Performance Analysis of Accelerated Biophysically-Meaningful Neuron Simulations

Georgios Smaragdos*, Georgios Chatzikostantis‡, Sofia Nomikou‡, Dimitrios Rodopoulos‡, Ioannis Sourdis†
Dimitrios Soudris‡, Chris I. De Zeeuw* and Christos Strydis*
*Neuroscience dept., Erasmus Medical Center, The Netherlands
†Computer Science and Engineering dept., Chalmers University of Technology, Sweden
‡MicroLab, National Technical University of Athens (NTUA), Greece

***Abstract*—In-vivo and in-vitro experiments are routinely used in neuroscience to unravel brain functionality. Although they are a powerful experimentation tool, they are also time-consuming and, often, restrictive. Computational neuroscience attempts to solve this by using biologically-plausible and biophysically-meaningful neuron models, most prominent among which are the conductance-based models. Their computational complexity calls for accelerator-based computing to mount large-scale or real-time neuroscientific experiments. In this paper, we analyze and draw conclusions on the class of conductance models by using a representative modeling application of the inferior olive (InfOli), an important part of the olivocerebellar brain circuit. We conduct an extensive profiling session to identify the computational and data-transfer requirements of the application under various realistic use cases. The application is, then, ported onto two acceleration nodes, an Intel Xeon Phi and a Maxeler Vectis Data Flow Engine (DFE). We evaluate the performance scalability and resource requirements of the InfOli application on the two target platforms. The analysis of InfOli, which is a real-life neuroscientific application, can serve as a useful guide for porting a wide range of similar workloads on platforms like the Xeon Phi or the Maxeler DFEs. As accelerators are increasingly populating High-Performance Computing (HPC) infrastructure, the current paper provides useful insight on how to optimally use such nodes to run complex and relevant neuron modeling workloads.**

I. Introduction

For decades, scientists have been fascinated by the methods and computational capabilities of the biological brain. The US National Academy of Engineers has listed the simulation of the human brain as one of the Grand Engineering Challenges [1]. Inspired by the scientific effort, engineers began to copy computational concepts found in the brain, which led to the creation of the first Artificial Neural Networks (ANNs) with the creation of the perceptron [2]. ANNs do not execute commands sequentially like the typical Von Neumann computer, but each node (or neuron) in a neural network is a separate set of functions and they are all evaluated concurrently during execution. The relation between input and output is defined largely by the network size, topology and interconnectivity of the neurons. Interconnectivity could eventually be adaptive, thus, mimicking the behavior of biological systems.

Eventually, more advanced versions of neural-network models were developed, based upon greater understanding of the biological processes. Spiking Neural Networks (SNNs) [3] do not abstractly mimic biological-neuron behavior but outright simulate the computational behavior of brain processes. True to their biological counterparts, SNNs have the ability to encode information using the transfer rate, amplitude and spike-train patterns, which gives them more capabilities than traditional ANNs [4], [5]. As a result, they are currently heavily used to model the complex behavior of biological-brain systems in neuroscientific research.

In-vivo and in-vitro experiments are a traditional and powerful experimentation tool for neuroscience, however they are typically time-consuming while there is always the possibility of the experimental data to become contaminated (from factors such as the effects of anesthesia). Many of the complex brain-system dynamics that define biological behavior are hypothesized and many in-vivo or in-vitro techniques are not always able to provide the means to validate them. Computational neuroscience attempts to use SNN models of various complexities the accuracy of which can provide predictive behavior and insight on those hypothesized dynamics and can exploit them to guide further biological experiments. Long-term advancement in computational neuroscience can hopefully lead to improved medical treatment of brain-related health issues, novel artificial-intelligence applications and groundbreaking computer architectures.

The main challenge with this type of modeling is that the models used typically have great computational or data-transfer demands. In the meantime, it is only through large-scale network sizes and/or real-time simulation that biological dynamics can be properly modeled for certain types of experimentation (e.g. Brain-Machine Interfaces). This is especially the case for advanced, biophysically-meaningful neuron models [6]. In any case, traditional execution of such models on CPUs with generic programming suites (such as MATLAB) or neuromodeling-specific languages (like NEURON or GENESIS), could take a prohibitive amount of time to complete. The introduction of accelerator-based computing solutions in neuroscience can substantially accelerate the research efforts. Given the advent of accelerators in modern High Performance Computing (HPC) infrastructure, it is imperative that such applications are well understood, especially in the context of the accelerating platforms that are utilized. Additionally, as these applications are to be used in scientific research that is very dynamic and many times conducted by non-HPC experts, the goal should not be to over-optimize them, but keeping the programming effort moderate, resulting to short development times, while providing sufficient performance.

In this paper we analyze a class of detailed, brain-modeling applications, known as conductance-based or Hodgkin-Huxley (HH) models. We use a state-of-the-art, biologically meaningful model of the inferior-olive nucleus (InfOli). We isolate 3 realistic use cases for the target application, capturing the most representative instantiations of HH neuron modeling workload. Before delving into the accelerator platforms, we start with a general, platform-independent analysis of the application. Then, the InfOli use cases are ported and evaluated on two accelerators: a many-core processor and a data flow engine (DFE). To the best of our knowledge, this work is the first to attempt a characterization of the class of HH models on state-of-the-art accelerator computing fabrics.

- We define three InfOli-application use cases that reveal different aspects of HH-model requirements and represent realistic cases for neuroscientific computing workloads.
- We perform a detailed performance and scalability evaluation on two state-of-the-art accelerating platforms: an Intel Xeon Phi and a Maxeler Vectis Data-Flow Engine (DFE).
- We compare and contrast the two platforms, in view of the three InfOli workloads considered and comment on the suitability and usability of each accelerator.

The paper is organized as follows: Section II offers a general background on neural modeling, covering types of SNNs that are widely used in computational neuroscience. In Section III, related prior art is discussed. Section IV contains a detailed description of the InfOli model and the three use cases that we explore in this paper. Section V describes the target acceleration platforms and the InfOli implementation details on each one. Section VI presents performance measurements from the target platforms. In Section VII a brief discussion on the results is given. Finally, in Section VIII, conclusions are summarized.

II. Neural Modeling Background

Neuroscientific SNN models vary in complexity and computational/communication demands. The best choice of SNN model depends on the targeted accuracy level and the available execution platform [6]. There are three main categories of SNNs: (A) *Integrate & Fire* models, (B) *Izhikevich* models, and (C) *Conductance(-based)* models.

The simplest version of SNNs are Integrate-and-Fire (I&F) models. They emulate the most basic operation of a biological neuron, which is the integration of spikes and firing using a threshold mechanism. From this most basic version, extensions are derived which add more features to the model's behavior such as the Leaky I&F, I&F-or-Burst [7] and quadratic I&F [8]. I&F models have extremely low computational demands but also have very limited biological plausibility. They are, thus, useful for exploring large-scale network dynamics in relation to the very basic features they can emulate. Izhikevich neurons [9] are a special type of models which – even though they have similar complexity to I&F models – emulate an impressive fraction of the biological-neuron behavior. This model boasts the capability of emulating all possible input/output spiking activity found in its biological counterpart. Although it treats the neuron as a black box, its flexibility permits to create very accurate high-level representations of large-scale, biological-neural-network behavior. If, on the other hand, a researcher seeks to explore the electrochemical characteristics that produce the neuron's response, they require a biophysically-meaningful neuron model, such as Conductance models. These are the simplest *biophysical* representations of a neural cell. They capture closely the electrochemical behavior that produces the neuron activity by modeling the various ion channels observed in biological neurons. The most important conductance-based model is the one presented by Hodgkin and Huxley (HH) in 1952 [9]. HH models make heavy use of differential equations and are quite scalable, making the design of multi-compartmental models possible (the term "compartment" is used for the distinct parts of an accurate white-box neuron representation). The computational complexity of conductance-based models is orders-of-magnitude higher than that of the previously mentioned types, posing a significant challenge for their efficient simulation.

