Journal of Neural Engineering

PAPER • OPEN ACCESS

BrainFrame: a node-level heterogeneous accelerator platform for neuron simulations

To cite this article: Georgios Smaragdos *et al* 2017 *J. Neural Eng.* **14** 066008

View the article online for updates and enhancements.

BrainFrame: a node-level heterogeneous accelerator platform for neuron simulations

Georgios Smaragdos[1] , **Georgios Chatzikonstantis**[3]**, Rahul Kukreja**[4]**,**
Harry Sidiropoulos[3]**, Dimitrios Rodopoulos**[5]**, Ioannis Sourdis**[2]**,**
Zaid Al-Ars[4]**, Christoforos Kachris**[3]**, Dimitrios Soudris**[3]**, Chris I De Zeeuw**[1]
and Christos Strydis[1]

[1] Neuroscience department, Erasmus MC, Wytemaweg 80, 3015GE, Rotterdam, Netherlands
[2] Computer Science and Eng. department, Chalmers University of Technology, SE-412 96, Gothenburg, Sweden
[3] MicroLab, National Technical University of Athens (NTUA), 9 Heroon Polytechneiou, 15780, Athens, Greece
[4] Computer Eng. Lab, Delft University of Technology, Mekelweg 4, 2628CD, Delft, Netherlands
[5] imec, Kapeldreef 75, Leuven 3000, Belgium

E-mail: g.smaragdos@erasmusmc.nl and c.strydis@erasmusmc.nl

CrossMark

Abstract

Objective: The advent of high-performance computing (HPC) in recent years has led to its increasing use in brain studies through computational models. The scale and complexity of such models are constantly increasing, leading to challenging computational requirements. Even though modern HPC platforms can often deal with such challenges, the vast diversity of the modeling field does not permit for a homogeneous acceleration platform to effectively address the complete array of modeling requirements. *Approach*: In this paper we propose and build BrainFrame, a heterogeneous acceleration platform that incorporates three distinct acceleration technologies, an Intel Xeon-Phi CPU, a NVidia GP-GPU and a Maxeler Dataflow Engine. The PyNN software framework is also integrated into the platform. As a challenging proof of concept, we analyze the performance of BrainFrame on different experiment instances of a state-of-the-art neuron model, representing the inferior-olivary nucleus using a biophysically-meaningful, extended Hodgkin–Huxley representation. The model instances take into account not only the neuronal-network dimensions but also different network-connectivity densities, which can drastically affect the workload's performance characteristics. *Main results*: The combined use of different HPC technologies demonstrates that BrainFrame is better able to cope with the modeling diversity encountered in realistic experiments while at the same time running on significantly lower energy budgets. Our performance analysis clearly shows that the model directly affects performance and all three technologies are required to cope with all the model use cases. *Significance*: The BrainFrame framework is designed to transparently configure and select the appropriate back-end accelerator technology for use per simulation run. The PyNN integration provides a familiar bridge to the vast number of models already available. Additionally, it gives a clear roadmap for extending the platform support beyond the proof of concept, with improved usability and directly useful features to the computational-neuroscience community, paving the way for wider adoption.

1. Introduction

In vivo and *in vitro* experiments are a traditional tool of neuroscientific research. They are powerful experimentation methods, but are also time-consuming and not always reliable. A number of factors can contaminate results like, for example, the influence of anesthesia in *in vivo* experiments. What is more, most systemic neuroscientific phenomena require the monitoring of biological systems of very large scale and many such techniques do not allow for this kind of study. Computational neuroscientists use spiking neural networks (SNNs) to cope with such issues. By incorporating SNN models of varied complexity (which themselves are derived by biological experiments) they create predictive simulators that can test their scientific hypotheses and drive more targeted, thus more reliable and refined, biological experimentation [1].

A major challenge in executing such simulations is the sheer computational complexity that many SNN models entail, compared to simpler modeling classes. Traditional methods of computing, in which the common simulation tool-flows (such as MATLAB or specific neuromodeling tools like NEURON or Brian) are executed, are not up to the task of simulating neural networks of realistic sizes and high detail within a reasonable timeframe for brain research. High-performance computing (HPC) has been recently recognized as being able to provide a variety of solutions to cope with this limitation [2–7]. Unfortunately, the challenge of executing such simulations does not stop just at providing the necessary computational power.

In scientific applications such as neuronal simulations, modeling accuracy has a direct impact on simulation speed. The variety of options of viable SNN models used in studies is significant. Every type of model has scientific merit, depending on the subject under study, and models exhibit different characteristics when treated as computational workloads [6, 8]. Modeling features like the inter-neuron connectivity density (the modeling of which also varies according to the biological system under study) can break the embarrassingly parallel (data-flow compatible) nature that most neuron models have, significantly changing the behavior of the application.

Depending on the desired model characteristics, we identify two general types of simulations that are relevant in neuroscientific experiments. The first one has to do with highly accurate (biophysically accurate and even accurate to the molecular level) models of smaller-sized networks that requires *real-time* or close to real-time performance. These kinds of experiments can be used with artificial real-time setups or brain–machine interfaces (BMI) and are closely related to brain-rescue studies (TYPE-I experiments). The second type involves the simulation of large- or very large-scale networks in which accuracy can often be relaxed. These experiments attempt to simulate network sizes and connection densities closely resembling their biological counterparts (TYPE-II experiments) [5, 9]. This, in combination with the variety of models commonly used, makes for a class of applications that vary greatly in terms of workload, while also requiring high throughput, low latency or both. A single type of HPC fabric, either software- or hardware-based cannot cover all possible use cases with optimal efficiency.

