An Intrinsic Method for Fast Parameter Update on the SpiNNaker Platform

M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco
Instituto de Microelectronica de Sevilla (IMSE-CNM)
CSIC and Univ. de Sevilla
Sevilla, Spain
bernabe@imse-cnm.csic.es

***Abstract*—Neuromorphic Computing or Spiking (also called Event-Driven) Neural Systems are becoming of high interest as they potentially allow for lower power hardware computing platforms, where power consumption is data driven. Traditional approaches (both in software and in hardware), which are not data driven, rely on generic system state updates, consuming a fixed amount of computing resources at each step, independent on the data itself. In neuromorphic spiking or (event-driven) computing systems power is consumed (in principle) if new data is transferred, either at the system input, system output, or internally between computing nodes. One such neuromorphic event-driven computing platform is the scalable SpiNNaker system, which is aimed for a million ARM core platform, capable of emulating in the order of a billion neurons in real time. An important practical drawback of the platform is the long time it takes to download to the hardware a given computational architecture. This step has to be repeated even if one wants to update a set of parameters. Here we present a method for updating internal parameters without downloading again the full architecture, by adding special neurons into the computing architecture which when they spike change given parameters. This allows to download the computing architecture only once to the SpiNNaker platform, and then take advantage of its highly efficient communication network to command specific parameter changes. This allows for intensive parameter searches in a more efficient manner.**

Keywords—Neuromorphic Computing, Spiking Circuits, SpiNNaker Platform, Event-Driven Computation, Addres-Event-Representation (AER)

I. Introduction

This paper tackles the problem of performing extensive parameter searches or obtaining the optimum parameters in neural systems implemented on the SpiNNaker hardware platform [1]. Prior reported parameter optimizations of spiking convolutional networks using software simulators [2], required several thousand of simulated annealing iterations to find an optimum set of about 10 network parameters in a 4-layer ConvNet (convolutional neural network). Similar ConvNets have been implemented on the SpiNNaker platform by tweaking the official software release to support weight sharing topologies optimized for ConvNets [3], or by simply replicating all synapses that use the same weight without tweaking the official software. In both cases, the software requires an important amount of time to "compile" and download the network with all its parameters, described in a high level abstract language (such as PyNN [4]), to the multi-core hardware. Depending on the complexity of the network, this time may range from several minutes to hours. Once compiled, the network runs then in real time a specific simulation. If one wants to change just one parameter in the network, it is necessary to recompile and download it again.

In this paper we present a method for avoiding recompiling the full network every time. It consists of defining a set of special types of neurons. These neurons can be stimulated directly from the outside, this is, they receive directly external dedicated input spikes. Also, these neurons connect through special synapses to other neurons in the network, but they do not actuate on them like typical neurons (this is, they increase/decrease their internal state through weighted synaptic connections). Instead, they change specific parameters of the destination neurons.

Section 2 briefly describes the steps followed by the SpiNNaker platform to simulate a specific design described in PyNN. In Section 3 we explain the proposed approach to modify neuron parameters during the simulation time and show an example using real DVS (Dynamic Vision Sensor [5]-[7]) recording data and providing time and classification accuracy results. Section 4 presents the conclusions.

II. OPTIMIZATION USING THE DEFAULT SPINNAKER SOFTWARE

The SpiNNaker platform is a bioinspired neuromorphic hardware based on spike communication and massively parallel computation [1]. The communication infrastructure is optimized for spike communication. The SpiNNaker hardware together with the SpiNNaker software allow for a fast and easy design and simulation of different neural networks. The SpiNNaker software can be divided in two parts. One that runs in the SpiNNaker chips and another that runs in the host machine. The main interface to communicate the host machine with the SpiNNaker chips is via Ethernet.

Using PyNN description language [4], the user can specify the neural network structure (populations, synapses, projections, neuron models, etc) and the SpiNNaker software configures the routers, splits populations and downloads the design using Ethernet to the SpiNNaker hardware. Next, it

runs the simulation for a specific time. During the simulation, the SpiNNaker chips store the neuron output spikes or/and neuron membrane voltages internally. When the simulation finishes, it can show the results on the screen or copy them to files. However, if the simulation time is too long, it can result in a lack of memory error. Another option for collecting the results is to activate live output packets. In this case, the SpiNNaker software can send the spikes out of the SpiNNaker board to a host computer using the Ethernet connection. Additionally, it can also export spikes to another hardware using available SATA ports [8].