III. Related Work

A growing volume of prior art attempts to port complex models, such as SNNs, onto a variety of platforms. However, biophysically meaningful implementations of conductance models have a limited presence in bibliography, especially for accelerator-based fabrics, such as the ones used in this paper. A notable attempt for real-time cerebellum simulations using I&F models has been proposed by Yamazaki et al. [10] using GPUs. This work, however, uses non-biophysically meaningful modeling for the cerebellar circuit, lacking many of the intricate details of the biological processes leading to the usage of black boxes within the modeling structure. Additionally for HH models, GPU implementations have been shown to be less efficient compared to reconfigurable hardware solutions [11], [12], even though providing notable speedups [13].

The Xeon-Phi platform [14] is very recent, thus, its capabilities for SNNs' or similar applications' simulation and acceleration are not widely explored yet. A promising use reported in the bibliography has been the incorporation of the Phi for convolutional neural networks (CNN) [15], [16]. These however, are artificial neural networks and, are not representing real biological systems. An attempt has also been documented for the porting of HH-based neuron models on a research-grade many-core chip [17].

On the side of reconfigurable hardware, most works have concentrated on porting simpler I&F or Izhikevich models([18], [19]). Moreover, most conductance models accelerated in reconfigurable platforms employ fixed-point arithmetic, which until recently was more straightforward and efficient on FPGAs. Unfortunately, arithmetic issues with fixed-point representation have been detected often in complex conductance models [20]. Beuler et al. [21] have proposed an FPGA-based framework for real-time, single-compartment HH models using floating-point arithmetic. The real-time achievable network was 400 neurons and the system included a GUI for easy experiment configuration. Another FPGA-based approach for inferior olive brain models, able to simulate 96 cells in

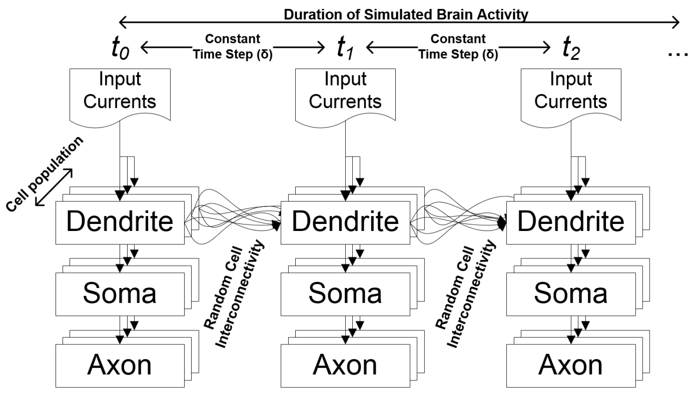


Fig. 1: Representation of the target InfOli model/application.

real-time and using FP arithmetic, was presented in [12]. Such attempts are limited in scope and in-depth analysis of the requirements, relevant to the neuroscientific experiments, thus, fall short of painting the big picture. To the best of our knowledge, only Bhuiyan et al. [22] have made an attempt for a more complete analysis of large-scale SNNs. The design was implemented on a SRC-7 H MAP platform, a software/hardware co-design HPC system that includes a Stratix II FPGA for acceleration. Even though impressive results and optimizations were derived, the models themselves were not analyzed as applications in general but only as implementations on the particular SRC platform. Additionally, the networks were considered only as an image-recognition application and no neuroscientific-experiment instances with biological plausibility were considered.

IV. The Inferior-Olive Model

The InfOli application, which is the focus of this paper, is a model that represents the Inferior-Olive Nucleus. This is an intricate part of the olivocerebellum system, which is one of the most dense brain regions and plays an important role in sensorimotor control. It does not initiate movement by itself but it does provide rhythm and coordination for motor functions. It is considered to be imperative for the instinctive learning and smooth completion of motor actions [23].

The inferior olive provides one of the two main inputs to the olivocerebellum system through the so-called climbing fibers, the other being the mossy fibers. The inferior-olive neurons are also heavily interconnected to one another through direct electrical connections called *gap junctions (GJs)*. The gap junctions define the synchronization behavior between the Inferior-Olive cells and, subsequently, influence the synchronization and learning properties of the overall system [23].

A. Abstract Model Description

The InfOli model considered in this work was originally developed by De Gruijl et al. [24]. It is an extended-HH (eHH) model representation of the inferior-olive cell. It implements a neuron with three distinct compartments, the dendrite, the soma and the axon. Within the dendrite the model also includes the gap junctions which implement the inter-neuron connectivity within the inferior-olive nucleus. It

TABLE I: Neuron requirements per simulation step.

Computation	**FP Operations per neuron**
Gap Junction	475 per connection
Cell Compartment	859
I/O and storage	**FP Variables per neuron**
Neuron States	19
Evoked Input	1
Connectivity Vector	1 per connection
Neuron Conductances	20
Axon Output	1 (Axon Voltage)
Compartmental Task	**% of FP ops for 96 cells**
Soma	13
Dendrite	10
Axon	8
Gap Junction	69

is these gap junctions that complicate the model further and add the term "extended" to the standard HH model. The dendrites represent the cell input stage, the soma is the cell part wherein most of the processing takes place, and the axon represents the cell output stage towards the climbing fibers. In reality, every compartment includes biophysical attributes and, thus, computational processes take place in all three of them and also within each GJ connection itself. The GJs are associated with important aspects of cell behavior as they are not just simple connections; rather, they include significant and intricate electrical processes, which is reflected in their software implementation.

Every compartment includes a number of state parameters denoting its electrochemical state and the *neuron state* as a whole. The neuron state is updated at each simulation step; every new state update is based upon: (i) the previous state of the compartment updated, (ii) the previous state of the other compartments of the same neuron (mainly, the compartment voltages), (iii) the previous state of the dendritic compartment of the neurons to which the updating neuron is connected through the GJs (mainly, the dendritic voltage), and (iv) the externally evoked input to the InfOli, representing the input coming from the rest of the cerebellar circuit into the nucleus.

The three compartments and GJs are evaluated/updated concurrently at each simulation step. The model is calibrated with a simulation time step of $\delta = 50\ \mu sec$. As a result, for a simulation to be able to run the InfOli network with *real-time performance*, every simulation step for the entirety of the network must be completed within $50\ \mu sec$.

Figure 1 depicts a representation of the InfOli model. The GJs are part of the dendritic compartment, thus the compartment receives the extra input coming from the inter-neuron connection. The network works in a step-wise fashion providing discrete output values at every time step which – when aggregated in time – represent the response of each neuron in the network. Depending on the experiment objectives and the kind of behavior required to explore, InfOli implementations can be tuned in terms of (i) the number of simulated cells, (ii) the degree of cell inter-connectivity and (iii) the degree of detail of the simulated GJs.