A better approach is to provide scientists with an acceleration platform that has the ability to adjust to the aforementioned variety of workload characteristics. A heterogeneous system that integrates multiple HPC technologies, instead of just one, would be able to provide this. In addition, a framework for a heterogeneous system using a popular user interface for all integrated technologies can also provide the ability to select a different accelerator, depending on availability, cost and performance desired.

Such a hardware back-end must overcome additional challenges to be used in the field. It requires a front-end which should provide two crucial features:

- An easy and commonly used interface through which neuroscientists can employ the platform, without the constant mediation of an engineer.
- A front-end that can reuse the vast amount of models already available to the community.

In this paper, we propose a framework for an *heterogeneous acceleration platform* for computationally challenging neuroscientific simulations called *BrainFrame*. By using this system, we demonstrate the effect of model characteristics on performance and thus make a concrete case for the significance of employing heterogeneity in HPC systems used the field of computational neuroscience. To this end, we use a state-of-the-art, extended Hodgkin–Huxley (biophysically-meaningful) model [10] of the inferior-olivary nucleus as a benchmark to evaluate the framework. We chose this model as a respective workload of such neuron representations, as their efficient simulation poses a significant engineering challenge. Even though this model is not the most biophysically accurate representation in the field, it is one of the most accepted and widely used models for brain simulations. We evaluate BrainFrame using three distinct instances of the workload, each differentiated by the presence and complexity of the neuron interconnectivity modeling, leading to vastly different computational requirements, while still reflecting realistic neuroscientific experiments. We propose a front-end for the framework based on the PyNN language [11]. PyNN has been widely adopted by the computational-neuroscience community and has direct integration with many other well-known neuron modeling frameworks, covering both aforementioned features that such a front-end would require.
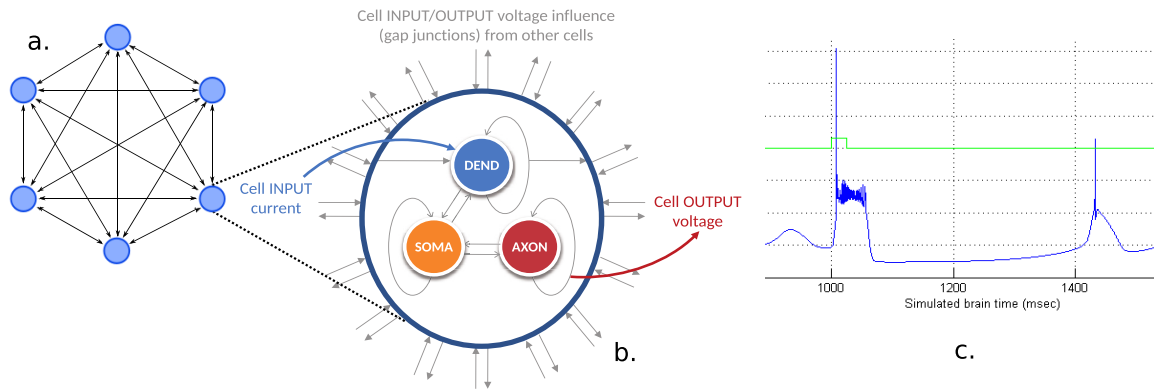
Figure 1. Graphical representation of the inferior-olivary network model. (a) 8-neuron network (b) single-neuron model in detail (c) sample axon response.

2. Methods

2.1. The inferior olive

The inferior-olivary nucleus forms an intricate part of the olivocerebellar system, which is one of the most dense brain regions and plays an important role in sensorimotor control. Activity in the inferior olive probably only directly triggers movements when it is synchronized among multiple neurons [12, 13]. In addition, the olivary neurons can provide rhythm and coordination signals for motor functions [14]. It is considered to be imperative for the instinctive learning and smooth completion of motor actions [15]. The olive provides one of the two main inputs to the cerebellum through the climbing fibers.

What makes the inferior-olive neurons special is their dense interconnection through electrical connections called gap junctions (GJs), which differ from typical synapses in that they are purely electrical. The gap junctions facilitate the synchronization behavior between the olivary neurons and, subsequently, influence the synchronization and learning properties of the entire olivocerebellar system [14].

2.2. The InfOli workload

In this work, a detailed inferior-olive (InfOli) model is considered, which was originally developed by De Gruijl *et al* [1]. It implements a neuron with three distinct compartments, the dendrite, the soma and the axon. Within the dendrite, the model also includes gap junctions (thus the characterization as 'extended', eHH), while the cell output represents the input to the climbing fibers (figure 1(b)). The GJs are associated with important aspects of cell behavior as they are not just simple connections; rather, they involve significant and intricate electrical processes, which is reflected in their modeling details.

Every compartment includes a number of state parameters denoting its electrochemical state and the *neuron state* as a whole. The neuron states are updated at each simulation step; every new state update is based upon: The neuron state of the previous simulation step of the executed neuron, the previous dendritic states coming from the GJ connectivity and the externally evoked input to the network, representing the input coming from the rest of the cerebellar circuit.

The three compartments and GJs are evaluated/updated concurrently at each simulation step. The model is calibrated with a simulation time step of $\delta = 50$ μs. Simulations steps are identical to each other in terms of operations performed. This simulation step also defines the *real-time* behavior of the whole network. Figure 1(a) depicts a representation of the InfOli network model. This network model effectively defines a transient simulator through computing discrete axon output values in time steps which, when integrated over time, recreate the output response of the axon (figure 1(c)).

