Vol.2017-ARC-227 No.23
2017/7/27

Accelerating Spiking Neural Networks on FPGAs using OpenCL (UNREFEREED WORKSHOP MANUSCRIPT)

Artur Podobas[1,a)] Satoshi Matsuoka[1,b)]

Abstract: Spiking Neural Networks (SNNs) are artificial neural networks inspired by the biological brain. They are used to study everything from various aspects of neuroscience to artificial intelligence and machine learning.
SNNs are typically computed using general-purpose processors and the use of custom hardware is still fairly uncommon. Creating custom hardware is often time-consuming and error-prone. However, with the recent maturity in High-Level Synthesis (HLS) tools, algorithms can now be described using more abstract C/C++/Java programming models and automatically be transformed down to custom hardware.
In the present work we present our findings and experience in using HLS to accelerate SNNs. We describe our prototype framework and our FPGA design and empirically evaluate its performance against modern general-purpose processors. Our evaluation shows that our design can reach up-to 82% (73% average) of the performance delivered by Intel Xeon E5-2650v3– a CPU that is two years younger and built using better technology than our FPGA platform.

1. Introduction

Spiking Neural Networks (SNNs) are the third generation [1] neural networks. They are inspired by the biological nervous system and thus the main feature of SNNs is that they communicate through spikes– action potentials that are exchanged between neurons at varying frequency. Hence, information between neurons or groups of neurons is conveyed through the intensity of spikes and their amplitude (weights).

SNNs are today used to study various aspects of the brain and brain-inspired computing. Researchers are using SNNs to study learning (synaptic plasticity) [2], [3], modeling various parts of the human brain (e.g. the eye [5]) and aspects of machine-learning [6], [7], [8].

SNNs are commonly simulated using general-purpose processors (CPUs). Several frameworks such as NEST [9], Neuron [10] and Brian [11] support intra-node (e.g. OpenMP) and/or inter-node (e.g. MPI) parallelism to boost simulation performance. Some of these frameworks also support alternative *custom* compute platforms such as the SpiNNaker [12]. There are also some frameworks that focus on Graphics Processing Units (GPUs) (e.g. NeMo [13]) as well as more exotic low-power ASICs such as the IBM TrueNorth [14].

FPGAs have recently emerged as a viable, general-purpose, computation platform, able to compete with GPUs and general-purpose CPUs on some applications [15] with respect to power- and execution performance. However, FPGAs have classically been programmed using complex Hardware-Description Languages (HDLs) such as VHDL, Verilog or SystemC. Unfortunately, these low-level languages are complex to use, require hardware expertise and are often not portable. An alternative to HDLs is High-Level Synthesis (HLS).

HLS tools aim to bridge the gap between software and hardware engeering, providing a well-known interface (e.g. C/C++, OpenCL) to the software programmer and automatically transforming his application into custom hardware.

In the present paper we present a prototype framework for accelerating SNNs using FPGAs. Our framework leverages Intel's OpenCL FPGA compiler to synthesize our design down to hardware. Building upon Python, we provide users with a simple-to-use yet powerful framework for simulating spiking neural networks.

In short, our contributions are:

(1) A prototype framework capable of creating, managing and simulating spiking neural networks,
(2) An FPGA architecture for SNNs with focus on the Izhikevich neuron model,
(3) Empirical performance evaluation of our design, including a comparison to a state of the art SNN framework for CPUs

2. Background

Spiking Neural Networks (SNNs) are artificial neural networks created to be as biologically plausible as possible.

A SNN will typically consist of a large number of neurons that are interconnected to each other through the *axons*.

An axon is (most commonly) a single wire that transmits information (spikes) from a single neuron to other neurons; essentially

[1] Global Scientific Information and Computing Center, Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8552, Japan
[a)] podobas.a.aa@m.titech.ac.hp
[b)] matsu@is.titech.ac.jp

a broadcast to the set of neurons that the triggered neuron is connected to. Each axon has a delay, which is the amount of time it takes for a spike to move from one end of the axon to the other. At the end of each axon is a variable number of axon terminals; each axon terminal makes a *synaptic* connection with the *dendrites* of other neurons.

Each synapse has a weight (or conductance). The dendrites of a neuron accumulates all the potentials from connected synapses. When the potential across the cell membrane reaches a certain threshold (V_{th}) a spike is emitted into that neuron's output axon.

