Prototyping a Biologically Plausible Neuron Model on a Heterogeneous CPU-FPGA Board

Kaleb Alfaro-Badilla[*], Alfonso Chacón-Rodríguez[*], Georgios Smaragdos[‡], Christos Strydis[‡], Andrés Arroyo-Romero [*], Javier Espinoza-González[*], and Carlos Salazar-García [†]
[*]Escuela de Ingeniería Electrónica, Tecnológico de Costa Rica
[†]Área Académica de Ingeniería Mecatrónica, Tecnológico de Costa Rica
[‡]Dept. of Neuroscience, Erasmus Medical Center, Rotterdam, The Netherlands
Email: {jaalfaro, aarroyo23r, jaespinoza}@estudiantec.cr
{csalazar, alchacon}@tec.ac.cr {c.strydis, g.smaragdos}@erasmusmc.nl

***Abstract*—A heterogeneous hardware-software system implemented on an Avnet ZedBoard Zynq SoC platform, is proposed for the computation of an extended Hodgkin Huxley (eHH), biologically plausible neural model. SoC's ARM A9 is in charge of handling execution of a single neuron as defined in the eHH model, each with a $O(N)$ computational complexity, while the computation of the gap-junctions interactions for each cell is offloaded on the SoC's FPGA, cutting its $O(N^2)$ complexity by exploiting parallel-computing hardware techniques. The proposed hw-sw solution allows for speed-ups of about 18 times vis-à-vis a vectorized software implementation on the SoC's cores, and is comparable to the speed of the same model optimized for a 64-bit Intel Quad Core i7, at 3.9GHz.**

***Index Terms*—Biologically accurate neural networks models, systems on chip, hardware-software co-design, high level synthesis, heterogeneous systems, spiking neural networks.**

I. Introduction

Biologically accurate brain simulation is a very useful for neuroscientific research often offering better understanding of brain behavior without having to resort to in-vivo or in-vitro experimentation, but also pointing to the future development of seamless brain-rescue devices for medical applications. There are plenty of mathematical models of biological neural networks (NNs) that accurately represent both cells and their interactions. Spiking Neural Networks (SNNs) –for the spiking nature of their output response–, is one of the many modeling formalisms used to represent biological NNs, because they not also model output neural firing rates, but are also able to produce complex spiking patterns as those seen in biological neurons (for instance: different modes of threshold variation, various types of spike bursting, sub-threshold oscillations, etc.; see [1] for greater detail).

This representational complexity often requires extensive computational and communication resources. In [1], one can find a detailed analysis of the most common models, tested in large-scale simulations of cortical neural networks, evaluated both in terms of computational resources, their biological accuracy and biophysical meaningfulness (that is, their relation to real, measurable biological, chemical, and physical phenomena). Efforts at accelerating these models include the use of high-performance multicore data centers, arrays of GPUs for vectorized processing, and dataflow engines, exploiting different parallel-processing techniques. Yet, these approaches still do not offer real-time execution for meaningful SNN sizes ($>$100 cells), and are expensive in terms of power [2].

While some have advocated the use of digital and mixed-signal ASICs [3] due to their high data performance and inherent low power consumption, FPGAs come as a compromising solution, being intrinsically reconfigurable, and already equipped with high-speed serial and memory interfaces, and with a much lower implementation cost. Yet, as of now, most of the approaches using FPGAs in the literature seem to opt for brute-force approaches, where the SNN algorithm is completely uploaded onto the programmable logic, using either HDL RTL design or high level synthesis tools (see an example in [3]). Certainly, in the FPGA one can improve data-processing micro-architectures and data-routing techniques that take advantage of the inherent concurrent nature of custom digital circuits [4], and integrate clusters of such devices able to handle big SNNs (see [5]). And one can try using relatively cheap heterogeneous boards such as Avnet's Zynq 7020-based ZedBoard, building a standard HPC framework using MPI, as in [6]. Nonetheless, all of the studied approaches follow a sub-optimal strategy by downloading the whole neuronal model on the FPGA's resources. First, the size of the network has a linear impact on memory capacity, as shown later (and RAM is always limited inside FPGAs). Second, due to the basic routing programmability of a typical, RAM-based FPGA, FPGAs are not capable of clocking at the maximum speed provided by the CMOS node in which they are fabricated. As an example, the ARM core on a Zynq 7020 runs at 667MHz, while designers will consider themselves lucky at achieving more than 200MHz in that chip's FPGA fabric.