The InfOli network must be synchronized in order to guarantee the correct exchange of previous dendritic data within a step. Thus, execution can only be parallelized in space (simultaneous evaluation of neurons within a simulation step), but not in time (parallelization of multiple simulation steps). The cells—even when not actively spiking—present an oscillatory behavior, thus affecting network synchronization. As a result, event-driven execution of the network model is not an option.

By profiling the application using an operation and memory-access profiler [16], it is revealed that the GJs have great impact on the total model complexity. As seen in table 1, the total number of floating-point (FP) operations needed for simulating a single step of a single cell including a single GJ are 871[6]. For many complex experiments, it is not the number of connections but, rather, the *connectivity density* (C) that is indicative of neuron interconnectivity. That is, the average percentage of the total neuron inventory to which neuron cells are connected (measured in %), whereby the complexity becomes quadratic. This makes GJ computations the prevalent contributor, as they break the dataflow nature of the application and dominate computational demands. This is true even for small-scale networks. As an example, for a 96-cell, all-to-all connected network (table 1) the GJs comprise almost 60% of the overall computations.

2.3. Application use cases

For our analysis, we employ three use cases drawn from [8], which are representative of the memory and computational requirements of the InfOli workload. All of the use cases are realistic instances of the InfOli application and have

---

[6] Table numbers have been updated to amend a profiling mistake reported in previous work [8].

Table 1. Neuron compute requirements per simulation step.

Computation	FP operations/neuron
Gap junction	12 per connection
Cell compartment	859
I/O and storage	**FP operations/neuron**
Neuron states	19
Evoked input	1
Connectivity vector	1 per connection
Neuron conductances	20
Axon output	1 (Axon voltage)
Neuron computation task	**% of FP ops for 96 cells**
Compartmental computations	43
Gap junctions	57
Computations per step:	$859 * N + 12 * N^2 * C$
	C: connectivity density
	N: network size

Figure 2. Floating-point operations required per simulation step of the InfOli model for each use case and for different connectivity density percentages (%).

Figure 3. Compute-to-memory-access ratio per simulation step of the InfOli model for each use case and for connectivity density percentages (%).

neuroscientific merit. They can also be considered as plausible instances of multi-compartmental modeling using HH models with various cases of modeled inter-neuron connectivity.

The application allows for the connectivity of the InfOli network to be programmable by the user before the simulation is deployed. Network connectivity (when present) is defined by an $N \times N$ *connectivity matrix* (where N: Network size) of FP weights signifying the weight of each connection. Weights are used in the GJ computations to calculate the connection impact on each neuron. The three use cases which focus around the biological complexity of the GJs:

(i) **InfOli with realistic gap junctions (RGJ)**—InfOli HH cells modeled with (biophysically) realistic GJ kinematics as presented in [1]. The highest amount of detail is included in the GJ modeling.

(ii) **InfOli with simplified gap junctions (SGJ)**—InfOli HH cells modeled with simplified GJ kinematics. This constitutes a simpler connectivity compared to the RGJ use case.

(iii) **InfOli with no gap junctions (NGJ)**—InfOli HH cells modeled without GJ kinematics modeling. This is the simplest use case, whereby the neurons are modeled as independent computational islands.

In figure 2, we see the amount of FP operations, based on the aforementioned profiling of the InfOli application. The FP operations are calculated for each of the aforementioned use cases for different connectivity densities. From the same profiling run we can derive the compute (in FLOPS) to memory (in single-FP memory accesses) ratio for the application, that reveals whether each use case is computation- or memory-bound (figure 3).

2.3.1. 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
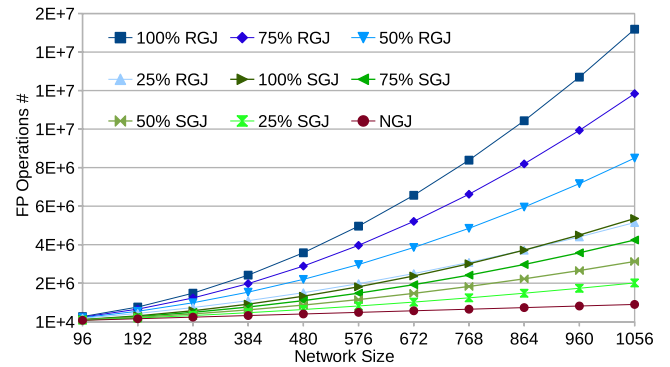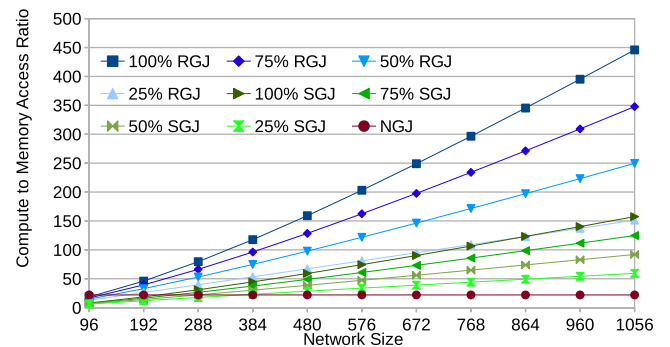
representation of the biological nucleus (algorithm 1). Each cell C in a population of N cells accumulates the influence of an interconnected cell to it (through a GJ) by subtracting its own dendritic voltage (`prevVdend`) from the dendritic voltage of that cell (`neighVdend[i]`). It, then, accumulates the resulting voltage influence in an aggregate current `Ic`, by factoring in the respective GJ-connection weight (`C[i]`).