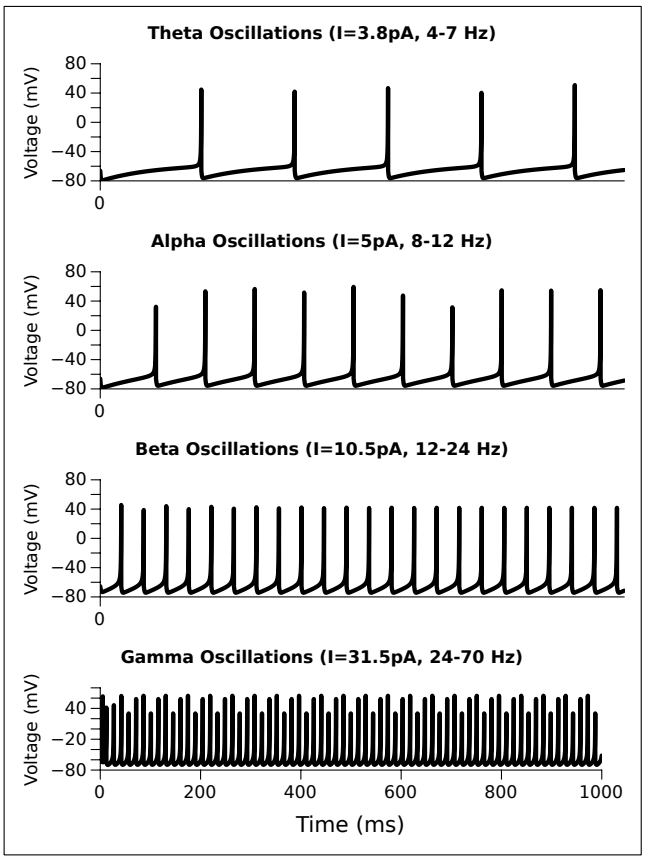

Fig. 1 Regular spiking of a Izhikevich-type neuron exercised by four different current levels, showing four naturally occuring oscillations in biological brains [16].

Figure 1 illustrates the activity of a single neuron when exercised with currents of various strengths. We see that a low-current yields a low spiking frequency while stronger current increase the spiking frequencies. Note how the membrane voltage always resets back to a preset voltage level after the membrane voltage reached the spiking threshold of (in this case) 30 mV.

2.1 Neuron Model

Choosing a neuron model is difficult: there are numerous neuron models with various advantage and disadvantages. For example, an Integrate-and-Fire (I&F) model is computationally very simple (5 FLOP per neuron and millisecond) but can only produce a single firing pattern and is biologically implausible. Such simple neurons (and their leaky variants) are often used to study various aspects related to plasticity and learning of SNNs [2], [17].

On the other hand, the Hodgkin-Huxley squid neuron model [18] has biological meaning and is capable of producing a rich set of firing patterns, but is extremely computationally costly (1k+ FLOP per neuron and millisecond). Complex models such as Hodgkin-Huxley or Morris-Lecar [19] are used when greater biological detail and meaning is required.

One of the more popular models – and the model we focus on in the present study – is the Izhikevich model [20]. The Izhikevich model variables and parameters carry no biological meaning. However, it is computationally simple and yet capable of producing a rich set of neuron spiking patterns [21], rivaling that of the far more complex Hudkin-Huxley model.

The Izhikevich model is described using the following two ordinary differential equations:

$$\frac{\partial V}{\partial t} = 0.04V^2 + 5V - U - I_{membrane} \tag{1}$$

$$\frac{\partial U}{\partial t} = a(bV - U) \tag{2}$$

The spiking behaviour of the neuron is set through the following condition:

$$if(V \geq 30mV)then \begin{cases} V = c \\ U = U + d \end{cases} \tag{3}$$

where V is the neuron membrane voltage, U is the membrane recovery variable, $I_{membrane}$ is the accumulated current given by all synaptic acitivty and a,b,c,d are tunable membrane parameters.

3. An OpenCL-based Neomorphic Architecture for FPGAs

We developed a framework capable of exploiting FPGAs to accelerate spiking neural networks. An overview of our framework is seen in Figure 2a. The front-end consists of a Python library that allows the user to interact with our framework.

Our frameworks aims to be as minimalistic as possible, meaning that only core functionality for creating neurons, managing axons and synapes, generating stimuli and controlling the simulation is provided. We expect our library to be leveraged by other, more abstract, frameworks such as PyNN [22]. Currently our framework supports three neuron models: Izhikevich, Morris-Leccar and Hodgkin-Huxley. Only the Izhikevich model is currently accelerated through FPGA execution and is thus the topic of this paper.

When the user has finished creating her network and initiates the actual simulation, our framework goes through a number of steps. Initially, we check for the presence of an FPGA device capable of support OpenCL execution. This is done by OpenCL API function calls provided by Intel's OpenCL runtime system. Upon finding a viable device and when initialization is finished, we continue to fully rebuild the neural network.