In this paper, we present a heterogeneous hardware-software co-designed implementation of a biophysically meaningful SNN model using single-precision floating-point (SPFP) arithmetic, running on an Avnet's Zynq 7020 Zedboard. This application simulates an essential area of the olivocerebellar system, the inferior olivary nucleus (ION), using an eHH conductance-based model. By applying results of the prelim-

978-1-7281-0453-9/19/$31.00 ©2019 IEEE

inary study of computational complexity of the eHH model given in [3], and some experimentation on partitioning the algorithm and its implementation, i.e., exploring what to run in the SoC's cores, and what on the SoC's FPGA fabric, and evaluating the use of a bare-metal approach versus the use of an operating system (OS), to name a few of the options explored, the proposed system offers: 1) a balanced HW/SW co-design that takes advantage of several features of the Zynq platform (DMA and cache access from the FPGA fabric, programming and data downloading via TCP-IP and hardware resources administration via OS), 2) a C++ flexible design that is portable to other Zynq platforms with more processing power, 3) a noticeable improvement in processing speed (over 18 times when compared with the same algorithm, optimized with NEON instructions, running single-threaded on the Zynq's ARM A9).

The paper is organized as follows: in Section II, an overview of the eHH model of the ION is given, along with some sketches on its computational complexity. Section III offers a description of the hardware-software heterogeneous co-design strategy followed in this work. Section IV shows results of the proposed system's performance. Conclusions and future steps are given in Section V.

II. The Inferior Olivary Nucleus eHH Model: Computational Load Analysis

The olivocerebellar system is a dense brain region which plays an important role in sensorimotor control, and the ION belongs to its circuitry [7]. The eHH, one of the most used biophysically meaningful models describing the ION, was developed by de Gruijl et al. [8], extending Hodgkin and Huxley's standard model [9]. Here, neurons are divided into three compartments —dendrite, soma and axon— modelled by four non-linear basic equations and dozens of parameters representing the conductance-based electrical activity of the cell. These compartments resemble the biological parts of a neuron, each with a state that holds the electrochemical variables determining each compartment's potential in a concurrent way. The eHH integrates as well a dendritic-compartmental sub-unit that models the influence on any neuron of its neighbor cells through their biological gap-junctions (called the Gap-Junction Unit (GJU) in this paper; see [3]). This non-linear system is solved in $50\mu s$ lock-step discrete computations, using SPFP arithmetic, without a noticeable impact in accuracy (see [10]). The number of total SPFP operations required is given in Table I. The SPFP operations related to any cell compartment scale linearly with the network size. But, in the case of the the GJU, the nested nature of computing the effect on each neuron of all its neighboring cells increases its computational cost in a $O(N \times C)$ rate, where N is the cell population of the network and C is the number of connections for each cell.

The worst case is when N=C, as complexity becomes $O(N^2)$. Now, the ION shows high degrees of connectivity, and implementing such a network scaffold in hardware is only practical when implementing for all-to-all connectivity (otherwise, a multiple bitstreams with different connectivity

Table I
Single-Precision Floating-Point (SPFP) operations and transfers per simulation step, per neuron. Data taken from ([10] [11])