The compute-to-memory-access ratio (from figure 3) sug-

Algorithm 1. Example of RGJ implementation in C.

```
1: for i = 0; i<InfOli_N_INPUT; i++ do
2:        V = prevVdend—neighVdend[i];
3:        f = 0.8*V*exp(-1 * V * V/100) + 0.2;
4:        Ic = Ic + (C[i] * f * V);
5: end for
6: return Ic;
```

gests also that this use case is strongly computation-bound for all connectivity cases: With increasing problem sizes, the computations increase at a much faster pace than the memory-access requirements.

2.3.2. InfOli with simplified gap junctions (SGJ). The level of detail as in the RGJ case is useful for many modeling experiments but is also an overkill in many other cases that more simple rudimentary connection are involved (like simple

synapses that accumulate inputs). Lighter workloads are represented by the SGJ case. We assume a use case of the InfOli application that simplifies the connection between neurons to a few simple input accumulators. The computations per simulation step are now $859 * N + 4 * N^2 * C$. The accumulation is parameterized using the weight that is assigned to each connection between two neurons, thus the connectivity information needs to be accessed the same way as is in the RGJ case. The actual FP operations are reduced by about one order of magnitude compared to the previous use case (see figure 2). Yet, the connectivity aspect still disrupts the pure dataflow nature of the model. A high compute-to-memory ratio is seen here as well, since the computations still increase at a faster pace than the memory requirements.

2.3.3. InfOli with no gap junctions (NGJ). This is the case where the application becomes purely dataflow and can achieve the greatest parallelism possible. The processing requirements scale almost linearly with the network size and, compared to the other use cases, fewer computations are needed, as shown in figure 2 (computations per simulation step : $859 * N$). As we can see in figure 3, although the NGJ use case shows that computation is still the most important aspect of the application, both computation and memory access scale linearly and at a similar pace.

2.4. HPC fabrics and implementation

Our heterogeneous platform incorporates three accelerator fabrics; a Maxeler *Maia* data-flow engine (DFE) board [17], an Intel Xeon Phi 5110P CPU [18] and a Maxwell-based Titan X GPU by NVidia [19] (table 2). All there boards are PCIe-based which is how they communicate with the host system. The use of PCIe interfaces ensures that composition of BrainFrame-enabled machines can been easily tailored on a per-case basis depending on the availability of funds and hardware resources of a research laboratory. Different types and mixes of PCIe-based accelerators can be selected.

The Maia DFE is a Maxeler HPC technology 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 uniquely massive pipelines, maximizing the throughput and overall performance. The DFE boards also incorporate a high-bandwidth, multichannel, highly parallel, customizable interface to the onboard DRAM memory resources (up to 96 GBs) making it ideal for scientific applications. The DFE board used in our experimental setup is a 4th-generation Maia-DFE board implemented using an Altera Stratix V 5SGSD8 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 generation of Phi cards used in this work, named Knights Corner, are programmed using well-known programming tools such as OpenMP and

OpenCL. However, and in contrast to GPU mentality, the Phi can also be thought of as an accelerator that can act as a standalone processor and even features its own operating system. This is expected to increase memory-consistency and cache-coherency delays.

GP-GPUs have also been prominent in the HPC domain and in scientific computing in particular. The Titan X includes 3072 CUDA micro-cores, which are used to parallelize computation execution, and 12 GB of on-board RAM. GPU implementations also benefit from the generally good adoption of the NVidia CUDA-library open environment that allows porting of applications with similar ease to the Phi OpenMP and OpenCL frameworks. GPUs also come at a relatively lower cost than the other two accelerator types. However, as opposed to the the Xeon Phi, a GPU cannot act as its own host increasing communication delays between host and accelerator during execution.

Lastly, it must be noted that BrainFrame is to be used in scientific research that is very dynamic and fast-paced. The goal is not to over-optimize the different accelerator implementations, but to propose and maintain a balance between the programming effort and optimization needed, resulting in shorter development times for cutting-edge research tools. In real research, such development times should be kept short so as not to delay the scientific process.

2.4.1. Infoli on the maia DFE. The DFE implementation of the InfOli application can be seen in figure 4 and is a more advanced version of the work done in [20]. New features include the addition of programmable connectivity and programmable neuron state by the user between experiment runs without the need to re-synthesize the design. The design implements three pipelines on the DFE hardware to accelerate the application, one for each part of a neuron (Dendrite, Soma, Axon), executing the respective computations. The state parameters for each neuron are stored on separate BRAM blocks for fast reading/updating of the network state, as they are the data that are most used throughout the experiment execution. Since every new neuron state is dependent only on the network state of the previous simulation step, a single copy of each neuron state is required at any point during execution. The input stream to the DFE kernel originates in the on-board DRAM and represents the evoked (external) inputs, used in the dendritic computations comprising the network input. The initialization data are also streamed in from the on-board memory only once at the start of execution. The size of the connectivity matrix makes it impossible to store on the on-chip memory. It is, thus, placed on the on-board RAM and streamed in batches dictated by the computations. The kernel output is streamed back to the on-board memory and—at the same time—is updated in the (on-chip) BRAM blocks of the DFE.