Originally, the graph is built to be as versatile and functional as possible. This means that there are a large number of extra variables allocated (debugging info etc.) that are unneeded when executing on the FPGA. This is why we fully rebuild the

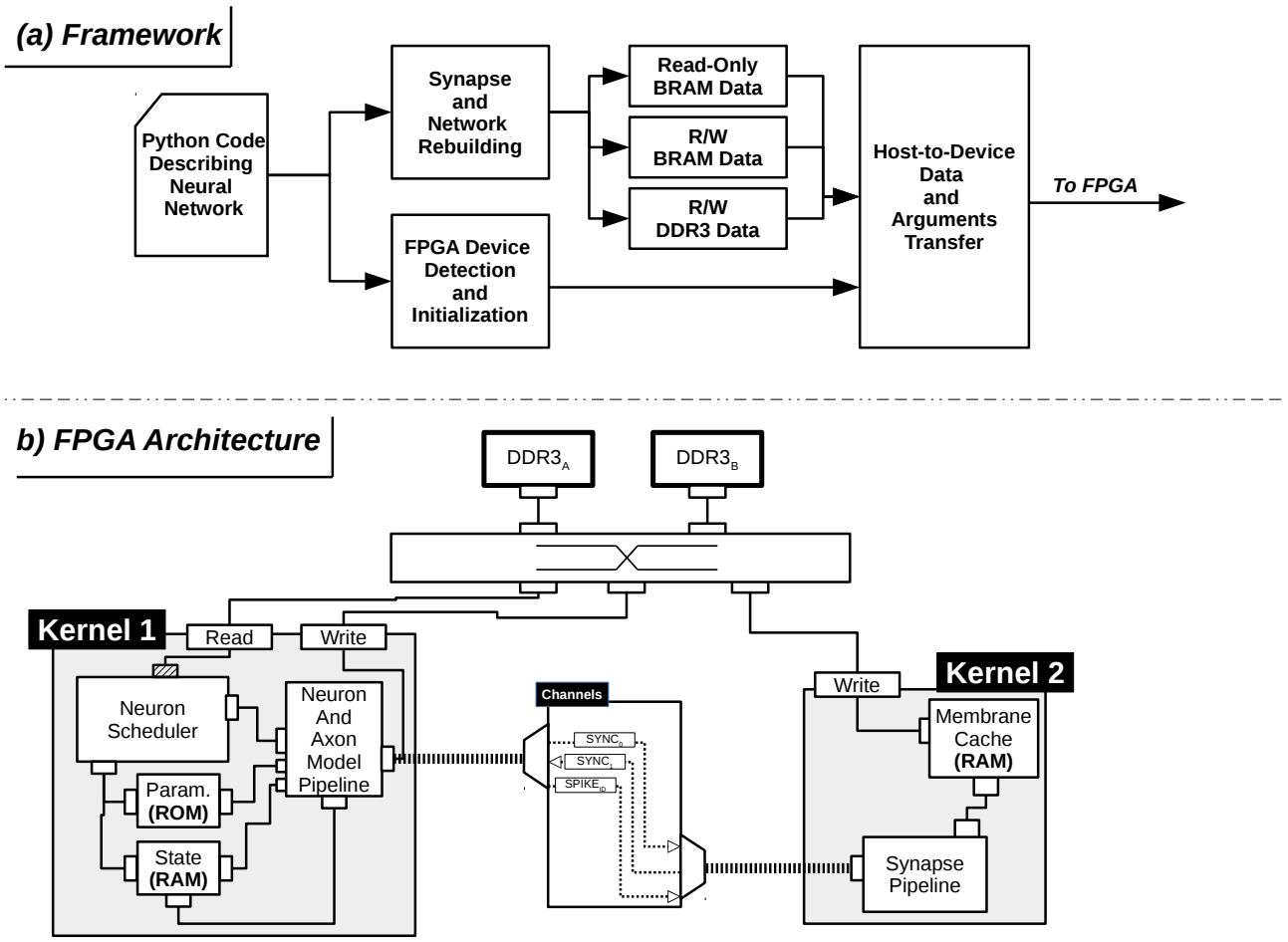

Fig. 2 a) Our prototype framework, and b) the developed FPGA architecture

graph the first time the simulation starts on the FPGA. More specifically, we sort the network-related data into three groups: DDR3-data, BRAM-ReadOnly, BRAM-ReadAndWrite. These are sorted according to the architecture of the FPGA (described in subsection 3.1) and must be transferred independently. For example, the membrane potential of each neuron is located inside the class/structure of the owning neuron on the host processor; on the FPGA, the membrane potentials are continuous in memory to promote spatial locality and wide burst accesses.

When the graph has been rebuilt and transfered, the framework sets the parameters for the two OpenCL kernels and starts execution.

3.1 Hardware Architecture

Figure 2b shows the OpenCL based architecture that we developed. It consists of two kernels – both operating in parallel – interconnected through the avalon interface to two DDR3 banks. The kernels have been developed to use the *single workitem* programming model; that is, both kernels focus on exploiting large amounts of instruction-level parallelism rather than relying on multithreaded execution.

Each of the two kernels are responsible for different parts of the neural computation and they both communicate and synchronize through dedicated on-chip channels (also known as pipes or FIFOs).