Operational compartment	**No. of SPFP operations**
Gap junction unit	12 (per connection)
Cell compartent (Axon, soma, dendrite)	859
I/O and storage	**No. of SPFP transfers**
Neuron state (R/W)	19
Evoked input (R)	1
Connectivity vector (R)	1 (per connection)
Neuron conductances (R)	20
Axon output (W)	1 (axon voltage)

densities are required for better usage of resources). The bottleneck of simulation is then determined by its interconnection complexity. Memory requirements behave similarly. At each simulation step, neuron states are read, updated and written back to memory. The amount of memory reads increases on a $O(N^2)$ basis if N=C, and memory writes increment on a $O(N)$ basis. To complement on the decision on what to port to the hardware, a profiling of the C code of the eHH was carried on, running on a single thread in the ZedBoards's ARM A9, at SPFP. Results are shown in Fig. 1. Notice the gap-junctions computational load impact on the performance, as the network interconnection density grows. Without such load, the dendrite's computation time falls under that of the other two compartments. For realistic network sizes (over 100's of cells), the GJU is then the costliest function), with $O(N^2)$ complexity, while the other compartment functions are $O(N)$. Results agree with reports in the literature (see [3]).

III. Proposal of a heterogeneous approach for implementing the ION model

For the system's implementation, an Avnet's ZedBoard development board was chosen because of its price and availability. The ZedBoard has a Zynq-7020 SoC and 512MB DDR3 RAM, besides several I/O sub-systems. The processing region of the Zynq (called PS) includes a dual core ARM A9 CPU, with NEON SIMD capabilities, plus several I/O controllers. The integrated Artyx-7 FPGA fabric (called the PL region) is interconnected via several channels to the PS. The problem consists now in determining what processing elements could take care of what sections of the model in a more efficient way. Distributing tasks is the first obvious answer (for instance, using MPI), as it would provide flexibility and some scalability.

Secondly, based on the prior complexity analysis, the GJU was ported to the SoC's PL, keeping the rest of the operations embedded in the PS. An overview of the proposed architectural solution is given in Fig. 2. The GJU is mapped via AXI-Lite to the PS for programming and monitoring. Calculation data is transferred via AXI4-DMA from the DDR3 RAM.

As seen in Listing 1, the computation of the dendritic currents for each cell (`Iout[N]`, requires the traversing for each neuron of the network of a $N \times N$ conductance matrix `Conn[N][C]`). This matrix defines which cells in the

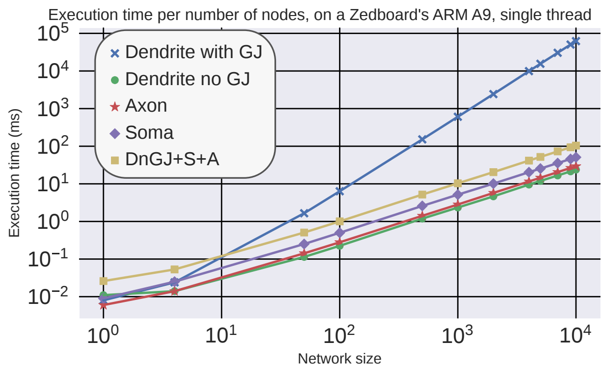

Figure 1. ZedBoard's ARM A9 execution time per compartmental model of the eHH, SPFP, single-threaded implementation. The *DnGJ+S+A* curve plots the sum of the three compartments, excluding the GJ. The soma compartment dominates, once the GJU is excluded from the dendrite compartment.

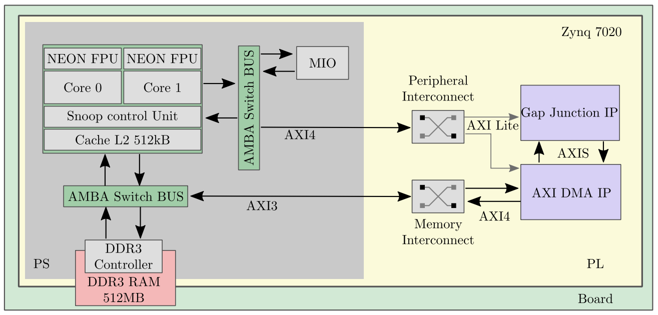

Figure 2. Overall system architecture. The GJU is managed by one thread of the ARM A9, via AXI4, while the other thread resolves software calculations and I/O outside of the ZedBoard.

$N \times N$ sized network share synaptic connections and thus computation also requires as inputs the dendrite voltages of each neighboring cell, `Vdend[N]`.