The parameters of the neurons have to be defined along with the neural network design. Therefore, every time the user modifies the neuron parameter the design has to be changed. Finding the neuron parameter configuration that obtains the best performance for a neural network is a difficult task. The common procedure is to use automatic parameter search or some optimization algorithm (in this case we use basin-hopping from the python optimize packet [9]). The neural parameters optimization process using the default SpiNNaker software would be as follows. First, the user configures the callbacks and ports to receive UDP messages with the output spikes. Then, the user specifies the design and starts the SpiNNaker simulator. The simulator will download the design to the SpiNNaker hardware and launch the simulation. When the simulation is running, the user can start an external event generator that feeds physical events to the network (for example, from a DVS spiking sensor or from an event playback PCB [10]), and receive the output spikes via UDP communication. Once the SpiNNaker simulation finishes, the optimization algorithm computes a given cost function and if the result is not acceptable, it generates a new set of parameters. This set of new parameters is integrated onto the network description, which needs to be compiled and downloaded again to the SpiNNaker hardware.

In our case we used the poker card recognition 4-layer ConvNet presented elsewhere [2] as a benchmark. In this case, the compile and downloading time to the SpiNNaker platform was about 145 seconds. If the optimization process needs to iterate 2000 cycles it will spend at least 80 hours downloading the design to the SpiNNaker board (which is longer than what it took to optimize the original software simulations [2]).

III. Proposed Intrinsic Approach

A. Implementation

We define a new "config_neuron" capable of receiving multi-cast packages (MCP) directly from the host computer via Ethernet. This "config_neuron" is connected through new "config_synapses" to normal neurons in the computational network. This is illustrated in Fig.1. Using this approach it is possible to add a 32-bit or 16-bit payload to the MCP so that the payload encodes the neuron parameters to be changed and the new values to be assigned. Depending on the number of neuron parameter to be modified during the simulation time, there will be more or less bits of accuracy to represent the value of the neuron parameter. For example, using a 32-bit word of payload, to change 8 neuron parameters, 3 bits can be used to encode the parameter and 29 bits to represent the parameter value. The "config_neuron" can be connected to a single neuron or a population of neurons. All the neurons connected to the "config_neuron" will decode the incoming MCP packet and change its parameters. Therefore, if we want to change at the same time all the neuron parameters of a layer, we will need a "config_neuron" for each layer and this will be the extra overhead of the neural network.

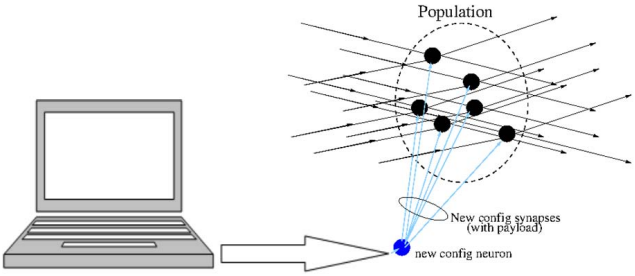

Fig. 1. "Config_neuron" interfacing with host computer and connected to a population

We used the SpiNNaker software version 2015.005, which did not support MCP with payload. Therefore, some software changes were done to adapt it. In particular, when a MCP with payload arrives to the core, it has to decode the neuron parameter to change, and call a specific function to change that parameter.

B. Single neuron behaviour

This section explains an example using a single neuron as shown in Fig. 2. Pop_1 is an IF_curr_exp population of 1 neuron connected to a spikes source that generates 1 spike every 10ms during 10s. Equation (1) describes the IF_curr_exp neuron model.

$$\tau_m(dV_m/dt) = (E_L - V_m) + R_m I(t) \quad (1)$$

where τ_m is the membrane time constant, E_L is the resting potential, V_m is the membrane voltage, R_m is the membrane resistance, and I represents the input current from the synapses. When V_m reaches a threshold V_θ, it is reset to E_L and the neuron remains inactive during a refractory time τ_{ref}.

For our purpose, we are interested in changing three neuron parameters from the PyNN front-end: τ_{ref}, τ_m, and V_θ. In the proposed example a "config_neuron" is used to interface Pop_1 with the host computer.

