Performance Comparison of Time-Step-Driven versus Event-Driven Neural State Update Approaches in SpiNNaker

Amirreza Yousefzadeh[1], Mikel Soto[1], Teresa Serrano-Gotarredona[1],
Francesco Galluppi[2], Luis Plana[2], Steve Furber[2], and Bernabe Linares-Barranco[1]

[1]*Instituto de Microelectronica de Sevilla (CSIC and Univ. de Sevilla), Sevilla, Spain Email: bernabe@imse-cnm.csic.es*
[2]*Dept. Comp. Science, University of Manchester, UK*

***Abstract*—The SpiNNaker chip is a multi-core processor optimized for neuromorphic applications. Many SpiNNaker chips are assembled to make a highly parallel million core platform. This system can be used for simulation of a large number of neurons in real-time. SpiNNaker is using a general purpose ARM processor that gives a high amount of flexibility to implement different methods for processing spikes. Various libraries and packages are provided to translate a high-level description of Spiking Neural Networks (SNN) to low-level machine language that can be used in the ARM processors. In this paper, we introduce and compare three different methods to implement this intermediate layer of abstraction. We have examined the advantages of each method by various criteria, which can be useful for professional users to choose between them. All the codes that are used in this paper are available for academic propose.**

I. Introduction

The SpiNNaker platform is a digital multi-core and multi-chip neuromorphic hardware optimized for spike communication and massively parallel computation [1].

Each SpiNNaker chip is composed of 18 ARM968 cores, each with 32kB of instruction memory and 64kB of data memory[1]. A 128MB SDRAM is shared between the cores [2]. Each chip can be connected to other chips using the 2-of-7 non-return-to-zero protocol [3] and has 16 available cores for user[2]. Fig. 1 shows the layout of the SpiNNaker chip.

For this work, we have used two different boards, one with 4 SpiNNaker chips (SPIN-3) and the other with 48 SpiNNaker chips (SPIN-5) as shown in Fig. 2.

The SpiNNaker software sits on top of this hardware to allow a smooth design and simulation of different neural network configurations. Each SpiNNaker chip runs an application programming interface (API) on top of a specific event-based kernel. The host machine runs a Python package (PyNN) for the specification of the neural network structures.

Using PyNN description language [4], the user can specify different neural network topologies and parameters such as populations, synapses, projections and neuron models. Finally, another specific tool maps the PyNN neural network description to the SpiNNaker resources generating and downloading binary files for real-time simulation of the neural network.

In hardware, a population of neurons is assigned to each ARM core. The states of these neurons are stored in the local data memory and can be updated with different events. ARM cores communicate with each other through a packet switch network on chip. Each packet carries the source address that contains the neuron's ID, core ID, and chip ID. Each chip includes a router that communicates to all the cores and external links. The routing tables of the routers are programmed to establish the predefined neural connections. Additionally, the SDRAM memory in each chip can be used to store synaptic weights and allow each neuron to be connected to a few thousand synapses. Several alternative methods to implement

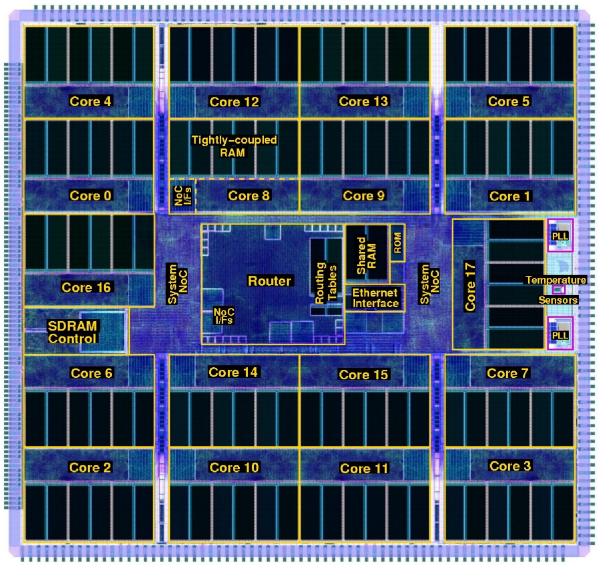
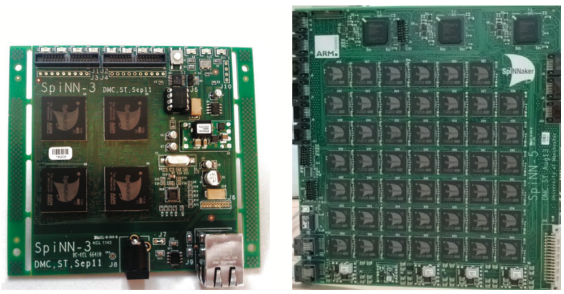