The program flow is tracked using hardware counters monitoring GJ loop iterations (except for the NGJ case), the neurons executed and the number of simulation steps concluded. The data flows through the DFE pipelines with each kernel execution step (or *tick*) consuming the corresponding input or producing the respective output and new state at the

Table 2. Specifications of the accelerator fabrics used.

| Specification	Maxeler DFE (Maia)	Intel Xeon Phi CPU (5110P)	NVidia GPU (Titan X)
On-board DRAM	48 GB	8 GB	12 GB
RAM bandwidth	76.8 GB/s	320 GB/s	336.5 GB/s
Memory streams/channels	15	16	—
On-chip memory	6 MB (FPGA BRAMs)	30 MB (L2 cache)	3 MB (L2 cache)
Number of chip cores	—	61	3072 CUDA Cores
Chip frequency	Depends on design kernel	1.053 GHz	1 GHz
Instructions set	n/a	64 bit	32 bit
Power consumption (TDP)	140 W	225 W	250 W
IC process	65 nm	22 nm	28 nm
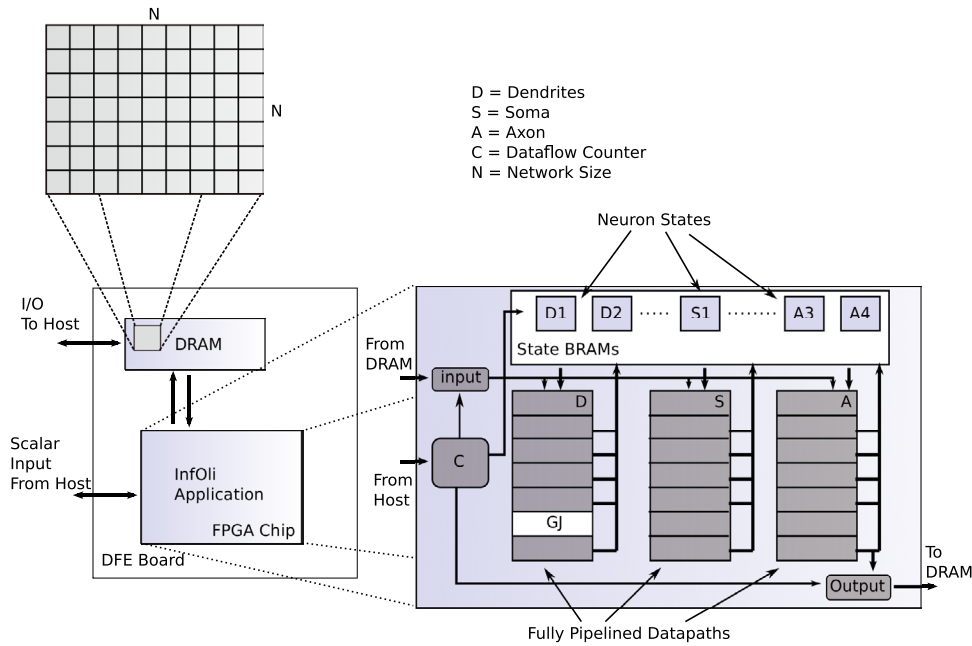
Figure 4. DFE implementation of the InfOli application.

correct execution points according to the hardware counters. DFE execution naturally pipelines the execution of different neurons within one simulation step. Simulation steps are not themselves directly parallelizable, as every neuron must have the previous state of all other neurons available for its GJ computations (only in the RGJ or SGJ cases) before a new step begins. The DFE pipeline is, thus, flushed before a new simulation step begins execution. This dependency is lifted when in the NGJ case. The GJ calculations form a loop that must finish before the rest of the dendrite-compartment state is calculated. The rest of the dendrite pipeline does not produce valid data for the operation ticks that the GJ influence is being calculated. This delay is partially amortized by using hardware loop unrolling on the GJ calculations, but only to the point that the available chip area allows it. Additionally, in use cases where programmable connectivity is included, the ticks for the evaluation and execution of a GJ connection are always spent regardless of whether a connection actually exists or not. Thus, this implementation cannot benefit from a smaller connectivity density in terms of performance. On the other hand, since one synthesized design can account for all possible connectivity scenarios, the DFE implementation can guarantee predictable performance under all use cases.

2.4.2. Infoli on the Xeon Phi. The InfOli application on the intel Xeon Phi co-processor, depicted in figure 5, is based on a typical shared-memory implementation. The application uses the OpenMP library to spawn threads, which can work in parallel. As the Xeon Phi 5110P uses one core to handle OS-related tasks and each core features multithreading technology that can service up to 4 instruction streams simultaneously, the InfOli application on the Xeon Phi uses up to $60 \times 4 = 240$ OpenMP threads. Each thread is programmed to handle a part of the neuronal network (sub-network), which is partitioned as uniformly as possible to prevent workload imbalances.