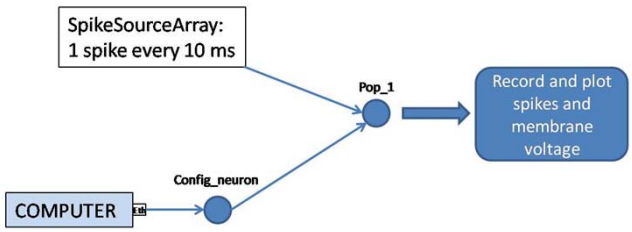

Fig. 2. Outline of the configuration of a single neuron parameters example

Fig. 3(a) shows the membrane voltage of a 10s simulation using the default SpiNNaker software and Fig. 3(b) shows the effect of changing the neuron parameters during the simulation time of the same experiment with the adapted software. The blue line in the figure represents the membrane voltage and the red stars correspond to output spikes.

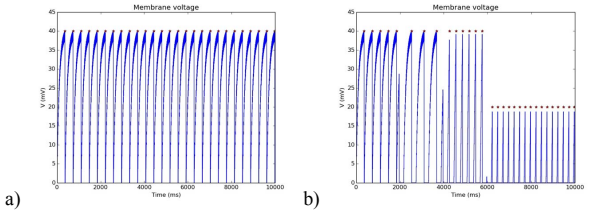

Fig. 3. Single neuron simulation result a) without changing the neuron parameters, b) changing the neuron parameters

Fig. 4 shows zoomed-in details of the times where parameters change during run time. At millisecond 2000, the host computer sends an MC packet with payload to change τ_{ref} to a longer time (see Fig. 4(a)). At millisecond 4000 the host computer changes τ_m increasing the value and therefore decreasing the leakage (see Fig. 4(b)). At millisecond 6000 V_θ is changed from 40mV to 20mV (see Fig. 4(c)).

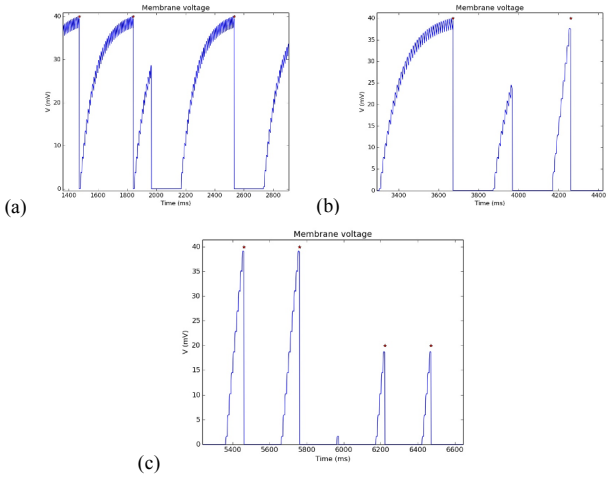

Fig. 4. Details of the changes in the (a) τ_{ref}, (b) τ_m and (c) V_θ during run time

C. Optimization of the Poker card recognition ConvNet

Fig. 5 shows the 4 layer poker symbol recognition ConvNet used for parameter optimization [2]. We want to optimize parameters τ_{ref}, τ_m, and V_θ for maximum recognition. All neurons in the same layer share the same parameter values. For the 1st and 4th layers $\tau_{ref} = 0$, thus only 10 parameters need optimization. The system is set up with just four config_neurons, as shown in Fig. 5. All the neuron parameters of a layer will be changed at the same time. In this case, sending a multicast (MC) message with payload and KEY = key0, will change the parameter encoded in the payload for all the neurons of layer C1. If the message KEY is key1, the corresponding parameter of all the neurons of layer C3 will be changed, and so on. The overall optimization procedure (which uses the basin-hopping algorithm of the python package Optimize [9]) is depicted in Fig. 6. The network is downloaded once to the SpiNNaker hardware and a spike burst stimulus is presented each iteration, evaluating a "recognition accuracy". A set of new parameters are sent through MCP, and this is iterated until convergence.

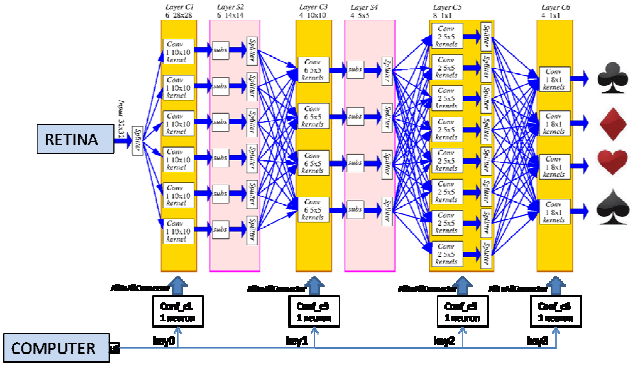

Fig. 5. Poker card recognition ConvNet with the addition of the "config_neurons"