Inspecting a simple C version of the InfOli application reveals that the GJs have potentially great impact on the total model complexity due to the intense computations required to simulate each one of them. As seen in Table I, the total number of floating-point (FP) operations needed for simulating a single step of a single cell including a single GJ are 1,334. Of those, more than 35% are the operations required just for the GJ. In an N-cell network, if each neuron maintains a constant number of connections C to neighboring cells, the overall GJ computation cost has linear complexity $\mathcal{O}_{gj}(C \times N)$. For realistic experiments, it is not the number of connections C but, rather, the *connectivity density* that is indicative of neuron interconnectivity. That is, the *average percentage of the total neuron inventory to which neuron cells are connected* (measured in % units). As a result, the previous complexity can be expressed as $\mathcal{O}_{gj}(N \times N \times K)$, where K is the connectivity density. It is clear that the worst-case interconnectivity scenario occurs when $K = 1$, i.e. all neurons are connected with all other neurons, whereby the complexity is $\mathcal{O}_{gj}(N^2)$. All remaining, non-GJ computation increases in a linear fashion $\mathcal{O}_{cell}(N)$, as the remainder of the application is of purely dataflow nature. This makes GJ computations the dominating factor in eHH models when GJ functionality is being modeled. This is true even for relatively small-scale networks like, for example for a 96-cell, all-to-all connected network (Table I). In Section VI, we shall see that connectivity density is a deciding factor for the performance of the target application.

B. *InfOli Use Cases*

For our analysis, we use three use cases, which are representative of the memory and computational requirements in typical InfOli workloads. All of the use cases are realistic instances of the InfOli application and have neuroscientific merit.

The biology of each neuron is characterized by the internal conductances of the ion channels modeled in each compartment. In all use cases, the user can set each neuron ion channel conductance separately with every experiment and for each cell, giving the greatest possible control over the biological behavior of the simulated network. Additionally, the application allows for the connectivity of the InfOli network to be programmable by the user before the simulation is deployed. The network connectivity is defined by an $N \times N$ *connectivity matrix* (where N is the network size) of FP values signifying the weight of each connection. The weight value is used in the GJ computations to calculate the connection impact on the neuron. A weight of 0.0 denotes the absence of the corresponding GJ connection. The three use cases are focused around the biological complexity of the GJs:

1) **InfOli with Realistic Gap Junctions (RGJ)** – InfOli cells modeled with (biophysically) realistic GJ interconnectivity. The highest amount of detail is included in the GJ modeling.
2) **InfOli with Simplified Gap Junctions (SGJ)** – InfOli cells modeled with GJs replaced by simplified , passive connections. This constitutes a simpler implementation in comparison to the previous use case.

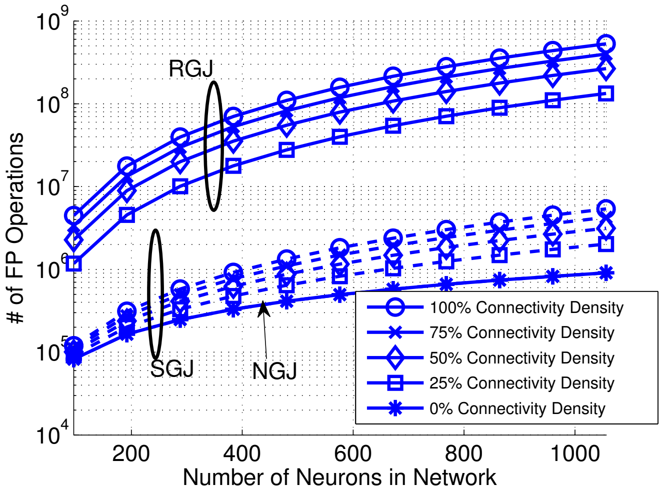


Fig. 2: Floating-point operations needed per simulation step of the InfOli model for each use case and for different connectivity densities.

3) **InfOli with No Gap Junctions (NGJ)** – InfOli cells modeled without accounting for GJs and without any interconnectivity implementations. This is the simplest use case, whereby the neurons are modeled as separate computational islands.

In Figure 2, we present the amount of FP operations, based on the manual profiling of a simple, sequential C implementation of the InfOli application. Numbers appear for each of the three above use cases for different network sizes. It should be noted that the lowest connectivity density of the RGJ and SGJ cases corresponds to the NGJ case, since no neuron connections are implemented whatsoever. As a result, it is expected for the respective number of FP operations to coincide between the RGJ and SGJ cases in Figure 2.

1) InfOli with Realistic Gap Junctions (RGJ): This use case represents a fully featured version of the InfOli application. The complex Gap Junction dominates the computation in this use case. GJs here are implemented as a very specific representation of the biological nucleus (Listing 1). In Figure 2 the computations increase quadratically, as we simulate more neurons and increase their connectivity density.

2) InfOli with Simplified Gap Junctions (SGJ): Often, experiments explore more rudimentary connections between HH-modeled neurons. Such level of detail as in the RGJ case is an overkill while inducing a significant amount of computational cost. Thus, we assume a use case of the InfOli application that simplifies the connection between neurons to a simple input accumulator – more typical for this kind of experiments. The accumulation is parameterized using the weight that is assigned to each connection between two neurons. This use case has significantly lower processing requirements. Even though increasing the network size leads to similar results (compared to the RGJ case), the actual FP operations are reduced by about two orders of magnitude compared to the previous use case (Figure 2).

TABLE II: Specifications of Evaluation Platforms

Spec	Maxeler Vectis	Xeon Phi
Host system	i7-2600@2.8GHz with 16GB of RAM	Xeon E5-2697v2@ 2.7GHz with 64GB of RAM
On-Board DRAM	24 GB	8GB
RAM bandwidth	38.4 GB/s	320 GB/s
Memory streams/channels	15	16
On-chip memory	6.5 MB (FPGA BRAMs)	30 MB (L2 cache)
Number of chip cores	Not Applicable	61
Chip frequency	depends on the implemented design	1.053 GHz
Instructions set	fully configurable	64 bit
Power consumption	140 W	225 W

Listing 1: Example of RGJ implementation in C.

```
    for (i=0; i<InfOli_N_INPUT; i++) {
        V = prevVdend − neighVdend[i];
        f_new = V * exp(−1 * V * V/100);
        F_acc =+ f_new;
        V_acc =+ V;
    }
    Ic = CONDUCTANCE * (0.8*F_acc + 0.2*V_acc);
    return Ic ;
```

3) InfOli with No Gap Junctions (NGJ): This use case represents the minimally featured version of the InfOli application in which neurons run as independent computational islands. No connectivity is being modeled and the application exhibits a high degree of data-level parallelism. In cases where the simulated brain time is very short and the connectivity not dense, it might be more efficient to implement the connectivity outside the acceleration platform, thus, this use case is also relevant for real life experiments. The processing requirements scale almost linearly to the network size and compared to the other use cases fewer computations are needed as shown in Figure 2. Here, the only communication overhead is due to input/output traces transfer.