Fig. 1: SpiNNaker chip layout [1]. It contains 18 ARM processors, a Router and SDRAM controller.



(a) (b)

Fig. 2: (a) 4 chip SpiNNaker board [SPIN-3], (b) 48 chip SpiNNaker board [SPIN-5]

[1]In this paper we used the first commercial version of SpiNNaker chip. A Newer version of SpiNNaker chip is under development at the time of this paper

[2]From 18 cores, one is used for management and another one is reserved.

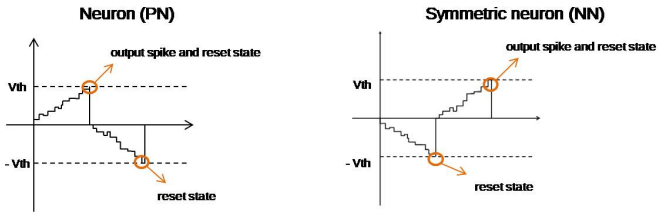


Fig. 3: Behavior of the neuron that acts as positive neuron (PN) and negative neuron (NN)

Spiking Neural Networks (SNN) in SpiNNaker (besides the standard PyNN based approach) have been reported to process spikes and store neuron states and synaptic weights [5], [6], [7]. In this work, we present three different methods to map an SNN on SpiNNaker and compare their performances. All methods use the same neuron model, which is not directly supported in the standard PyNN based approach.

The neuron model used is entirely event-driven. The state of the neuron evolves with time. In our work, we have used signed event-driven spiking neurons, as described in [8]. This neuron is a Leaky Integrated and Fire neuron with two thresholds. When a neuron membrane exceeds the positive threshold, it will generate a positive spike, and when it exceeds a negative threshold, a negative spike will be created. Moreover, leakage in this neuron is linear.

For this work, a previously reported poker card symbol recognition Convolutional Spiking Neural Network (ConvNet) is used as benchmark [8]. This network is composed of 4 convolutional layers interleaved with two subsampling stages. Input spikes come from Dynamic Vision Sensor [9]. Section 2 describes the different implementations. Section 3 shows the experimental setup and results obtained from each implementation. Finally, Section 4 summarizes the peculiarities of each implementation.

II. Alternative Benchmark Implementations on SpiNNaker

A. Time-Step-Driven Implementation (based on the Standard SpiNNaker Approach)

This implementation was done using the available SpiNNaker software version. A new neuron model compatible with the standard models was created to implement the poker card ConvNet. In standard SpiNNaker software, each neuron will be updated at a regular time step that is 1ms by default.

Our event-driven neuron fires positive and negative spikes, however, the SpiNNaker software does not support negative spikes. Therefore, we have created a new sub-neuron model PN that has two voltage thresholds with the same value but inverse sign. This sub-neuron fires and resets when the membrane voltage exceeds the positive threshold, and it resets without firing when the membrane voltage goes below the negative threshold, as shown in Fig. 3. If we now create a symmetric sub-neuron NN that behaves inversely and combine both {PN, NN}, we get the positive and negative spikes.