Kernel_1 is responsible for the main computation of the Izhikevich neuron model. It consists of a neuron scheduler, which fetches information regarding each neurons state (U,V and axon state) and its associated parameters (the variables a,b,c,d and V_{th}); these values are fed into a pipeline that perfoms the core computation. The scheduler can schedule one neuron every clock cycle, that is, the pipeline has an initiation interval (II) of one. The neuron models' two differential equations are computed through forward Euler method with a time-step of 0.1 ms. The results of the each computation is saved back into the BlockRAM that holds the newly computed neuron's state.

Kernel_1 is also responsible for simulating the spikes and their transversal down the neuron's axon. Each axon is modeled as a as a 64-bit unsigned value. A newly spiked neuron will set a bit in this variable according to the axon delay. For example, for a axon delay of 1 ms, the neuron will set the 10th bit of the axon value upon spiking. The axon variable will be be logically shifted right one step per timestep. When the least significant bit of the axon variable becomes set, it means that the spike reached the end of the axon. This implementation also support multiple spikes in flight inside one axon.

Table 1 Platforms used in our evalation (Tech. = Litography technology, HTs = hyper-threads)

Param.	E5-2670	E5-2650v3	E5-2650v4	Phi-7210F	FPGA
Date	2012	2014	2016	2016	2012
Tech.	32 nm	22 nm	14 nm	14 nm	28 nm
Cores	8	10	12	64	-
HTs	16	20	24	256	-
L2	256 kB	256 kB	256 kB	32 MB	-
L3	20 MB	25 MB	30 MB	-	-
Freq. (GHz)	2.6	2.4	2.2	1.3	0.258

When a spike reaches the end of the axon, Kernel_1 will send a message over to the Kernel_2, notifying it of an arriving spike. The message includes a pointer to the topology data in off-chip memory as well as a time-stamp and the id-number of the spiking neuron. After each fully simulated time-step, Kernel_1 will synchronize with Kernel_2 through dedicate channels.

Kernel_2 is responsible for delivering and accumulating synaptic potentials across the network topology; in other words, everytime a spike reaches the end of its axon, the spikes potential (weight) should be added to the membrane of all neurons that have a synaptic connection with the axon terminal.

Kernel_2 contains a synaptic pipeline, which fetches parts of the network topology belonging to the axon– that is, it fetches all synapses that are associated with this axon including their weight and their target neuron. For each connecting synapse, we accumulate its membrane potential locally inside a *membrane cache* before writing it back to memory. Note that the calculated membrane voltage is not for the current timestep, but rather for the next one (double-buffering).

The main design strategy we employed here is that everything except synaptic data, current membrane and topology information is held locally, either in BRAM or BROM. Furthermore, because Kernel_2 updates information of the next time-step (and not the current), it allows us to run both kernels in parallel to each other, maximizing the amount of work that can be done.

4. Experimental Setup

We evaluated our design on the Stratix V DE5-Net board, which features one 5SGXEA7N2F45C2 FPGA device as well as two independent banks of DDR3 memory capable of delivering a total of 1024 bits of data per cycle (@200 MHz). We used Intel's High-Level Synthesis compiler for OpenCL version 16.1.2 to compile our design, while using Python version 3.6 and GNU Compiler version 4.8.5 for the front-end.

We compared our design against that of NEST 2.12.0 [9], compiled with GNU Compiler version 4.8.5. NEST is a well-known and well-maintained simulator for spiking neural networks on general-purpose processors. We executed NEST on three generations Intel Xeon machines, ranging from E5-2670 to E5-2650v4, as well as one Xeon PHI 7210f. Table 1 lists the details for these machines. We tested both using a single thread per physical core and with hyper-threading enabled. All machines had CentOS Linux release 7.3.1611 with kernel version 3.10.0-514.16.1 running on them.

To benchmark our design, we created a custom network that tries to isolate various performance aspects of both our design and NEST. The network we create is fully connected, meaning that each neuron has a synaptic connection to each other neuron in in the network. Furthermore, we set the weights for each synapse to be ˜0 mV and control network activity using a per-neuron constant current that forces the neuron to self-oscillate. The frequency of this self-oscillation is controlled by the magnitute of the constant current. We vary the intensity of this self-oscillation in order to isolate how well each framework copes with spike activity. Furthermore, we also vary the size of the network while keeping spike activity between 4 and 70 Hz, which represents the "natural" frequencies in the brain [16]: 3.5-7 Hz (Theta), 8-12 Hz (Alpha), 13-24 (Beta) and 24-70 Hz (Gamma). Figure 3 visualizes one of the example networks and its activity through a spike plot.