C. Quantifying Neuron Interconnectivity

While the InfOli application can implement any neuron interconnectivity (through a simple connectivity matrix), for the purposes of this work we feed the InfOli application with connectivity maps that were created *a priori* in order to reflect a certain connectivity density. Thereby, inter-neuron communication (and its associated computing complexity and memory overheads) can be manipulated for profiling purposes. To fully enable this configuration, we have implemented a *connectivity generator* which prepares the input of the InfOli application. The network to be simulated is, in the general case, expressed as a three-dimensional mesh of neurons. Each neuron is assigned a set of three integers representing the neuron's normalized Cartesian coordinates. These sets can be used to calculate the distance between each neuron pair. The distance between neurons adjacent to each other is considered as the unit of distance measurement. Based on the distance of each neuron pair, the probability of them forming a synaptic connection can be computed, according to a pre-defined distribution. Figure 3 illustrates an example of the probability of a connection being formed (P_c) for a variety of distances between neurons (r) and a range of exponential distributions (differentiated by their average value μ). For each evaluated

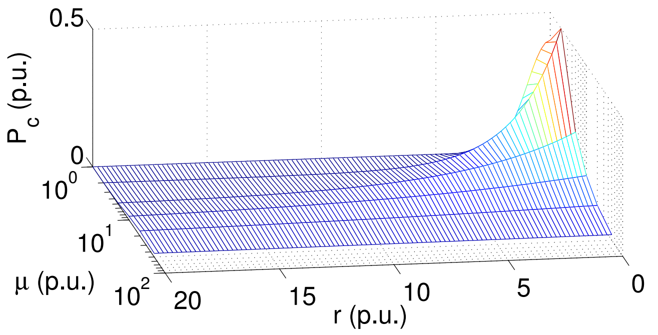


Fig. 3: Example of connection probability, using the exponential distribution with mean value μ and the distance between neurons (r). Both quantities are measured per unit (p.u.).

pair of neurons in the network, P_c is calculated and a random number x is produced between 0 and 1. Iff $x < P_c$, then we assume that a connection will be implemented for these two neurons. As a result, by tuning distribution parameters (μ in the example of Figure 3), a different intensity in the formation of neuron connections is achieved, and consequently different connectivity densities.

V. Target Platforms

All three InfOli use cases have been evaluated on two platforms. More precisely a hardware-based implementation was developed to be ported to a Maxeler *Vectis* Data-Flow Engine (DFE) board as well as a software-based implementation for the Intel Xeon Phi 5110P system [14] (part of the Blue Wonder iDataPlex cluster hosted at STFC, UK [25]). The specifications' overview for both evaluation platforms is presented in Table II.

The Vectis DFE is a Maxeler HPC node based on reconfigurable hardware. Its tool flow is designed and optimized to accommodate the acceleration of dataflow applications; that is, applications with the bulk of their implementation using purely raw computations with the absence (partially or totally) of branching execution or feedback paths. The Maxeler tools can exploit the nature of dataflow applications to implement very fine-grain pipelined designs, maximizing processing throughput and overall performance. Maxeler DFEs provide several GBs of DRAM with on-board high bandwidth connections to the reconfigurable chip. Thus, they are an excellent fit for many scientific workloads, such as ours, and in general for workloads that require to process large amount of data. What makes Maxeler DFEs stand out from the rest of the

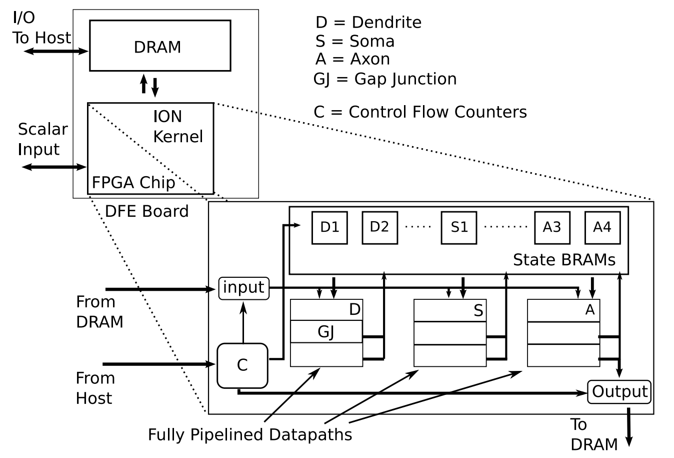


Fig. 4: Vectis architecture of the InfOli kernel. The GJ is not present in the NGJ use case.

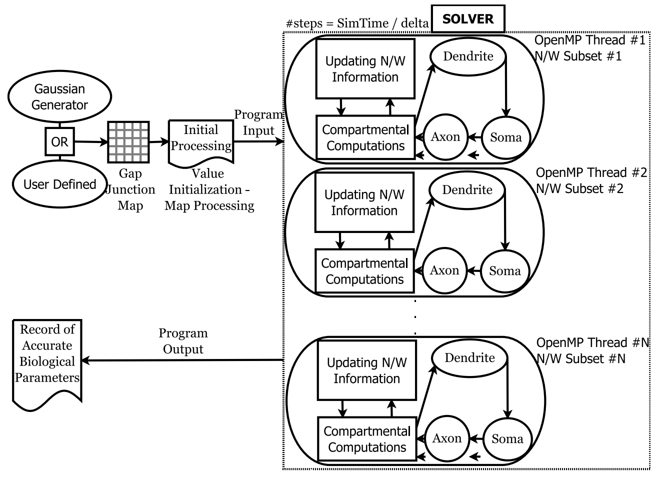


Fig. 5: Xeon-Phi architecture of the InfOli kernel.

FPGA-based solutions is the excellent high-level programming language employed for kernel coding (Java with Maxeler-related extensions) and the ability to form scaled up, multi-DFE platforms in a seamless (i.e. user-transparent) manner [26]. Consequently, Maxeler has covered significant ground in bridging part of the programmability gap between reconfigurable hardware and general purpose processing nodes, such as the Xeon Phi. The DFE board used in our experiments is a 3rd-generation Vectis-DFE board, that includes a Xilinx Virtex-6 FPGA chip.

The Xeon Phi is a Many Integrated Core (MIC) architecture co-processor, which features 61 cores, each capable of supporting up to 4 instruction streams. The current generation of Phi cards, named Knights Corner, use an Intel Xeon host processor which can offload work to the Phi, much like a GPU, using well-known programming models such as OpenMP and OpenCL. However, in contrast to GPU mentality, the Phi can also be thought as a stand-alone processor in that it has its own Operating System. This allows for an application to run natively on the platform, which is what our InfOli implementation opts for.

A. *InfOli on Maxeler Vectis*

The DFE implementation of the InfOli application depicted in Figure 4 is based on the design presented in [11]. It incorporates 3 internal pipelines, one for each part of the neuron (Dendrite, Soma, Axon), each performing the respective computations. The state parameters for every neuron are stored in separate BRAM blocks (one per neuron) for faster and higher-bandwidth access. The input stream to the DFE kernel is written in the on-board DRAM and represents the evoked (external) inputs used in the dendritic computations comprising the network input. The initial state and neighboring (gap-junction) influence are streamed in the DFE from the on-board memory only once (initialization data) at the beginning of the simulation. During execution, the kernel output is streamed out to the on-board memory while also updated on the (on-chip) BRAM blocks of the DFE. The connectivity matrix weights are also sent to the DFE at every simulation step, for the use cases that include programmable connectivity. Using a memory-based connectivity matrix allows us to avoid the time-consuming process of resynthezing a new DFE for updating the connectivity density of an experiment.

The program flow is tracked using hardware counters which monitor the neurons executed, the number of simulation steps completed, as well as the GJ loop iterations (where applicable). The data flows through the DFE pipelines at each kernel execution tick, consuming an input set and producing the output and a new neuron state. The DFE execution naturally pipelines the processing of different neurons within a simulation step. Simulation steps are not independent from each other and thus are not parallelizable. That is because every neuron requires the previous state of all other neurons to compute its GJ (in the RGJ or the SGJ case) before a new step begins. As a consequence, the DFE pipeline is flushed before a new simulation step. This dependency is removed when 0% connectivity density is imposed on a simulation. In experiments with random connectivity, a constant number of ticks is spent by the DFE for computing each GJ connection whether it actually exists or not. This is because the kernel is designed with a fixed pipeline for computing GJ so as to be reusable for experiments with different connectivity parameters. In doing so, re-synthesizing the DFE kernel is avoided saving setup time for each experiment. Conversely, this DFE implementation cannot get a performance benefit from lower connectivity densities. However, using the same DFE for experiments of different connectivities offers predictable, guaranteed performance.