Listing 1
GAP-JUNCTION COMPUTATION PSEUDOCODE

```
float Vdend[N]; float Iout[N]; float Conn[N][C]; float facc=0; float vacc=0;
for(indx_neu=0; indx_neu<N; indx_neu++){
    for(indx_con=0; indx_con<C; indx_con++){
        v=Vdend[indx_neu]-Vdend[indx_con];
        f=v*expf(-v*v*0.01);
        facc+=Conn[indx_neu][indx_con]*f;
        vacc+=Conn[indx_neu][indx_con]*v;
    }
    Iout[indx_neu]=0.8*facc+0.2*vacc;
    facc=0;vacc=0;
}
```

Now, instead of loading the complete conductance matrix along with the dendrites' output voltages into the GJU, some data restructuring is carried on. This subdivision of the data allows for more efficient DMA block transfers, and also makes hardware optimization of the GJ operations easier (as more logic, DSPs blocks, FFs and BRAM become available for data processing, instead of being occupied with data tables). Let the interaction among neurons be defined by a conductance matrix $\mathbf{C} \in \mathbb{R}^{\mathbf{N}\times\mathbf{N}}$ where $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N]$, and with each dendrite's output voltage stored in a vector as $\mathbf{v} \in \mathbb{R}^{\mathbf{N}}$. Let us define now a vector storing the potential difference between a particular dendrite and all of its neighbors,

$$\mathbf{v}_{\text{diff}}^{(i)} := \mathbf{v}[i]\vec{1} - \mathbf{v} \tag{1}$$

where $\vec{1} = [1, \ldots, 1]$. For each i neuron, one can create a vector $\mathbf{f}^{(i)} \in \mathbb{R}^N$ where each element $\mathbf{f}^{(i)}[j]$ is paired with an element $\mathbf{v}_{\text{diff}}^{(i)}[j]$ as

$$\mathbf{f}^{(i)}[j] = \mathbf{v}_{\text{diff}}^{(i)}[j] \exp\left[-(\mathbf{v}_{\text{diff}}^{(i)}[j])^2/100\right] \tag{2}$$

One can now find the vector storing all the currents generated in each dendrite by the synaptic influence of its neighbors as defined by the eHH model, such that

$$\mathbf{i}_{\text{GJ}} = [I^{(1)}, I^{(2)}, \ldots, I^{(N)}] \in \mathbb{R}^{\mathrm{N}} \tag{3}$$

where

$$I^{(i)} = 0.8 f_{\text{acc}}^{(i)} + 0.2 v_{\text{acc}}^{(i)} \tag{4}$$

$$f_{\text{acc}}^{(i)} = \mathbf{f}^{(i)} \cdot \mathbf{c}_i' \tag{5}$$

$$v_{\text{acc}}^{(i)} = \mathbf{v}_{\text{diff}}^{(i)} \cdot \mathbf{c}_i' \tag{6}$$

Note that the dot product in (5) and (6) can be divided in subsets of products of some arbitrary number of M samples. If one defines the amount of sub-products as $K := \text{ceil}(N/M)$, then its recursive sequence for $k \in \{0, \ldots, K-1\}$ is

$$\begin{gathered}
f^{(i)}(-1) = v^{(i)}(-1) = 0 \\
f^{(i)}(k) = \mathbf{f}^{(i)}[kM:(k+1)M] \cdot \mathbf{c}_i'[kM:(k+1)M] + f^{(i)}(k-1) \\
v^{(i)}(k) = \mathbf{v}_{\text{diff}}^{(i)}[kM:(k+1)M] \cdot \mathbf{c}_i'[kM:(k+1)M] + v^{(i)}(k-1) \\
\Rightarrow f^{(i)}(K-1) = f_{\text{acc}}^{(i)}, v^{(i)}(K-1) = v_{\text{acc}}^{(i)}
\end{gathered}$$