Table 2 Parameters to the evaluated network

Parameter	
Neurons	1000 - 15000
Synapses	10^6 - 2.2510^8
Connections	all-to-all
Synaptic Weight	˜0 mV
Axon delay	1.0 ms
I_e	3.8pA - 129.85pA
V	init to random
U	init to random
a, b, c, d	0.02, 0.2, -65.0, 8.0
Sim. resolution	0.1 ms
Sim. length	1000 ms

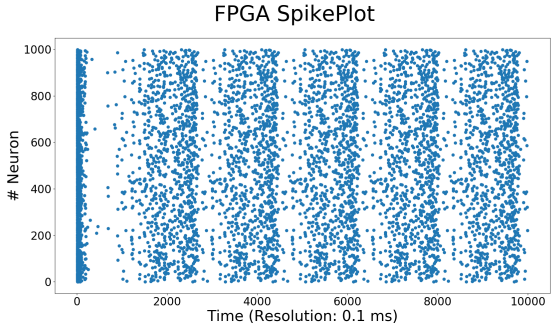

Fig. 3 Spikeplot for one of the example networks, showing 1000 simulated neurons (y-axis) and the the time when they spike (x-axis). Eac neuron spikes on average six times per second (6 Hz).

The network parameters are shown in Table 2. For all benchmarks we simulate 1000 milliseconds of neural "real life" activity. For both NEST and our framwork we sample the spiking activity in all neurons of the simulation. We also initialized the Izhikevich specific parameters V and U to random; we did this to avoid having the all neurons spike at the same time. We only time the kernel execution itself and do not include the time for rebuilding the network nor any transfer overheads.

5. Results

The performance of NEST with the various general-purpose processors and our FPGA implementation is seen in Figure 4. The figure plots the total execution time for each of the processor configuration without (left-side) and with (right-side) hyper-threading enabled. Each of the four graphs represents various spiking activity, ranging from low-intensity theta frequencies to fairly high-intensity gamma frequencies.

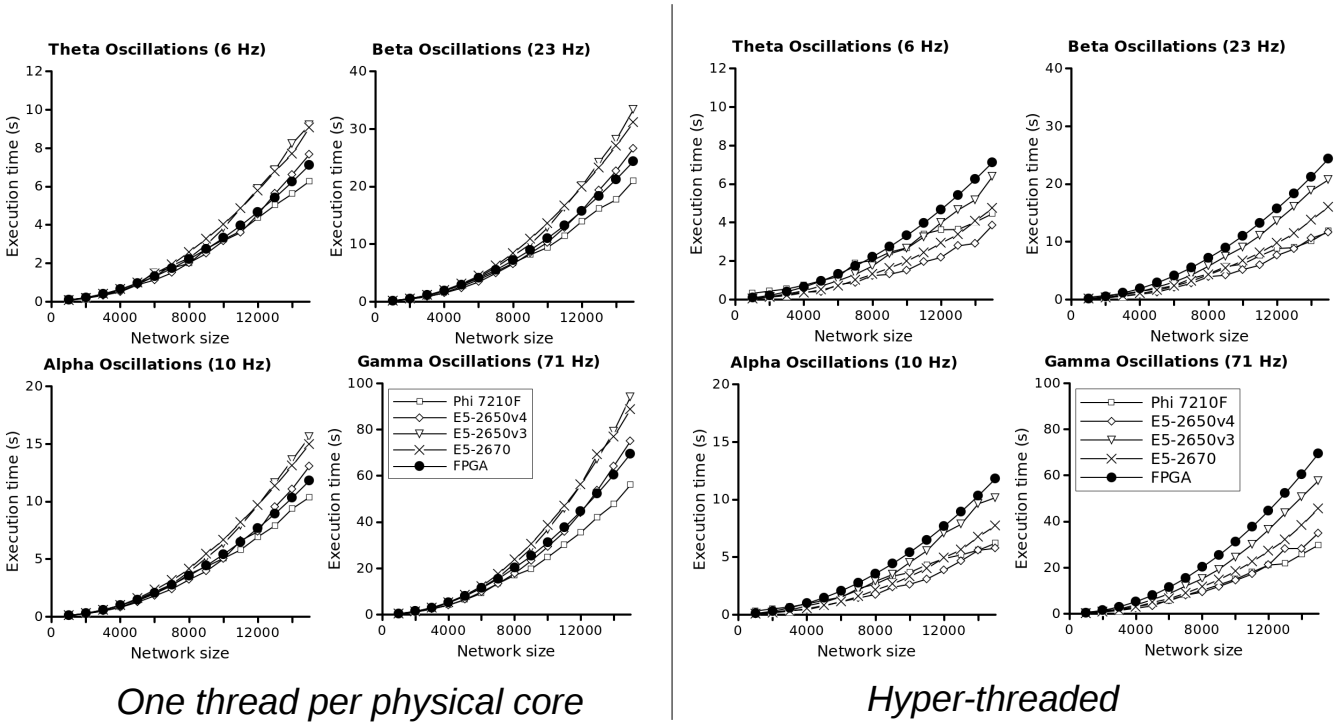

Fig. 4 Performance evaluation of the various networks sizes (x-axis) and the measured execution (y-axis) times with different processors without (left) and with (right) hyper-threading, as well as our FPGA implementation. Note the large impact that network activity has on the execution time.

Overall we note that the general-purpose processor performance intuitively correlates with the lithography technology– the smaller the technology the more recent the processor is and the better it performs. There is one notable exception: the older Sandy Bridge-based E5-2670 outperforms the newer Haswell-based E5-2650 V3 on larger networks, despite having fewer physical cores and being manufactured in 10 nm larger technology.