The resource utilization of the FPGA device of the DFE is reported in Table III. A single computation kernel (DFE) fits on the FPGA device for each use case. SGJ has reduced computations allowing a higher degree of unrolling the GJ computation loop yielding added performance benefits.

B. *InfOli on Xeon Phi*

The InfOli application on the Intel Xeon Phi co-processor, depicted in Figure 5, uses a shared-memory programming model by utilizing the OpenMP library. The model first accepts a user-defined map detailing connections and size of the desired neuron network to be simulated. This map is pro-

TABLE III: Logic utilization for the Virtex 6 FPGA chip on the Vectis.

RGJ	**SGJ**	**NGJ**
257409 / 297600 (86.49%)	268458 / 297600 (90.20%)	167147 / 297600 (56.16%)

cessed once and a network is dynamically generated, forming connections between the dendrites (Gap Junctions) as dictated by the map. Each neuron's GJ allocates arrays of double-precision floats. These arrays need to always store up-to-date information for a subset of the entire network, specifically the most recent dendritic membrane voltage potentials of the neurons on the other side of the GJ connections. Thus, for each simulation step, the algorithm of the model is summarized as refreshing the information of these arrays and, then, performing the necessary computations for each of the three neuron compartments (dendrite, soma and axon). In each algorithmic step, the neurons are divided into subsets as evenly as possible. Each subset is then handled by an OpenMP thread. Since the Xeon Phi 5110P can support up to $61 \times 4 = 244$ instruction streams, the application can use up to 244 OpenMP threads, splitting the network in just as many subsets. Therefore,that for network sizes below 244 neurons, the maximum number of threads used (and thus the degree of parallelism) is limited by the number of simulated neurons. This is because a single neuron cannot be split into multiple threads. It has also been observed that dividing the network into very small subsets does not yield better performance, since each thread ends up having low workload, disproportional to its overheads. Therefore, it is not always efficient to maximize the number of OpenMP threads, particularly for small networks.

VI. Performance Evaluation

In this section, we evaluate the performance of the use cases on the Xeon-Phi and the Vectis-DFE nodes. All use cases were executed using a simple experiment, borrowed from corresponding biological experiments, designed to produce a typical response from the InfOli model: a complex spike at the neuron output stage (axon). The experiments simulate 6 seconds of brain time. The complex spike is produced by evoking a small 6.0 mA pulse as input to all InfOli cells at the same point after program onset for about 500 simulation steps (or 25 ms, in brain time).

A standard procedure for experimenting with SNN models typically begins with an extensive, initial parameter-space exploration using small- or medium-sized networks. Having fine-tuned all model parameters, real experiments can then commence by simulating either small- to medium-sized networks (10s to 100s of cells) for exploring *real-time, closed-loop control* such as Brain-Computer Interfaces [10] (TYPE 1), or large-scale networks (>1000s of cells) for mounting *behavioral experiments* [27] (TYPE 2). To tackle both experiment types, in this section we evaluate a range of 96- to 1,056-cell networks as well as a range of 960- to 7,680-cell networks.

A. Measurement Methodology

Timing measurements on the Vectis DFE were taken measuring the kernel time within the host code using timestamps before and after the kernel call. The CPU host code is blocking, thus, only the DFE kernel is active during the measurement. The time includes the kernel execution (processing and DRAM data-exchange delay) and the activation delay of the FPGA device. This activation takes about 1 ms, which is negligible compared to the overall execution time that takes several seconds to several minutes in our experiments. The execution time of a single-simulation-step is derived from the total execution time divided by the number of simulation steps. The DRAM communication delay can be estimated by the amount of data exchanged between the FPGA device and the on-board DRAM per simulation step, considering the DFE DRAM bandwidth.

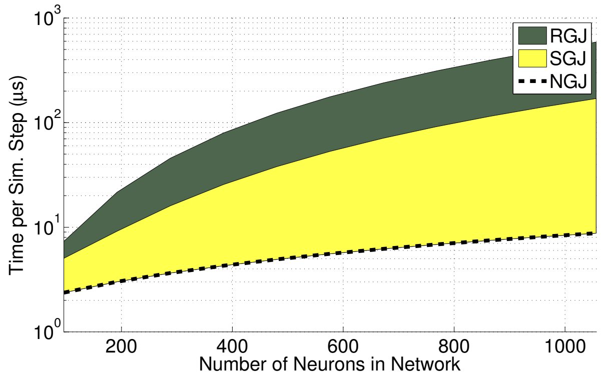
All measurements concerning the Xeon Phi have been carried out with Intel's profiling and analysis tool Vtune Amplifier XE 2015. Information for the DRAM accesses and average bandwidth used by the application, was obtained by performing bandwidth-mode analysis. This analysis further provides insights related to the program execution time and CPU utilization. The profiler was installed on the Intel Xeon host (since the Phi card has a minimal OS) and launched with directives for collecting information from the accelerator platform (*-target-system=mic-native* flag).

B. Execution Time vs. Network Size vs Connectivity Density

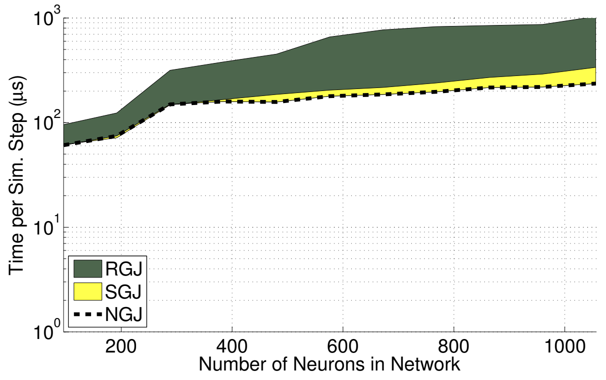
Next, we evaluate the performance of the two platforms for different network sizes and connectivity densities. All execution results are presented per simulation time step.

The performance results of the three use cases on the Maxeler Vectis-DFE platform are depicted in Figures 6a and 6c, for small-to-medium and for large network sizes, respectively. Connectivity density does not affect the execution time of the DFE implementations. That is because the design statically supports all-to-all connections (100% connectivity density) in a fixed dataflow pipeline. As discussed earlier, this avoids re-synthesis of the design whenever connectivity parameters change, reducing the time needed to setup an experiment.

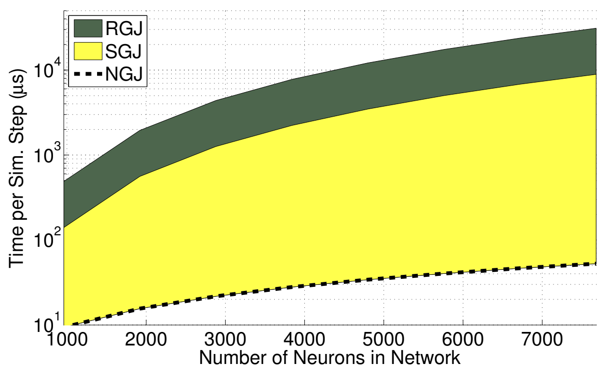
As illustrated in Figure 6a, for small/medium network sizes, the DFE's fine-grain parallelism yields good performance scaling to the network size, even for the demanding case of InfOli neurons with realistic GJs (RGJ). Execution time ranges from 6 us to a little over 500 us for 1,056 neurons. Figure 6c shows the DFE's performance for larger network sizes (>1000). The use of the BRAM blocks on the DFE FPGA chip, where the most frequently accessed data (the neuron states) are stored, improves input/output bandwidth and overall performance but limits available chip area and, thus, maximum neuron capacity which is 7,680 neurons. Now, execution times, for the RGJ case, increase more rapidly with increasing problem sizes. The reason for this degradation is as follows: In order to parallelize the GJ computations efficiently, the DFE uses loop unrolling on the main computation loop of the GJs. The resources of the FPGA chip allow for a maximum unroll factor of 16. Although this is sufficient for exhibiting good performance scalability for networks of up to