The connection between a pre-layer neuron and a post-layer neuron is shown in Fig. 4. The connecting weight w can, in

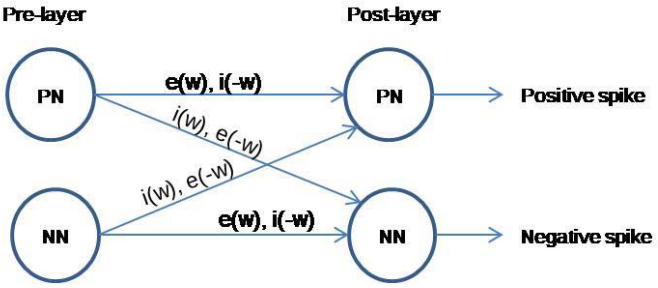


Fig. 4: Pre-layer post-layer connection scheme. e(w) is excitatory target and i(w) is inhibitory target

principle, be positive or negative. If positive, an excitatory synapse e(w) is used and if negative, an inhibitory synapse i(-w) is used. Positive spikes from pre sub-neuron PN are connected to post sub-neuron PN using {e(w),i(-w)}, but the effect is sign-reversed when connecting to post sub-neuron NN {e(-w),i(w)}. The connections from pre NN to the two post sub-neurons are symmetrically reversed.

The ConvNet architecture to perform recognition of the card symbols of [8] with this implementation needs 14332 neurons and 130 cores (9 SpiNNaker chips) using 100 neurons per core.

B. Spike-Driven Implementation

In the original SpiNNaker software, there is a millisecond time step and each neuron will be updated every millisecond. There are two significant disadvantages for this millisecond time step. First, temporal precision will be limited to the time step that is 1ms by default. Second, even without any incoming activity, neurons are continuously updated every time step. If more timing resolution or fully event-driven power consumption is desired, one may prefer a neuron that updates only after receiving a spike. It is believed that most of the information of spikes are at the time of firing [10] and losing timing accuracy may have a significant effect on some SNN implementations.

In this implementation, the original SpiNNaker software has been modified to include the neuron model that generates positive and negative spikes. SpiNNaker uses AER packets [11] for spike communication. To add polarity to spikes, we needed to modify the structure of the AER packets. Additionally, we removed the time step neuron update. In this case, neurons will be updated immediately after receiving spikes.

When a spike enters to a processor, first it will load the proper synaptic weight from SDRAM memory, then it will immediately update the neuron potential. If the spike has the positive polarity, the membrane potential of the neuron is increased by the synaptic weight value. Conversely, if the spike has a negative polarity, the membrane potential will be decreased by the synaptic weight value.

In this implementation there is no time step for every millisecond, so every neuron should keep track of their last time of receiving spike (for calculating leakage) and the last time of generating a spike (for calculating the refractory period).

After the update of the membrane potential of the neuron, if it exceeds the positive threshold, a positive post-synaptic spike will be generated, and if it overcomes the negative threshold, a negative post-synaptic spike will be generated.

All the above processes in each core of SpiNNaker chips will be done immediately after a spike enters and it takes around 10us in average. So each core in this implementation can handle around 100k synaptic updates per second independent of how many neurons are inside this core. For this poker card implementation, we used 104 ARM cores in SpiNNaker hardware.

C. Convnet Optimized Implementation

The main difference of this ConvNet implementation is that it takes advantage of the weight sharing property of the ConvNets. This weight sharing property highly reduces the number of synaptic weights that must be stored for a neuronal population.

In this implementation, the original SpiNNaker software has been modified to admit a particular "convolution connector" [7]. This "convolution connector" stores in the local data memory of the chip the kernel weights of the corresponding population. Each feature map shares a "convolution connector" for every neuron in it. The implementation is entirely event-driven because each event is processed at the time it arrives. When an event reaches to the convolution module, the corresponding kernel is applied to update the neuron state and the neighbor pixels. Because the weights are stored in the local data memory, there is no need to read the synaptic weights from the SDRAM.

Furthermore, we distinguish between the shared parameters of the neurons in a population like voltage threshold, leakage rate, refractory time and so on and non-shared parameters like each neuron state and firing times. The shared neuron parameters are stored once per population in the local memory. In the original SpiNNaker software, each neuron parameters are stored in the data memory. Storing the parameters in the local memory provides a high-speed access, but the capacity of this local memory can limit the number of neurons that can be implemented per core. Specifically, we are able to implement 2048 convolution neurons per core, where this number is determined by the maximum number of addressable neurons by the implemented routing scheme. This method uses 22 ARM cores for the poker card recognition benchmark. This method was the only one that could fit into the SPIN-3 board with 4 SpiNNaker chips.

