Biology* 4, 10 (10 2008), 1–27. <https://doi.org/10.1371/journal.pcbi.1000180>
- [11] De Ma, Juncheng Shen, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. 2017. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. 77 (2017), 43 – 51, 2017. <https://doi.org/10.1016/j.sysarc.2017.01.003>
- [12] Paul A. Merolla and et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (Aug. 2014), 668–673. <https://doi.org/10.1126/science.1254642>
- [13] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. 2018. First-Spike-Based Visual Categorization Using Reward-Modulated STDP. *IEEE Transactions on Neural Networks and Learning Systems* 29, 12 (2018), 6178–6190. <https://doi.org/10.1109/TNNLS.2018.2826721>
- [14] M. Naylor, S. W. Moore, and D. Thomas. 2019. Tinsel: A Manythread Overlay for FPGA Clusters. In *29th FPL*. 375–383. <https://doi.org/10.1109/FPL.2019.00066>
- [15] Michael Pfeiffer and Thomas Pfeil. 2018. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers Neurosci* 12 (2018), 774. <https://doi.org/10.3389/fnins.2018.00774>
- [16] Alexander Rast, Mahyar Shahsavari, Graeme M. Bragg, Mark L. Vousden, David Thomas, and Andrew Brown. 2020. A Hardware/Application Overlay Model for Large-Scale Neuromorphic Simulation. In *2020 International Joint Conference on Neural Networks (IJCNN)*. 1–9. <https://doi.org/10.1109/IJCNN48605.2020.9206708>
- [17] Johannes Schemmel, Daniel Brüderle, Andreas Gribbl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. 2010. A wafer-scale neuromorphic hardware system for large-scale neural modeling. *IEEE*, 1947–1950. <https://doi.org/10.1109/ISCAS.2010.5536970>
- [18] Mahyar Shahsavari, Jonathan Beaumont, David Thomas, and Andrew Brown. 2021. POETS: A Parallel Cluster Architecture for Spiking Neural Network. *IJMLC* 11, 4 (2021), 281–285. <https://doi.org/10.18178/ijmlc.2021.11.4.1048>
- [19] Mahyar Shahsavari and Pierre Boulet. 2018. Parameter Exploration to Improve Performance of Memristor-Based Neuromorphic Architectures. *IEEE Transactions on Multi-Scale Computing Systems* 4, 4 (2018), 833–846. <https://doi.org/10.1109/TMSCS.2017.2761231>
- [20] M. Shahsavari, P. Devienne, and P. Boulet. 2015. N2S3, a Simulator for the Architecture Exploration of Neuromorphic Accelerators. In *2nd International Workshop on Neuromorphic and Brain-Based Computing Systems (NeuComp 2015) in DATE Conference, Grenoble, France*.
- [21] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press. <http://www.cs.ualberta.ca/~sutton/book/the-book.html>
- [22] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. 2019. Deep learning in spiking neural networks. *Neural Networks* 111 (2019), 47 – 63. <https://doi.org/10.1016/j.neunet.2018.12.002>
- [23] M. Walravens, E. Verreyken, and J. Steckel. 2020. Spiking Neural Network Implementation on FPGA for Robotic Behaviour. *Lecture Notes in Networks and Systems* 96 (2020), 694–703. https://doi.org/10.1007/978-3-030-33509-0_65
- [24] Timo Wunderlich, Akos F. Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, Christian Pehle, Sebastian Billaudelle, Gerd Kiene, Christian

Mauch, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici. 2019. Demonstrating Advantages of Neuromorphic Computation: A Pilot Study. *Frontiers in Neuroscience* 13 (March 2019). <https://doi.org/10.3389/fnins.2019.00260>

[25] Shijie Zhou, Rajgopal Kannan, Hanqing Zeng, and Viktor K. Prasanna. 2018. An FPGA Framework for Edge-Centric Graph Processing. In *Proceedings of the 15th ACM ICCF*. NY, USA, 69–77. <https://doi.org/10.1145/3203217.3203233>