Enabling hyper-threading seems to always result in an increase in performance, albeit it shifts the performance ranking: the Xeon Phi 7210F is the best performing when hyper-threading is disabled. With hyper-threading activated, the performance of the Xeon PHI is slightly lower than that of the Broadwell-based E5-2650v4 for networks with low spiking activity (theta, alpha, beta), while still being the the best performing on networks with high spiking activity (gamma).

When hyper-threading is enabled, our FPGA design is on average 28% slower than E5-2650v3, 44% slower than Phi 7210F, 70% slower than E5-2670 and almost twice as slow as the E5-2650v5 (96.8 %). The gap between our FPGA design and Phi 7210F is less on smaller networks (16% on networks below 8000 neurons) and greater on larger networks (75% slowdown on networks greater than 8000 neurons). The opposite holds true for the other processors, where our FPGA device performs better with increase network size. For example, our FPGA suffers only a 18% reduction in performance compared E5-2650v3 when considering networks larger than 8000 neurons in size.

When hyper-threading is disabled, our FPGA design performs on average better than both E5-2650v4 and E5-2670, while being faster than E5-2650v4 only when the network-size is larger than on-average 8000 neurons. The Phi 7210F is consistently faster than our device.

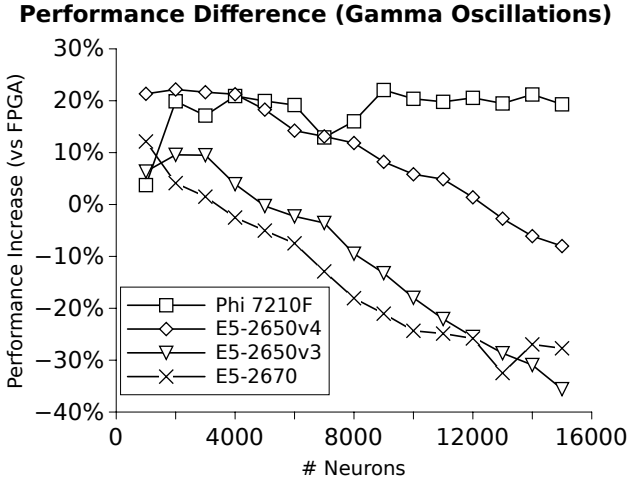

Fig. 5 Performance of the different design normalized to our FPGA design, showing a trend where the general-purpose processor's performance degrades less gracefully than our FPGA design with the network size.

We also found that the difference between the our FPGA design and the general-purpose processors *decreases* with increased network size. We see this behavior with and without hyper-threaded enabled, albeit it is more pronounced with hyper-threads disabled. Figure 5 shows this trend on one of the configurations. Here we see the how performance change for each of the CPUs compared to our FPGA design when the network size increases. We see that

Table 3 Resource utilization and occupancy of our FPGA architecture

Parameter	
ALUTs	141,007 (30%)
Flip-Flops	247,523 (26%)
RAM Blocks	946 (37%)
DSPs	144 (56%)
f_{max}	258.26 MHz

on smaller network sizes, the CPUs generally outperforms the FPGA. However, as the network size increases, the performance of our FPGA degrades more gracefully than the performance of the CPUs.

We suspect this *decrease* in general-purpose processor performance is due to cache effects, albeit this remains to be shown in a future study. Future work will also focus on larger network sizes in order to see for how long this trend in decreased CPU performance continues.

We conclude by presenting our synthesis results for our FPGA design, shown in Table 3. We are using a bit more than half of the available DSP resources, while using a third of all other resources. The design runs at a frequency of 258 MHz.

6. Related work

Several FPGA designs strive to keep all data on-chip. These approaches typically have reasonable performance (due to data being on-chip), but are limited in capacity in terms of numbers of neurons or the topology (synaptic connections) of the network. Some other limitations among these architecture includes shared bus for communication [7], randomized parameters [23] to reduce BRAM pressure, few neurons simulated (≤4000) [24], no time-sharing of processing elements [4]. Of particular interest in this group is the work by Smaragdos et al. [25], which (as we do) used a HLS tool (Xilinx Vivado HLS) to assist hardware generation. Their design did however only support network sizes up-to 96. An interesting all-reduce, ring-based design for membrane potential was designed back in 2009 [26] in order to distribute synaptic computation in parallel. The architecture designed by Pani et al. work [27] dedicates an exclusive wire per neuron axon inside the FPGA– such an architecture can only hande up to 1440 neurons (Xilinx Virtex-6). The architecture in [28] is based on a large number of FSMs, each controlling an ALU and the ALUs communicating with each other through a shared bus; the aim of the work is to increase accuracy rather than performance and only a few number of neurons were evaluated.