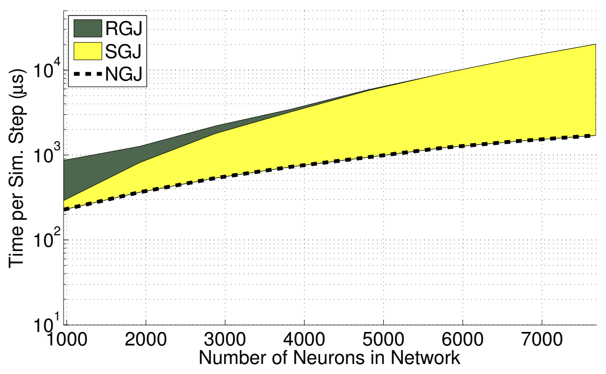
(a) Maxeler Vectis



(b) Xeon Phi



(c) Maxeler Vectis



(d) Xeon Phi

Fig. 6: The Vectis DFE has better performance at reduced neuron network sizes, however Xeon Phi is the better option for larger problem sizes. Besides, RGJs come at an extra performance cost for all considered cases. The NGJ case constitutes, by definition the lower bound of InfOli performance. Colored areas correspond to the range of possible execution-time values due to different connectivity densities (0%-100%). (Note: Areas are not stacked. RGJ performance is surpassing the SGJ case by the highlighted amount.)

1,000 neurons, for larger problem sizes speedup is limited. SGJ is an exception, as the simpler GJ connections allow for a higher unroll factor (96) and provide good scalability for both the small-to-medium and large-scale networks.

Although the maximum size of a simulated network is important for TYPE-2 experiments, achieving (brain) real-time speed is critical for TYPE-1 experiments. Table IV presents the *real-time* capabilities of each use case. For the RGJ use case, the Vectis DFE can simulate 300 neurons at real-time speed, while for the SGJ case the real-time network is 550 neurons. As expected, the NGJ case exhibits a linear increase in execution time; this results in a capacity to execute 7,200 neurons in real-time on the DFE.

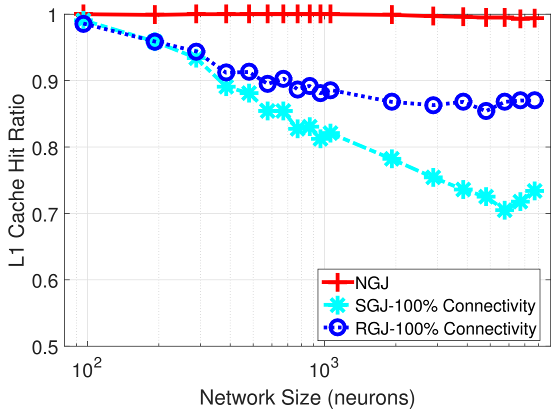
Application behavior is significantly different on the Xeon-Phi implementation. Here, the design can actually benefit from lower connectivity densities but, at the same time, cannot provide a performance guarantee for every problem size in each use case. Different problem sizes and connectivity distributions can produce quite varied connectivity densities that affect performance.

In general, the Xeon-Phi performs well in simulating small-to-medium scale networks. Still, it is about 50% (medium-scale networks) to almost an order of magnitude (in small-scale networks) worse than the DFE implementation in the RGJ use case. The gap is even greater for SGJ and NGJ as illustrated in Figure 6b.

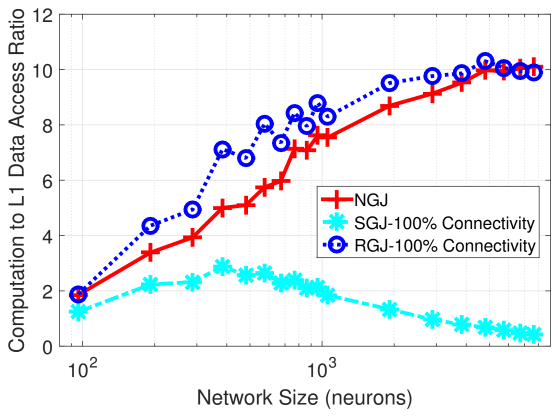
TABLE IV: Real-time achievable network for each use case on each platform

Vectis-DFE Implementation	**Real-time Network Size**
RGJ	300
SGJ	550
NGJ	7,200
Xeon-Phi Implementation	**Real-time Network Size**
RGJ (100% connectivity)	24
SGJ (100% connectivity)	54
NGJ	54

The above trends change notably when simulating larger-scale networks, mainly for the RGJ case as shown in Figure 6d. On one hand, the FPGA resources available in the Maxeler Vectis DFE limit the degree of unrolling in the GJ computations. On the other hand, the Xeon Phi is able to more efficiently handle the increasing GJ computations through a large number (244) of parallel threads, which are finally utilized efficiently. Thereby, the Xeon Phi can scale RGJ performance better for large network sizes, achieving up to 30-35% lower execution time than the Vectis DFE. That is



(a) L1 Hit Rates



(b) Compute-to-Data Ratio Metric

Fig. 7: L1 cache hit rates and compute-to-data ratio for each use case for 100% connectivity networks of the Xeon Phi implementation.

not, however, the case for the other two use cases. When no connectivity is present or simpler connections are used, the DFE manages to keep up with the increasing computations and keeps performing better than the Xeon Phi even for large networks.

To give a better insight on Xeon Phi's performance we discuss in more detail some of its performance metrics. Figure 7a illustrates the hit ratio of the Xeon Phi's L1 caches when simulating networks of various sizes and connectivity patterns. Large networks without GJs require small amounts of data and exhibit good data locality, leading to high hit ratios. On the other hand, simple and regular GJs have large memory footprints, leading to lower L1 hit rates. SGJ has a lower L1 hit ratio than RGJ as its GJs are less complex and exhibit less data reuse during processing, compared to the realistic GJs. Once the core fetches the GJ's data to its L1 cache, processing an a realistic GJ will yield more L1 hits than in the simpler GJ. Moreover, Figure 7b presents the ratio of CPU cycles spent in computations per CPU cycles spent in accessing L1 cache and leads to similar observations. NGJ and RGJ achieve more computations per L1 access compared to SGJ which appears to wait overall longer for the memory.

Aiming for real-time performance, Xeon Phi can simulate 24 neurons in RGJ experiments. The real-time achievable network for the SGJ and NGJ is 54 neurons. All cases are substantially lower than Maxeler DFE as shown in Table IV. This is explained by the fact that the Xeon Phi resources cannot be used efficiently for such small networks sizes.