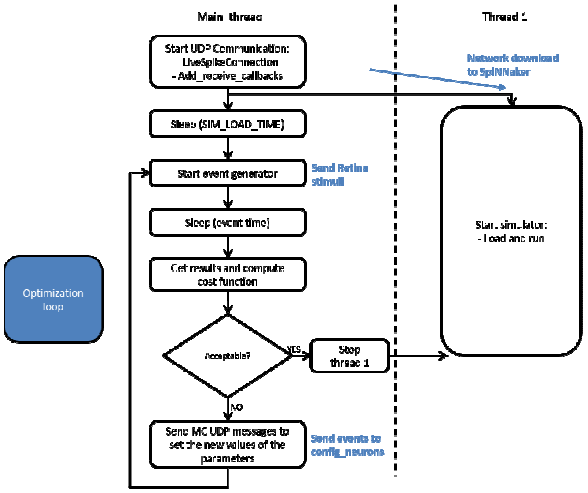

Fig. 6. Optimization Procedure

Fig. 7 shows the experimental setup. The event generator stores DVS recordings to be used as the stimulus. It is connected via an adapter pcb to a commercial RaggedStone2 board, which holds a Spartan6 FPGA programmed to convert DVS spikes into SpiNNaker events format, sending them to the parallel port of a 48-chip SpiNNaker board. Additionally, the Spinnaker board is connected via Ethernet to the host computer for receiving MC packets for the config_neurons.

The experiments were done using the Poker-DVS dataset [2], [11], where 40 card symbols are shown in about one second. The recognition was measured counting the number of symbols correctly recognized. Thus, 100% of recognition performance corresponds to 40 cards correctly recognized. The experiments were conducted by slowing down the input event stimulus at different slow-down rates SR = 1 (real time), 2, 5, 10, 50, 100. When using the proposed method, the overall optimization time is given by

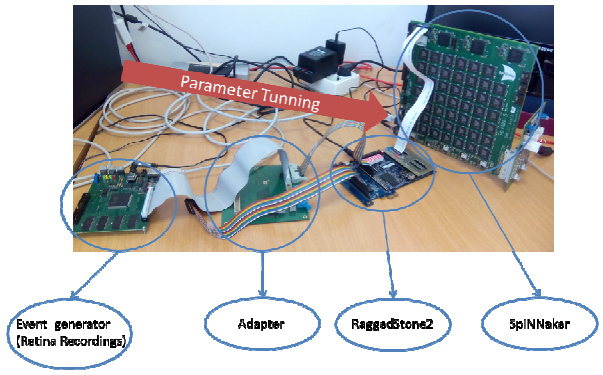

Fig. 7. Hardware setup

$$T_{optim} = T_{dl} + n_{iter}\,(T_{ov} + SR\;T_{sim}) \quad (2)$$

where T_{dl} is the download time, 145s in this case, n_{iter} is the number of optimization iterations, T_{ov} is the overhead time required in each iteration to launch the stimulus data and collect the output and compute the new parameters, SR the slow-down rate, and T_{sim} is the real time simulation latency on the SpiNNaker platform, 1s in this case. On the other hand, when using the conventional approach, the optimization time would be given by

$$T_{optim} = n_{iter}\,(T_{dl} + T_{ov} + SR\;T_{sim}) \quad (2)$$

Table 1 displays the results obtained when using both approaches.

TABLE I. NUMBER OF ITERATIONS AND OPTIMIZATION TIMES FOR DIFFERENT SLOW-DOWN RATES SR

SR	n_{iter}	T_{ov}	T_{optim} with proposed approach in seconds (and in hours)	T_{optim} with standard software in seconds (and in hours)
1	2160	14	32749 (9)	345600 (96)
2	1584	14	23952 (6)	253440 (70)
5	1130	15	23008 (6)	186450 (52)
10	138	15	3689 (1)	23460 (6)
50	1500	35	128068 (35)	345000 (95)
100	1440	41	203678 (56)	411840 (114)

D. *Testing optimized parameters in an application on SpiNNaker*

The optimized neuron parameters obtained from the optimization process at different slow-down rates were then used to run separate simulations, repeated 30 times, to measure the recognition performance and its standard deviation. Since SpiNNaker is an asynchronous network of synchronized cores with intentional timing jitters between the cores, when repeating a simulation the results may differ. Fig. 8 shows the recognition performance at the different slow-down rates. The error bar indicates the standard deviation. As can be seen, recognition improves with larger slow-down rates. With SR = 1, the recording plays back at real time with an input stimulus event rate of about 200*Keps* (kilo events per second). The SpiNNaker hardware updates all neural states using a 1*ms* time step. This time step implies that all the events that arrive to a neuron during the same 1*ms* will at the most generate one output spike. Therefore, when slow-down rate is 1, there are many input events coming in during the same millisecond (200 events for the rate of 200*Keps*) and significant information is lost degrading recognition accuracy. By increasing SR, fewer events are received during the same millisecond and therefore less information is lost, improving recognition. Recognition reaches its top with a slow-down rate of 50.