Finally, note that the computing of M sub-products $f^{(i)}(k)$ and $v^{(i)}(k)$ share the same input $\mathbf{v}[kM:(k+1)M]$, then if $\mathbf{v}[iM:(i+1)M]$ is available, $\mathbf{V}_{\text{diff}}^{(iM:(i+1)M)}$ can be computed on the fly with each step k. Thus, each element in the sub-products can be executed in parallel and accumulated as

$$\begin{aligned}
I^{(iM:(i+1)M)}(k) = 0.8 f^{(iM:(i+1)M)}(k) + \\
0.2 v^{(iM:(i+1)M)}(k) + I^{(iM:(i+1)M)}(k-1)
\end{aligned} \tag{7}$$

where $I^{(iM:(i+1)M)}(k)$ can be understood as the processing of sub-blocks $\mathbf{c}'_{(iM:(i+1)M)}[kM:(k+1)M]$ of size $M \times M$ and subsets $\mathbf{v}[kM:(k+1)M]$ of length M. Thus, after K sub-blocks and sub-vectors from $\mathbf{v}$ are processed, the final result are M values $I^{(iM:(i+1)M)}$. Some processing overhead is required in order to create the sub-vectors of the conductance matrix ($\mathbf{c}_i'$) and the dendrite's voltages, and sorting the final results. Besides, the DMA block size ought not be so small such that DMA transferences become impractical: an optimum block size need be determined here, but this is a work still in progress. In the meantime, a short exploration showed that an 8×8 sub-block size produced the best results here presented.

Operations were coded in C++ and synthesized using Xilinx's Vivado HLS tools. Hardware optimization included unrolling loops and introducing pipelining so to bring up the clock speed of the GJU to 100MHz, with a latency of only 416 clock cycles per sub-block. The utilization summary from the implementation indicates that only 38.18% of LUT, 18.75% of FF, 4.56% of BRAM and 24.09% of DSP resources are used, meaning there is still room for speed improvement).

IV. Software integration and performance evaluation

The complete system was implemented on a ZedBoard with the PetaLinux 2016.4 toolkit, with the GJU packed as an IP by the Vivado HLS compiler, and connected as described in Fig. 2. A memory space on the kernel was reserved for DMA, allocating a non-coherent 10MB memory map (coherency is manually managed, and still needs to be worked out completely: fixing this could cut latency further down). The DMA driver allocates a memory-mapped char device for the user-space application. DMA in the PL is handled by a Xilinx AXI-DMA IP. Figure 3 compares the execution-time of the proposed system (*sw hw accel*), for networks of different sizes, against C code running on the single-threaded 32-bit Zynq's ARM A9 @666MHz (*sw*), a single-threaded NEON SIMD optimized version on the same core (*sw simd*), a Quad Core Intel 64-bit i7-7820HQ @3.9GHz running the AVX2 SIMD optimized code on a single thread (*i7 sw simd*), and data from an eHH HLS version (*virtex hw*) completely ported to a Virtex XC7VX485T @100Mhz, taken from [3]. The improvement is of at least one magnitude order, when comparing the *sw hw accel* solution against the *sw simd* option, and it is equivalent to the *i7 sw simd* option. The implementation, nonetheless, underperforms by one magnitude order against what's reported in [3], but the latter running on a 10 times more expensive Virtex-7 board, and without scalability outside the board. Notice, as well, that for small problem sizes of neurons, CPU-FPGA communication overheads (DMA access, data sorting, AXI-Lite and cache accesses), dominate execution times, as shown by the breakdown between memory transaction time (*trans overhead*) and useful computation time (*hw exec time*). This means that this implementation is better suited for large neuron populations.

V. Conclusions

A heterogeneous implementation of the eHH model on an Avnet Zynq-7020 ZedBoard has been presented. Results show that this implementation improves over 18 times the execution time over the same model running on the ZedBoard's ARM A9. This implementation reaches equivalent times when compared with a top-of-the-line Intel 64-bit i7 processor, and though lagging behind a complete HLS porting of the model to a state-of-the-art FPGA, it points to a potentially more efficient and scalable solution for large SNNs, particularly if a more powerful board is used (the high-level specification of the proposed system makes it easily portable). Future works include finding an optimum sizing of the intermediate data structures used for the GJU, improving hardware timing in the same GJU applying RTL-oriented techniques, and interconnecting the board in a multi-FPGA array with high-speed data-communication channels, in order to either speed up simulations or handle larger networks.