III. Experimental Setup and Results

The POKER-DVS Database [12] is recorded by shuffling poker cards in front of the DVS. This dataset has high event rate. Processing events and recognizing symbols in real-time is a challenging task. In previous work [8] a convolutional spiking neural network to process poker card symbols is introduced and implemented in a software simulator. In this work, we have implemented this network in PyNN [4] and mapped it to SpiNNaker using the previous three implementation methods.

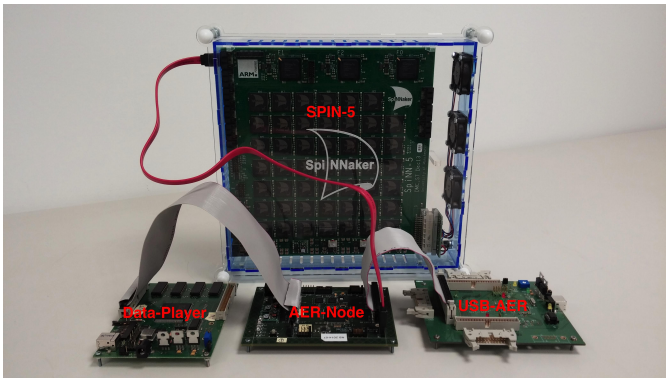


Fig. 5: Experimental setup. Events flew from Data-Player [11] to AER-NODE board [13] to SPIN-5 board. The processed events come back from the SPIN-5 board to AER-NODE board and they go to USB-AER [11] board to be sent to the computer.

When we send the spikes in real-time (high event rate) to SpiNNaker, the cores sometimes cannot handle all the spikes with the same speed as they arrive. In this case, the old spikes will be dropped which may decrease the accuracy of the network. A different implementation of the Spiking Neural Network in SpiNNaker can have different processing efficiency and throughput. We measured and compared the accuracy of the previously mentioned methods when presenting the POKER-DVS Database in real-time and with different slowed-down factors to the SpiNNaker boards.

Fig. 5 shows one of our experimental setups for real-time experiments with SpiNNaker. We loaded the POKER-DVS events sequence in a data player board [11]. The data player board stores the addresses and timestamps of the recorded events in a local memory and reproduces the events through a parallel AER link in real time.

The AER-Node board [13] contains a Spartan-6 FPGA and two parallel AER ports. This board receives AER events from the Data-Player and puts them on a fast serial link [14] to be sent to the SPIN-5 board. The processed spikes will be sent back to the AER-Node board on the same fast serial link. Finally, the AER-Node board sends the output spikes to the USB-AER board [11] to be sent to the computer through a USB port.

The classification is considered successful when the number of output events for the correct category is higher than for the other categories. We repeated the experiment for different slowed-down factors of the events of the input stimulus sequence. Also, we applied the same slowed-down factor to the network timing parameters for a correct time scaling. Fig. 6 shows the average recognition success rates obtained for different slowed-down factors. A '1' slowed-down factor means real-time operation.

As it can be seen, when the slowed-down rate is very high, all the implementations will show almost similar performance, which is close to what is observed when using the software simulator [8]. When spike presentation is closer to real-time, the time-step-driven implementation shows better performance. This result shows that the time-step-driven

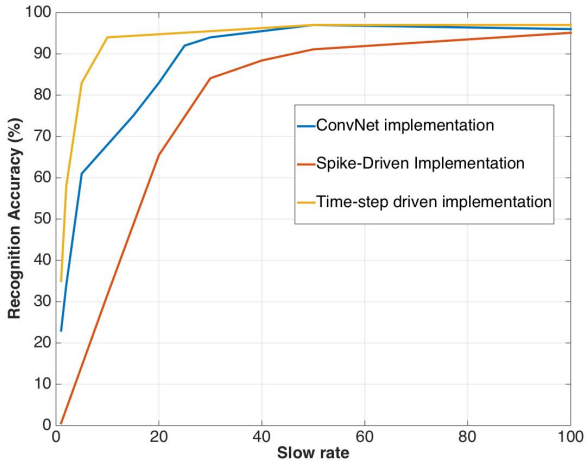


Fig. 6: Accuracy of symbol recognition versus slow rate of POKER-DVS events from real-time

implementation needs the minimum amount of processing (neuron updates) when the event rate is very high. The spike-driven implementation shows the worst results for high event rates because it needs the maximum processing time per spike among all the methods. Beside processing efficiency, other factors should be considered. For example, while the time-step-driven implementation occupied 130 ARM cores, the ConvNet implementation only needed 22 cores. Additionally, spike timing resolution in the spike-driven implementation is 10us while it is 1ms for the time-step-driven implementation. It is important to mention that the neural network parameters were obtained by mapping from a frame-based training method [8]. Therefore, the network accuracy could be more sensitive to the rate of spikes than the time of spikes.