C. DRAM Data-Transfer Overhead

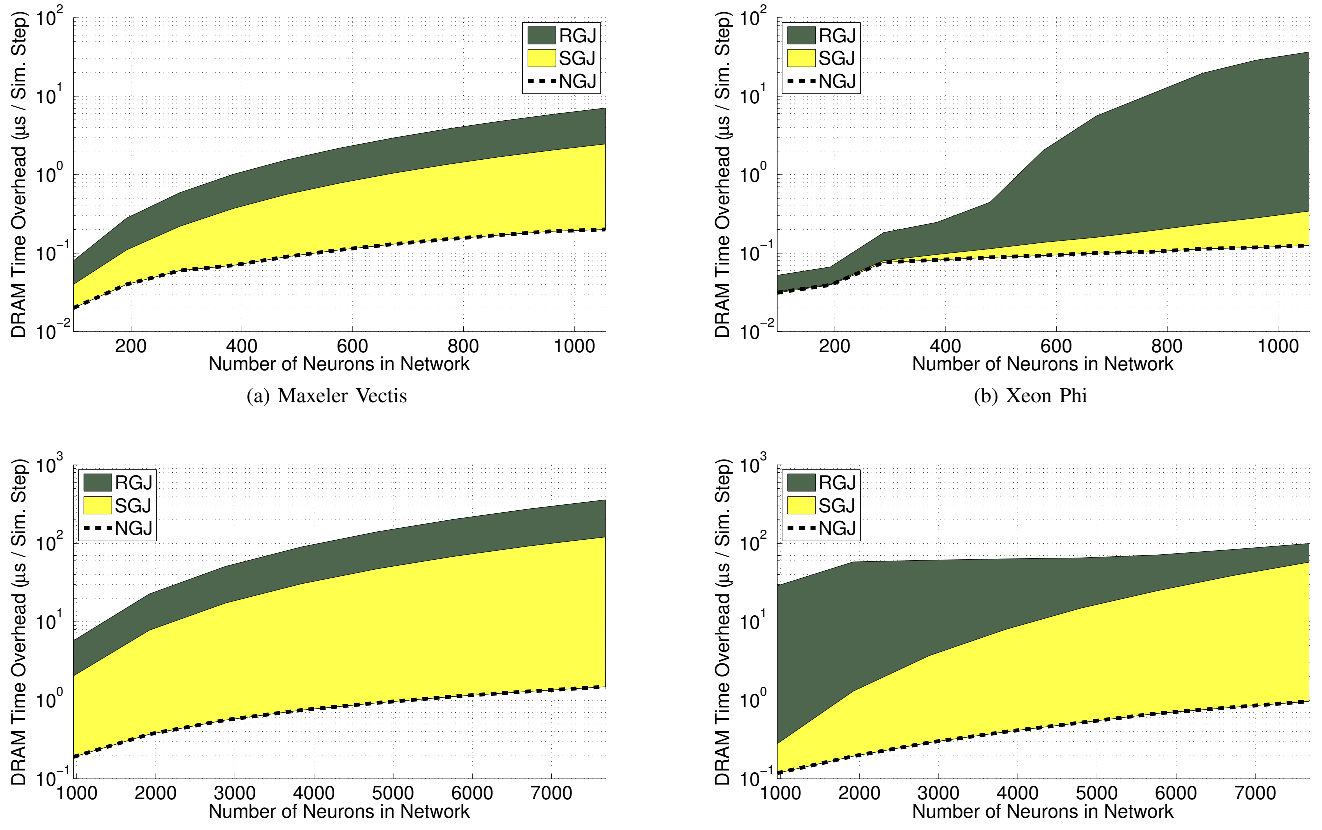
Besides analyzing the overall execution time, it is interesting to also measure the fraction of overall time spent in processing versus the time spent in waiting for the DRAM data to arrive. Typically, in this class of applications, where complex biophysically accurate models are simulated, processing dominates the overall execution time. So, the data-transfer timing overheads are expected to be small compared to the computation times.

In the Vectis-DFE implementations, fast on-FPGA-chip BRAM blocks are used to store the neuron states. BRAMs increase the overall input/output bandwidth of the DFE and require in practice a single cycle to access. Neuron states are the most frequently used data both for the neuron-compartment processing as well as for the GJs processing. As a result, the Vectis-DFE implementation has negligible data-transfer overheads. The fraction of time spent waiting for off-chip data in small-to-medium networks, depicted in Figure 8a, is consistently under 0.02% in the most memory-demanding RGJ case. Even for NGJ, where computations scale linearly to problem size, the time spent waiting for DRAM access is slightly over 0.02%, showing that the provided DRAM bandwidth is sufficient for feeding the DFE engine. For larger networks – even in the worst case – the time spent waiting for data transfers is still very low as shown in Figure 8c.

The Xeon-Phi offers an order of magnitude higher memory bandwidth than the DFE, achieving 320 GB/s versus 38.4 GB/s. As can be observed in Figure 8b, for the most data-intensive RGJ, the DRAM overhead in execution time does not scale well above 500 neurons. Still, the overall performance penalty is low due to the high memory bandwidth of the Xeon Phi. When simpler or no GJs are employed (SGJ, NGJ), the DRAM data-transfer overhead is negligible and remains so for all tested problem sizes. In larger-scale networks, the bandwidth is utilized even more efficiently as shown in Figure 8d, keeping the performance impact of DRAM data-transfer low. It is noteworthy that, in the RGJ experiments for more than 2,000 neurons the DRAM overhead in execution time reaches a plateau indicating that Xeon Phi gets to a steady state where DRAM feeds the cores with new data at the same rate that they are processed.

VII. Discussion

Even though HH models comprise challenging systems of differential equations due to their high modeling accuracy, such models form essentially *embarrassingly parallel* computational problems that can be solved with a typical divide-and-conquer strategy. In such cases, each neuron model effectively becomes a free-running oscillator, the execution of which can be parallelized independently of its neighbours. The



(a) Maxeler Vectis (b) Xeon Phi

(c) Maxeler Vectis (d) Xeon Phi

Fig. 8: DRAM data-transfer overheads for the considered workloads. It is clear that both platforms have enough provisions and are not DRAM-bound for any of the explored use cases. Colored areas correspond to the range of possible timing-overhead values due to different connectivity densities (0%-100%). (Note: Areas are not stacked. RGJ overhead is surpassing the SGJ case by the highlighted amount.)

more powerful the processing nodes employed, the higher the speedups achieved.

However, complementing cell models to include Gap-Junction (or any type of complex-connection) modeling leads to extended-HH models which not only feature increased computational complexity (due to the GJ calculations) but, additionally, cease to exhibit an embarrassingly parallel nature. The reason, of course, is that – with a rising connectivity density among neurons in a network – dependencies among differential equations also rise, leading to computational problems that are increasingly difficult to parallelize across simulation time steps. In effect, in such cases *coupled oscillators* are formed that need to be co-simulated in strict lockstep among them. This requirement, in turn, enforces the use of cycle-accurate, transient simulators where simulation steps are hardly compressible and all neuron states need to be completely updated at each simulation step.

From the above discussion it is obvious why the majority of the computational-neuroscience community has so far avoided employing HH models and multi-compartmental models with complex connections on large problem sizes using conventional computing machines. Yet, the eventual use of biophysically plausible neurons and connections on a larger scale is required, especially for the exploration of systems explaining biological behavior. The emergence of new mainstream HPC platforms and processing nodes, such as the ones discussed in this paper, has the ability to fill the technological gap required for such neuroscientific experiments.

The Maxeler Vectis-DFE implementation shows impressive performance for small-to-medium scale networks and also achieves high real-time networks, as compared to the current state of the art. This makes the DFE a suitable platform for speeding up experimentation on small-to-medium size neuron networks, that are often used for parameter-space exploration of neuron models such as the InfOli. The DFE is also suitable for experimentation with real-time setups (TYPE 1) as it can achieve networks of meaningful sizes at real-time speeds. Finally, it can also provide predictable performance for any kind of network size or input and connectivity, a crucial factor when planning lengthly experiments and a feature that software-based solutions cannot easily provide. The Vectis DFE does have limitations, though, in performance when the computational demands increase above the parallelization capabilities that the DFE can provide, as it can be seen for the RGJ use case.