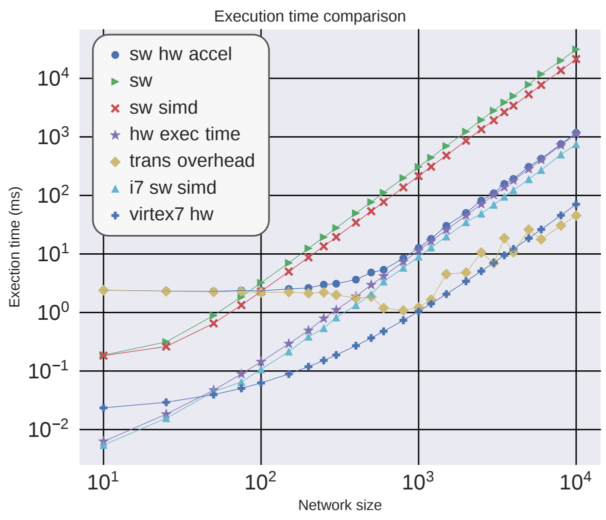

Figure 3. Execution time comparison for the eHH model running: on the proposed implementation (*sw-hw-accel*), on a single-thread of the Zynq's ARM A9 (*sw*), on the same ARM A9 NEON SIMD optimized, (*sw-simd*), on an Intel i7 AVX2 SIMD optimized on a single thread (*i7 sw simd*), and on a Virtex XC7VX485T @100Mhz (*virtex hw*). Note that *sw hw accel* breaks down to *hw exec time* and *trans overhead* curves, meaning that for small networks, the memory transactions time absorbs the useful computation time.

References

[1] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sept 2004.

[2] T. Nowotny, "Flexible neuronal network simulation framework using code generation for NVidia® CUDA," *BMC Neuroscience - BMC NEUROSCI*, vol. 12, pp. 1–2, 07 2011.

[3] G. Smaragdos, S. Isaza, M. F. van Eijk, I. Sourdis, and C. Strydis, "FPGA-based biophysically-meaningful modeling of olivocerebellar neurons," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 89–98.

[4] A. Zjajo, J. Hofmann, G. J. Christiaanse, M. van Eijk, G. Smaragdos, C. Strydis, A. de Graaf, C. Galuzzi, and R. van Leuken, "A real-time reconfigurable multichip architecture for large-scale biophysically accurate neuron simulation," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 2, pp. 326–337, April 2018.

[5] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, April 2012, pp. 133–140.

[6] P. Moorthy and N. Kapre, "Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 68–75.

[7] C. De Zeeuw, F. Hoebeek, L. Bosman, M. Schonewille, L. Witter, and S. Koekkoek, "Spatiotemporal patterns in the cerebellum." *Nature reviews. Neuroscience*, vol. 12, pp. 327–44, 06 2011.

[8] P. Bazzigaluppi, J. De Gruijl, R. Van Der Giessen, S. Khosrovani, C. De Zeeuw, and M. De Jeu, "Olivary subthreshold oscillations and burst activity revisited," *Frontiers in Neural Circuits*, vol. 6, p. 91, 2012.

[9] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 25 – 71, 1990.

[10] G. Smaragdos, G. Chatzikostantis, S. Nomikou, D. Rodopoulos, I. Sourdis, D. Soudris, C. I. D. Zeeuw, and C. Strydis, "Performance analysis of accelerated biophysically-meaningful neuron simulations," *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 1–11, April 2016.

[11] G. Smaragdos, G. Chatzikonstantis, R. Kukreja, H. Sidiropoulos, D. Rodopoulos, I. Sourdis, Z. Al-Ars, C. Kachris, D. Soudris, C. I. D. Zeeuw, and C. Strydis, "Brainframe: A node-level heterogeneous accelerator platform for neuron simulations," *IOP*, vol. abs/1612.01501, 2016.