IV. Conclusions

In this work, we have compared three different neural network implementations using the SpiNNaker platform. The first implementation is time-step-driven processing which is the standard method for the SpiNNaker software. We have shown that this method has the highest recognition performance and experiences less spike congestion and therefore less event dropping. For the second method, spike-driven implementation, we removed the time step constraint and spikes are processed immediately. We have shown that this method has a high timing resolution in comparison to the time-step-driven method while it needs more processing time per spike. Finally, for the ConvNet optimized implementation, the spikes are processed immediately and the synaptic weights can be stored in the local data memory of the ARM processors rather than in SDRAM. This method has very high timing resolution and good processing performance. When the input event rate is high, this method showed worse performance than the time-step-driven but a better performance than the spike-driven implementation. Also, this method is limited to convolutional connections and cannot be used for fully connected networks.

Acknowledgements

This work was supported in part by the EU H2020 grants 644096 ECOMODE and 687299 NEURAM3, and by the Spanish grant from the Ministry of Economy and Competitivity TEC2015-63884-C2-1-P (COGNET) (with support from the European Regional Development Fund).

References

[1] S. B. Furber, et al., Overview of the spinnaker system architecture, IEEE Transactions on Computers 62 (12) (2013) 2454–2467.

[2] E. Painkras, et al., SpiNNaker : A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation 48 (8) (2013) 1943–1953.

[3] L. A. Plana, et al., An on-chip and inter-chip communications network for the spinnaker massively-parallel neural net simulator, in: Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008), 2008, pp. 215–216.

[4] A. Davison, et al., Pynn: a common interface for neuronal network simulators, Frontiers in Neuroinformatics 2 (2009) 11.

[5] X. Lagorce, et al., Breaking the millisecond barrier on spinnaker: implementing asynchronous event-based plastic models with microsecond resolution, Frontiers in Neuroscience 9 (2015) 206.

[6] G. Orchard, et al., Real-time event-driven spiking neural network object recognition on the spinnaker platform, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 2413–2416.

[7] T. Serrano-Gotarredona, et al., ConvNets experiments on SpiNNaker, Proceedings - IEEE International Symposium on Circuits and Systems 2015-July (2015) 2405–2408.

[8] J. A. Perez-Carrasco, et al., Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets 35 (11) (2013) 2706–2719.

[9] T. Serrano-Gotarredona, B. Linares-Barranco, A 128 × 128 1.5% contrast sensitivity 0.9% *FPN* 3us latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers, IEEE Journal of Solid-State Circuits 48 (3) (2013) 827–838.

[10] T. Masquelier, et al., Microsaccades enable efficient synchrony-based coding in the retina: a simulation study, Scientific Reports 6.

[11] R. Serrano-Gotarredona, et al., Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking, IEEE Transactions on Neural Networks 20 (9) (2009) 1417–1438.

[12] T. Serrano-Gotarredona, B. Linares-Barranco, Poker-dvs and mnist-dvs. their history, how they were made, and other details, Frontiers in Neuroscience.

[13] T. Iakymchuk, et al., An aer handshake-less modular infrastructure pcb with x8 2.5gbps lvds serial links, in: 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014, pp. 1556–1559.

[14] A. Yousefzadeh, et al., On multiple aer handshaking channels over high-speed bit-serial bidirectional lvds links with flow-control and clock-correction on commercial fpgas for scalable neuromorphic systems, IEEE Transactions on Biomedical Circuits and Systems 11 (5) (2017) 1133–1147.

[15] A. Yousefzadeh, Real time demo, poker kard symbol detection, https://youtu.be/zkwezv7FknE (2015).