In each simulation step, every OpenMP thread computes its sub-network's state. This process is further broken down into two tasks. Initially, the sub-network needs updated information from the rest of the network, specifically the dendritic-membrane voltage of the other neurons connected with this sub-network (recall algorithm 1). Thus, each OpenMP thread accesses memory space shared by all threads so as to collect data from other neurons, with the purpose of re-evaluating the state of its sub-network's GJs. In this task, shared-memory accessing can cause stalls in thread operations due to issues such as memory contention.
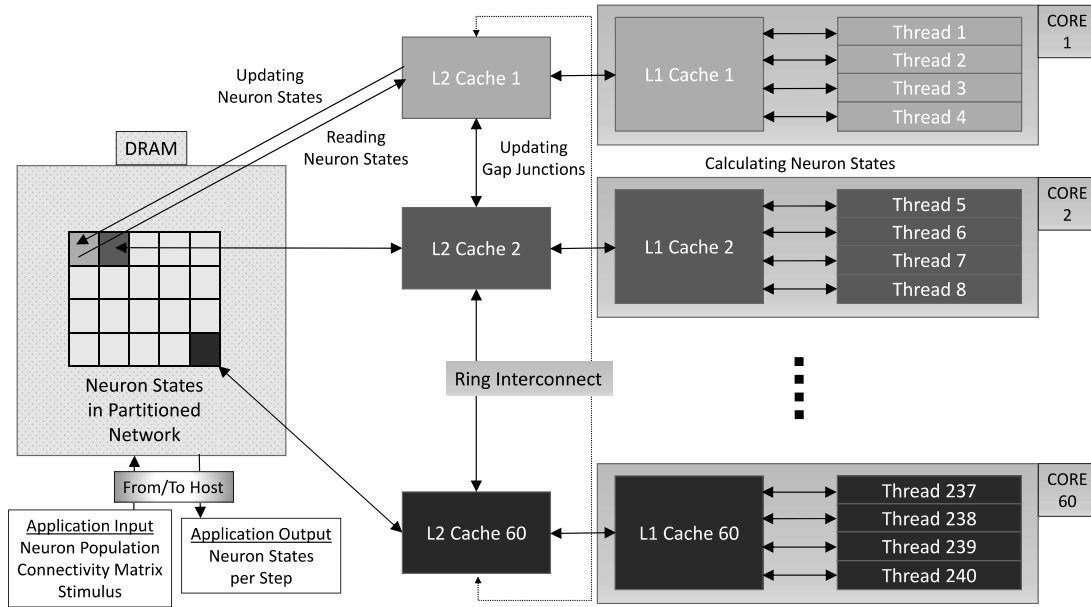
Figure 5. Xeon-Phi implementation of the InfOli application.
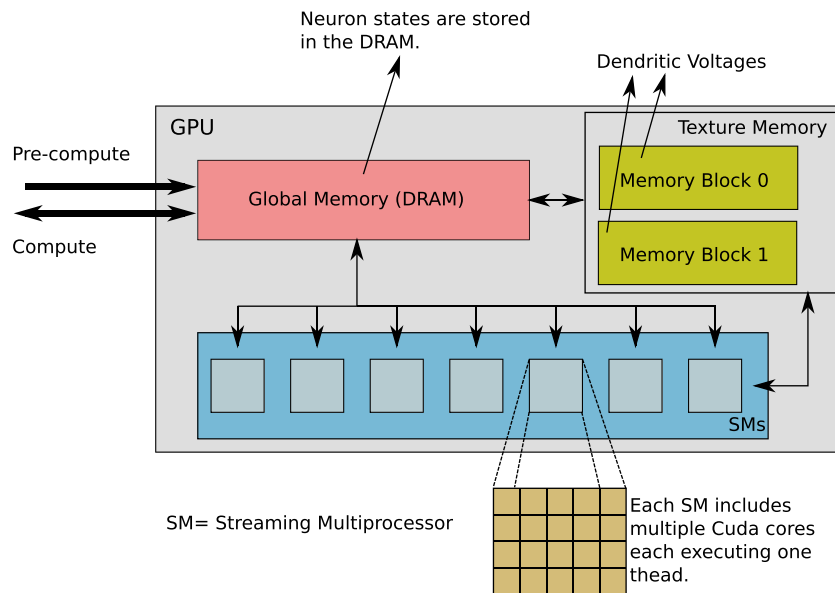

Figure 6. GPU implementation of the InfOli application. Pre-compute and compute operations are issued by the host.

Upon completion of its first task, each OpenMP thread updates the compartmental states of each neuron in the sub-network. Each of the neuron's three compartments is re-calculated (dendrite, soma and axon). The dendritic compartment specifically uses the updated GJ states evaluated in the previous task in order to assess the incoming current from connected neurons. As already explained, this particular process demands an amount of operations that increases significantly with neuron population in the case of densely connected networks, as we would expect with the increasing computational demands of GJs.

After performing its two tasks for the entirety of its sub-network, each OpenMP thread begins the process anew for the next simulation step, until there are none left. Under this paradigm, the threads operate constantly within the 'timeframe'

of the same simulation step. They sync before the execution of a new simulation step, so that stale data from previous simulation steps cannot be exchanged during GJ computation. This behaviour is enforced due to the stiffness of the eHH-model equations, which can be thrown off-balance even by small changes in the numerical data within a single time step. Under a more relaxed model (e.g. a typical HH model), some staleness in data exchange would be more tolerable and the user would be able perform thread synchronization less frequently in order to trade precision for execution speed. Furthermore, it should be noted that the implementation described assumes that the entire network is large enough to be partitioned in 240 parts. When dealing with smaller networks, the implementation utilizes less than the maximum amount of the platform's assets, since it is designed to require at assign one neuron on each OpenMP thread.
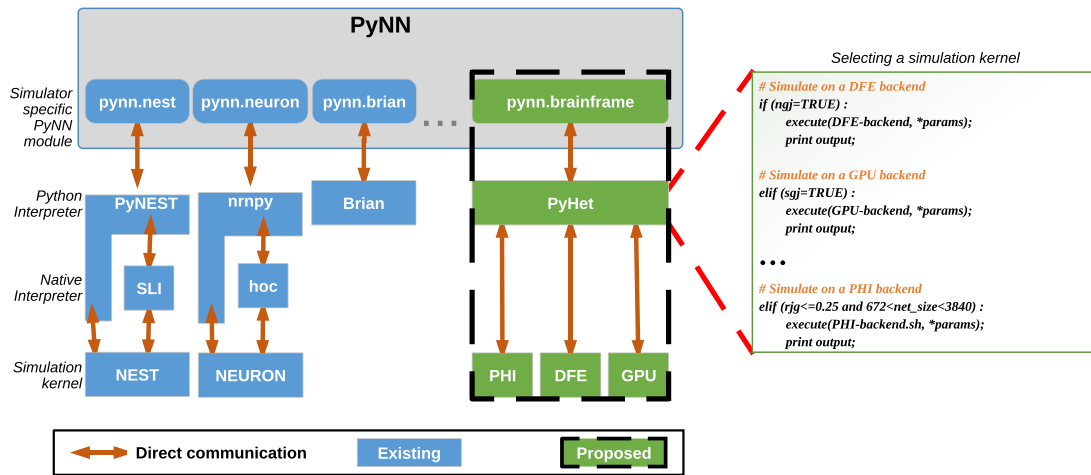
Figure 7. PyNN architecture and the proposed BrainFrame framework.