The Xeon-Phi platform, on the other hand, is suitable for larger-scale experiments (TYPE 2) of extreme computational demand, because of the high memory bandwidth and amount of parallel threads provided. Besides, despite the concessions Maxeler makes through use of its Maxeler-Java programming language, the Xeon Phi remains a much more straightforward platform to program on, as it is a software-based solution. Furthermore, it can exploit different connectivity densities in terms of performance efficiency, but this aspect also results in its inability to provide predictable performance. Additionally, the resources on the Xeon Phi can support a larger maximum network population that the Vectis DFE. However, when it comes to real time model execution, the Xeon Phi falls short of the $50 - \mu sec$ timing constraint for achieving real-time simulations, regardless of the network size used.

VIII. Conclusions

In this paper, we present a thorough performance analysis for a state-of-the-art neuroscientific application, targeting the Inferior Olive (InfOli) neuron cells. The societal and scientific importance of this class of workloads is directly tied to the insight they provide into obscure brain functionality. As such, they are typically ported on accelerator platforms, so that their computational complexity can be tackled. We target two accelerators that have demonstrated great potential, especially given their integration and clustering capabilities: the Maxeler Vectis Data-Flow Engine (DFE) and the Intel Xeon Phi. The quality and intensity of inter-neuron connections has been used to isolate representative use cases of the InfOli application for the two target platforms. We substantiate, in all cases, that the target neuron simulator scales gracefully in terms of DRAM utilization, with both platforms exhibiting enough slack in DRAM timing overhead. Regarding overall performance, the Maxeler Vectis Data Flow Engine (DFE) is clearly optimal for small- and medium-scale, real time simulations. Executing a fixed synthesized implementation, the performance of the DFE is not affected by changes in neuron connectivity density. The Xeon Phi, on the other hand, appears more suitable for large-scale simulations, with many neurons and dense interconnectivity between them.

IX. Acknowledgements

This work is partially supported by the European-Commission Horizon2020 ERC-PoC-2014 project BrainFrame (SEP-210151915) and Framework Programme Project Vineyard (Grant Agreement n 687628). We also like to thank the STFC Hartree Centre (UK) for providing the Maxeler and Xeon-Phi computational resources used in our experiments.

References

[1] National Academy of Engineering (nae.edu), "Grand Challenges for Engineering," 2010.

[2] W. McColloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[3] G. Wulfram and W. Werner, *Spiking Neuron Models*. Cambridge University Press, 2002.

[4] W. Maass, "Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons," in *Neural Information Processing Systems*, pp. 211–217, 1996.

[5] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, pp. 1659–1671, 1997.

[6] E. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?," *IEEE Trans on Neural Net.*, vol. 15, no. 5, 2004.

[7] G. Smith, C. Cox, S. Sherman, and J. Rinzel, "Fourier Analysis of Sinusoidally Driven Thalamocortical Relay Neurons and a Minimal Integrate-and-Fire-or-Burst Model," *Neurophysiology*, vol. 83, pp. 588–610, 2000.

[8] G. B. Ermentrout, "Type I membranes, phase resetting curves, and synchrony," *Neural Computation*, vol. 83, pp. 979–1001, 1996.

[9] E. Izhikevich, "Simple Model of Spiking Neurons," *IEEE Trans. on Neural Networks*, vol. 14, no. 6, 2003.

[10] T. Yamazaki and J. Igarashi, "Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit," *Neural Networks*, vol. 47, pp. 103–111, 2013. Computation in the Cerebellum.

[11] G. Smaragdos, C. Davies, C. Strydis, I. Sourdis, C. Ciobanu, O. Mencer, and C. De Zeeuw, "Real-Time Olivary Neuron Simulations on Dataflow Computing Machines," in *Supercomputing* (J. Kunkel, T. Ludwig, and H. Meuer, eds.), vol. 8488 of *Lecture Notes in Computer Science*, pp. 487–497, Springer International Publishing.

[12] G. Smaragdos, S. Isaza, M. V. Eijk, I. Sourdis, and C. Strydis, "FPGA-based Biophysically-Meaningful Modeling of Olivocerebellar Neurons," in *22nd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, Feb. 2014.

[13] H. D. Nguyen, Z. Al-Ars, G. Smaragdos, and C. Strydis, "Accelerating complex brain-model simulations on GPU platforms ," in *Design, Automation, and Test in Europe, DATE 2015*, Mar. 2015.

[14] J. James and J. Reinders, *Intel Xeon Phi coprocessor high-performance programming.* 2013.

[15] J. Liu, H. Wang, D. Wang, Y. Gao, and Z. Li, "Parallelizing Convolutional Neural Networks on Intel Many Integrated Core Architecture," in *Architecture of Computing Systems – ARCS 2015* (L. M. P. Pinho, W. Karl, A. Cohen, and U. Brinkschulte, eds.), vol. 9017 of *Lecture Notes in Computer Science*, pp. 71–82, Springer International Publishing.

[16] A. Viebke and S. Pllana, "The Potential of the Intel R Xeon PhiTM for Supervised Deep Learning ," in *17th IEEE International Conference on High Performance Computing and Communications (HPCC 2015)*, June 2015.

[17] D. Rodopoulos, G. Chatzikonstantis, A. Pantelopoulos, D. Soudris, C. De Zeeuw, and C. Strydis, "Optimal mapping of inferior olive neuron simulations on the Single-Chip Cloud Computer," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pp. 367–374, July 2014.

[18] S. W. Moore, P. J. Fox, S. J. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive — A Field-Programable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation," in *IEEE Int. Symp. on FCCM*, pp. 133–140, 2012.

[19] K. Cheung, S. R. Schultz, and W. Luk, "A large-scale spiking neural network accelerator for FPGA systems," in *Int. conf. on Artificial Neural Networks and Machine Learning*, ICANN'12, pp. 113–120, 2012.

[20] Y. Zhang, J. Nuñez-Yañez, J. McGeehan, E. Regan, and S. Kelly in *Proc. Int'l Conf. Field Programmable Logic and Applications*, 2009.

[21] M. Beuler, A. Tchaptchet, W. Bonath, S. Postnova, and H. A. Braun, "Real-Time Simulations of Synchronization in a Conductance-Based Neuronal Network with a Digital FPGA Hardware-Core," in *Artificial Neural Networks and Machine Learning – ICANN 2012*, September 2012.

[22] M. Bhuiyan, A. Nallamuthu, M. Smith, and V. Pallipuram, "Optimization and performance study of large-scale biological networks for reconfigurable computing," in *Fourth International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, pp. 1–9, nov. 2010.

[23] C.I. De Zeeuw, F.E. Hoebeek , L.W.J. Bosman, M. Schonewille, L. Witter, and S.K. Koekkoek, "Spatiotemporal firing patterns in the cerebellum," *Nat Rev Neurosci*, vol. 12, pp. 327–344, jun 2011.

[24] J. R. De Gruijl, B. Paolo, G. de Jeu Marcel T., and D. Z. C. I., "Climbing Fiber Burst Size and Olivary Sub-threshold Oscillations in a Network Setting," *PLoS Comput Biol*, vol. 8, 12 2012.

[25] "Science&Technology Facilities Council, The Hartree Centre."

[26] Maxeler Technologies, "https://www.maxeler.com/products/."

[27] B. Glackin, J. A. Wall, T. M. McGinnity, L. P. Maguire, and L. J. McDaid, "A spiking neural network model of the medial superior olive using spike timing dependent plasticity for sound localization," *Frontiers in Computational Neuroscience*, vol. 4, no. 18, 2010.