Those FPGAs that exploit external memory typically can handle a larger capacity of neurons. The general strategy (as is the case in the present paper), is to keep topology data (synapses and their connectivity) in external memory while keeping neuron parameters and state on-chip. Such designs have been implemented in Bluehive [29] and in the design created by Cheung et al. [30]. The latter is of particular interest because they used a HLS tools (Maxeller) to synthesize their design. Unfortunately they only estimated the peak performance of their design. Minitaur [8] is a FPGA architecture that consists of several processing elements that simulates the neurons in parallel; a scheduler schedules neuron onto the PEs, which contain synaptic data, neuron parameters and neuron state in BRAM. The Minitaur does however operate at a very low frequency (75 MHz).

Other FPGA works focus more on the feasibility or practically of using FPGAs on (usually) small scale demonstrations. These include: Image recognition through MNIST [7], [8], the XOR-problem [24], pattern classification [4] and Leech heartbeats [31], [32].

7. Conclusion and Future work

We have introduced, designed and evaluated a FPGA-based architecture capable of simulating SNNs. Our architecture leverages Intel's OpenCL HLS tools to assist in creating the hardware. We empirically compared the performance to that of NEST. We showed that the performance of our FPGA is comparable to the Xeon family of processors when hyper-threading is disabled, while capable of being as low as 18% slower with hyper-threading activated, depending on neural network size. Furthermore, we found our design's performance to degrade more gracefully with increasing network size compared to the general-purpose processors.

We are currently working on extending our design to support many more neural model, including the Hodgkin-Huxley [18], Morris-Lecar [19] and the generic Integrate&Fire model. We expect that each of the different models will place unique constraints on our architecture. We are also looking into supporting synaptic plasticity (time-dependent or otherwise), which the current work does not include. Finally, in the future we will look at more recent devices such as Intel's Arria 10 and Xilinx Ultrascale Kintex+ to further accelerate SNNs.

Acknowledgments This research has been funded by the Japan Society for the Promotion of Science (JSPS) under grant ID P-16764.

References

[1] Maass, Wolfgang: Networks of spiking neurons: the third generation of neural network models, *Neural Networks*, vol. 10, nr. 9, pp: 1659–1671

[2] Song, Sen and Miller, Kenneth D and Abbott, Larry F: Competitive Hebbian learning through spike-timing-dependent synaptic plasticity, *Nature neuroscience* vol. 3, nr. 9, pp. 919–926 (2000).

[3] Iakymchuk, Taras and Rosado-Muñoz, Alfredo and Guerrero-Martínez, Juan F and Bataller-Mompeán, Manuel and Francés-Víllora, Jose V: Simplified spiking neural network architecture and STDP learning algorithm applied to image classification, *Springer EURASIP Journal on Image and Video Processing*, vol. 2015, nr. 1, pp: 1–11

[4] Iakymchuk, Taras and Rosado, Alfredo and Frances, Jose V and Batallre, Manuel: Fast spiking neural network architecture for low-cost FPGA devices, *IEEE 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, pp: 1–6 (2012)

[5] Antonietti, Alberto and Casellato, Claudia and Garrido, Jesús A and Luque, Niceto R and Naveros, Francisco and Ros, Eduardo and DAngelo, Egidio and Pedrocchi, Alessandra: Spiking neural network with distributed plasticity reproduces cerebellar learning in eye blink conditioning paradigms, *IEEE Transactions on Biomedical Engineering*, vol. 63, nr. 1, pp: 210–219 (2016)

[6] Gamez, David and Fountas, Zafeirios and Fidjeland, Andreas K: A neurally controlled computer game avatar with humanlike behavior, *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, nr. 1, pp: 1–14

[7] Rice, Kenneth L and Bhuiyan, Mohammad A and Taha, Tarek M and Vutsinas, Christopher N and Smith, Melissa C: FPGA implementation of Izhikevich spiking neural networks for character recognition, *International Conference on Reconfigurable Computing and FPGAs*, pp. 451–456 (2009)

[8] Neil, Daniel and Liu, Shih-Chii: Minitaur, an event-driven FPGA-based spiking network accelerator, *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, nr. 12, pp: 2621–2628 (2014)