2.4.3. Infoli on the titan X GPU. In figure 6, we can see the InfOli implementation on the GPU. The execution flow includes two stages, a pre-compute and a compute stage.

In the pre-compute stage, the host initializes the neuron states and the external input currents for the entire simulation duration. It allocates global memory on the device to store the current-step neuron states, next-step neuron states and the external input currents. At the end of this stage, the host copies the required data for simulation onto the GPU. Similarly to the other two accelerator implementations, the current-step dendritic voltages of all cells are accessed frequently as they are used to determine the GJ influence. To reduce memory latency, they are bound to the GPU texture memory. The texture memory is a cached memory on the GPU used to reduce memory latencies when the application has specific memory-access patterns. Writes to texture memory, during the compute stage, are conducted only after all computations of a simulation step have finished. It must be also noted that after the pre-compute stage, no data is transferred from the host to the GPU; the GPU contains all necessary information for the simulation.

During the compute stage, the neuron calculations are performed and the new states are persistently stored throughout the simulation duration. To compute the new states for a single simulation step, the host launches a CUDA kernel on the GPU device. Before simulation, the kernel is configured for a particular use case (RGJ, SGJ or NGJ) and inter-neuron connectivity scheme (if applicable). The kernel is executed by a two-dimensional grid of CUDA threads on the device. Threads are executed in parallel by the CUDA micro-cores of the GPU. Every InfOli cell of the model is mapped to a corresponding thread that calculates the states of the neuron. On kernel completion, the host receives the calculated result of the simulation step from the device. The host uses two operation streams to issue the kernel execution and data-transfer operations to the GPU. A kernel in one stream is launched only when the kernel in the other stream has completed. Thus, when one stream is computing the currently executing simulation step, the other stream is performing the necessary data transfers to the host from the GPU. Since the texture

memory is updated only after the kernel completes execution, data coherency is maintained. Thus, computation of the current-step neuron states and data transfer of the previously computed states overlap, effectively hiding Host-to-GPU transfer delays.

2.5. BrainFrame and the PyNN front-end

PyNN is a Python package that facilitates the interchangeability and the study of different simulation environments within the computational neuroscience community [11]. It allows for simulator-independent specification of neuronal-network models and already supports many popular simulators like NEURON, NEST, PCSIM , Brian, and so on.

The PyNN API supports modeling at multiple levels of abstraction, both at the neuron level and the network level. It provides a library of standard neuron, synapse and synaptic-plasticity models and a set of commonly-used connectivity algorithms while also supporting custom user-defined connectivity in a simulator-independent fashion.

We integrated the three accelerator fabrics as back-ends on the BrainFrame system using PyNN as a front-end. The PyNN integration provides the neuroscientific community with easy access on the accelerators without constant mediation from the acceleration engineer while also providing an interface for the already established models to be used with the new heterogeneous acceleration back-end. These characteristics of PyNN can have decisive impact on the adoption of BrainFrame by the community.

As a proof of concept for the front-end of the BrainFrame platform, we have added the InfOli model the library of standard PyNN models. Following the PyNN paradigm, the user initially selects the simulator—in our case our BrainFrame simulator—and then proceeds to select the neuron model, in our case the Inferior-Olive model. A population of neurons using the chosen model is then generated, determining the inter-neuron connectivity type and, finally, a projection of the specified neuronal network is created.

The main difference between the proposed PyNN-backend substrate and the typical simulator back-ends within the PyNN environment is an additional selection step. In this step, a decision about which of the three alternative acceleration fabrics will be used for a specific experiment is made, based on the available hardware and the characteristics of the simulated neural network.

A conceptional view of the architecture of the PyNN BrainFrame module is shown in figure 7. For the simulator kernels to communicate with the PyNN frontend, a intermediate BrainFrame-specific PyNN module (pynn.brainframe) is required that implements and extends common methods and objects like the neuron models, synapse models and projection methods and objects. In the case of the proposed BrainFrame module, we implemented objects and methods: (i) for the initialization of the simulator, (ii) for the description of the neuronal network in PyNN, and (iii) for controlling the simulation execution. In some cases, an additional interpreter module is needed to translate these Python objects and parameters to each simulator's native parameters and language. For our system, we developed PyHet—the BrainFrame-specific Python interpreter—which serves the aforementioned role and also implements the accelerator selection.

The final BrainFrame system will be implementing more generic kernel libraries that will be used by the PyNN frontend to simulate user defined models. That way, the accelerator implementation will be completely transparent to the user and predictions can then be made based on the analysis of the individual kernels that can guide the selection algorithm.

3. Results

In this section, we present a thorough performance analysis of our heterogeneous BrainFrame platform. The goal is to evaluate the platform and give a clear view on how each accelerator performs when running various instances of the InfOli use cases, validating the usefulness of an heterogeneous HPC simulation framework for computational neuroscience. The performance analysis also acts as a guide for proposing an *accelerator-selection algorithm*.