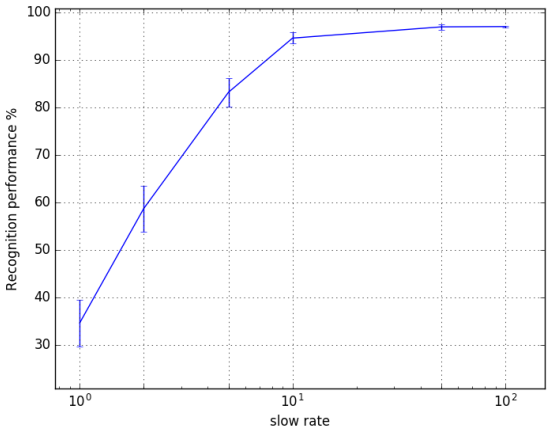

Fig. 8. Recognition capability at different slow-down rates

IV. Conclusions

An intrinsic approach has been proposed for updating neural population parameters on the SpiNNaker platform during run time. This allows for more efficient parameter searches and optimizations. Experimental tests have been performed using as benchmark a previously reported spiking ConvNet for high speed DVS-recorded poker symbol recognition. An improvement in optimization speeds of over a factor 10 has been observed. Further future improvements might be possible by investigating ways to reduce the overhead times required to trigger the stimuli and to process the collected outputs every cycle.

Acknowledgments

This work was supported in part by the EU H2020 grants 644096 "ECOMODE" and 687299 "NEURAM3", by Flagship grant "The Human Brain Project" FP7-604102, 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, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," Proc. IEEE, vol. 102, no. 5, pp. 652–665, 2014.

[2] J. A. Perez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate-Coding. Application to Feed Forward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, 2013.

[3] T. Serrano-Gotarredona, B. Linares-Barranco, F. Galluppi, L. Plana, and S. Furber, "ConvNets experiments on SpiNNaker," *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2015–July, pp. 2405–2408, 2015.

[4] Davison AP, Brüderle D, Eppler J, et al. PyNN: A Common Interface for Neuronal Network Simulators. Frontiers in Neuroinformatics. 2008;2:11. doi:10.3389/neuro.11.011.2008.

[5] Lichtsteiner P., Posch C., Delbruck T. (2008). A 128 × 128 120dB 30mW asynchronous vision sensor that responds to relative intensity change. IEEE J. Solid-State Circuits 43, 566–576.

[6] Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. IEEE J. Solid State Circ. 46, 259–275.

[7] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3us Latency 4mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Amplifiers," IEEE J. Solid-State Circuits, vol.48, No. 3, pp. 827-838, March 2013.

[8] A. Yousefzadeh, M. Jablonski, T. Iakymchuk, A. Linares-Barranco, A. Rosado, L. A. Plana, S. Temple, T. Serrano-Gotarredona, S. Furber, and B. Linares-Barranco, "On Multiple AER Handshaking channels over High-Speed Bit-Serial Bi-Directional LVDS Links with Flow-Control and Clock-Correction on Commercial FPGAs for Scalable Neurmorphic Systems," IEEE Trans. on Biomedical Circuits and Systems, vol. 11, No. 5, pp. 1932-4545, Oct. 2017.

[9] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.basinhopping.html

[10] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas, T. Delbrück, S. C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, B. Linares-Barranco, "CAVIAR: A 45k-Neuron, 5M-Synapse, 12G-connects/sec AER Hardware Sensory-Processing-Learning-Actuating System for High Speed Visual Object Recognition and Tracking," IEEE Trans. on Neural Networks, vol. 20, No. 9, pp. 1417-1438, September 2009.

[11] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their History, How They were Made, and Other Details," Frontiers in Neuromorphic Engineering. Front. Neurosci. 9:481. doi: 10.3389/fnins.2015.00481. 30-Nov-2015.