[9] Marc-Oliver Gewaltig and Markus Diesmann: NEST (NEural Simulation Tool), *Scholarpedia*, vol. 2, nr. 4, pp: 4 (2007)
[10] Hines, Michael L and Carnevale, Nicholas T: The NEURON simulation environment, *NEURON*, vol. 9, nr. 6 (2006)
[11] Goodman, Dan and Brette, Romain: Brian: a simulator for spiking neural networks in Python, (2008)
[12] Khan, Muhammad Mukaram and Lester, David R and Plana, Luis A and Rast, A and Jin, Xin and Painkras, Eustace and Furber, Stephen B: SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor, *IEEE International Joint Conference on Neural Networks*, pp: 2849–2856 (2008)
[13] Fidjeland, Andreas K and Roesch, Etienne B and Shanahan, Murray P and Luk, Wayne: NeMo: a platform for neural modelling of spiking neurons using GPUs, *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp: 137–144 (2009)
[14] Merolla, Paul A and Arthur, John V and Alvarez-Icaza, Rodrigo and Cassidy, Andrew S and Sawada, Jun and Akopyan, Filipp and Jackson, Bryan L and Imam, Nabil and Guo, Chen and Nakamura, Yutaka et al.: A million spiking-neuron integrated circuit with a scalable communication network and interface, *Science*, vol. 345, nr. 6197, pp: 668–673 (2014)
[15] Zohouri, Hamid Reza and Maruyamay, Naoya and Smith, Aaron and Matsuda, Motohiko and Matsuoka, Satoshi: Evaluating and optimizing opencl kernels for high performance computing with FPGAs, *IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp: 409–420 (2016)
[16] Başar, Erol and Başar-Eroglu, Canan and Karakaş, Sirel and Schürmann, Martin: Gamma, alpha, delta, and theta oscillations govern cognitive processes, *International journal of psychophysiology*, vol. 39, nr. 2, pp: 241–248 (2001)
[17] Diehl, Peter U and Cook, Matthew: Unsupervised learning of digit recognition using spike-timing-dependent plasticity, *Frontiers in computational neuroscience* vol. 9 (2015)
[18] Hodgkin, Alan L and Huxley, Andrew F: A quantitative description of membrane current and its application to conduction and excitation in nerve, *The Journal of physiology* vol. 117, nr. 4, pp: 500 (1952)
[19] Morris, Catherine and Lecar, Harold: Voltage oscillations in the barnacle giant muscle fiber, *Biophysical journal*, vol. 35, nr. 1, pp: 193–213 (1981)
[20] Izhikevich, Eugene M: Simple model of spiking neurons, *IEEE Transactions on neural networks* vol. 14, nr. 6, pp. 1569–1572 (2003).
[21] Izhikevich, Eugene M: Which model to use for cortical spiking neurons?, *IEEE Transactions on neural networks* vol. 15, nr. 5, pp. 1063–1070 (2004).
[22] Davison, Andrew P and Brüderle, Daniel and Eppler, Jochen and Kremkow, Jens and Muller, Eilif and Pecevski, Dejan and Perrinet, Laurent and Yger, Pierre: PyNN: a common interface for neuronal network simulators, *Frontiers in neuroinformatics*, vol. 2 (2008)
[23] Wang, Runchun and Hamilton, Tara Julia and Tapson, Jonathan and van Schaik, Andre: An FPGA design framework for large-scale spiking neural networks, *IEEE International Symposium on Circuits and Systems*, pp: 457–460 (2014)
[24] Wan, Lei and Luo, Yuling and Song, Shuxiang and Harkin, Jim and Liu, Junxiu: Efficient neuron architecture for FPGA-based spiking neural networks, *IEEE Irish Signals and Systems Conference*, pp: 1–6 (2016)
[25] Smaragdos, Georgios and Isaza, Sebastian and van Eijk, Martijn F and Sourdis, Ioannis and Strydis, Christos: FPGA-based biophysically-meaningful modeling of olivocerebellar neurons, *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp: 89–98 (2014)
[26] Cheung, Kit and Schultz, Simon R and Leong, Philip HW: A parallel spiking neural network simulator, *IEEE International Conference on Field-Programmable Technology*, pp: 247–254 (2009)
[27] Pani, Danilo and Meloni, Paolo and Tuveri, Giuseppe and Palumbo, Francesca and Massobrio, Paolo and Raffo, Luigi: An FPGA platform for real-time simulation of spiking neuronal networks, *Frontiers in Neuroscience*, vol. 11 (2017)
[28] Zhang, Yiwei and Mcgeehan, Joseph P and Regan, Edward M and Kelly, Stephen and Nunez-Yanez, Jose Luis: Biophysically accurate foating point neuroprocessors for reconfigurable logic, *IEEE Transactions on Computers*, vol. 62, nr. 3, pp: 599–608 (2013)
[29] Moore, Simon W and Fox, Paul J and Marsh, Steven JT and Markettos, A Theodore and Mujumdar, Alan: Bluehive-a field-programable custom computing machine for extreme-scale real-time neural network simulation, *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp: 133–140 (2012)
[30] Cheung, Kit and Schultz, Simon and Luk, Wayne: A large-scale spiking neural network accelerator for FPGA systems, *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp: 113–120 (2012)
[31] Moctezuma, Juan Carlos and McGeehan, Joseph P and Nunez-Yanez, Jose Luis: Biologically compatible neural networks with reconfigurable hardware *Elsevier Microprocessors and Microsystems*, vol. 39, nr. 8, pp: 693–703 (2015)
[32] Ambroise, Matthieu and Levi, Timothée and Saïghi, Sylvain: Leech Heartbeat Neural Network on FPGA, *Springer Conference on Biomimetic and Biohybrid Systems*, pp: 347–349 (2013)