To validate the correct functionality of the separate accelerator implementations, we use a simple experiment that recreates a typical response that is found in the inferior-olive network (axon response). In this experiment, each cell produces a so-called complex spike, seen in figure 1(c), from all simulated cells. 6 s of brain time are simulated, which translates to 120 000 simulation steps. The complex spike is produced by applying a small current pulse as input to all InfOli cells at the same instance after program onset, for about 500 simulation steps (or 25 ms, in brain time). Despite being rudimentary, this experiment is easy to validate, provided all neurons are initialized with the same state, and also gives a good indication whether synchronization between neurons is correct, thus validating cell interconnectivity (when present).

As mentioned in the introduction, we identify two distinct tracks that can be followed in conducting neuroscientific experiments, both covered in this evaluation. We perform one

batch of measurements ranging from 96 to 960 neurons representing small-scale, real-time TYPE-I experiments, and a second batch ranging from 960 to 7680 neurons representing larger-scale TYPE-II experiments). From our in-house neuroscientific experiments, we know the minimum network size for meaningful experiments to be around 100 neurons, thus our measurements for TYPE-I experiments begin at 96 neurons. The evaluation is focused on the performance of single-node accelerators, thus a network-size cap is set by the smallest maximum network supported by each of the three accelerator paradigms: in this case, the DFE fabric limits network sizes to 7680 cells.

3.1. Performance evaluation

All performance measurements concerning the Xeon Phi have been carried out through the VTune amplifier XE 2015 profiling and analysis tool by Intel. Timing measurements on the Maia DFE were taken by measuring the DFE-kernel time inlined within the host code using timestamps before and after the kernel call. Since, the host code (in the CPU) is blocking, only the DFE kernel is active during measurements. 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 test experiment. GPU kernel-time measurements were taken using the CUDA Event API.

3.1.1. TYPE-I experiments. Starting with the analysis for TYPE-I experimentation, in figure 8 we plot the execution time of a single simulation time-step (50 μsec) for the most demanding use case, that of the RGJ with 100% connectivity density. Even though it is not the most common case, a brain-simulation platform must support such high interconnectivity densities for certain TYPE-I experiments. The DFE exhibits the best performance for all tested network sizes. The Xeon Phi is a close second due to the local-memory delays and the less efficient use of its parallel threads: These network sizes are not large enough to provide sufficient parallelism for the Phi threads to be fully utilized. The GPU, on the other hand, has difficulties to cope with the computational intensity of the GJs, which involve mostly division and exponent FP calculations. Since each CUDA thread executes a single neuron, it cannot exploit any potential parallelism in the GJ calculation. This, alongside the fact that the CUDA threads are underutilized at such network sizes, impacts performance drastically.

The inefficiency of the Titan X GPU in performing the realistic GJ computations is clearly revealed in the SGJ case, next (see figure 9). In this use case, that the most demanding GJ calculations are dropped, the GPU presents excellent scalability as the problem size increases, compared to the RGJ case. The Xeon Phi, on the other hand, still suffers from core-to-local-memory synchronization delays even though the actual calculations are much simpler now. The DFE needs to spend the same amount of operation ticks as in the RGJ case to evaluate the connection influence, even though it does enjoy gains in performance because of the simpler calculations involved (achieving
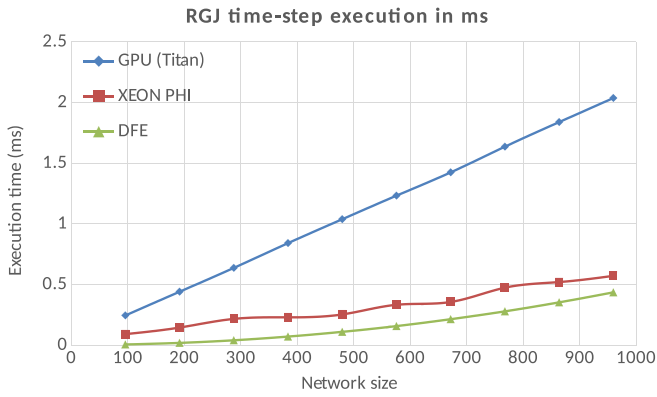
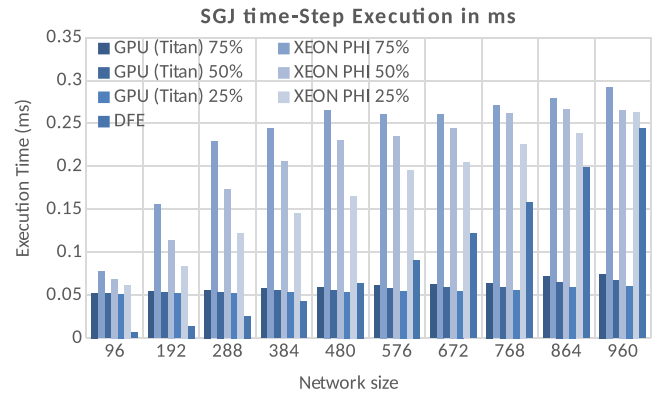**Figure 8.** RGJ execution time (TYPE I, 100% connectivity).

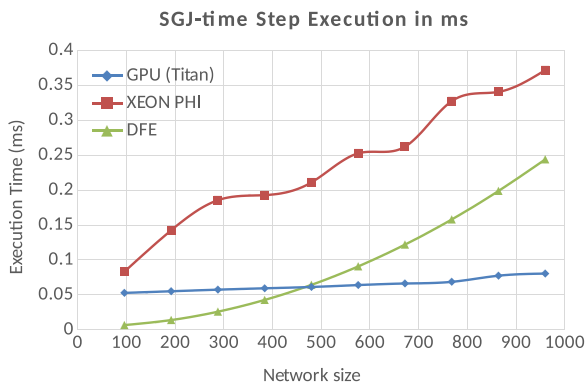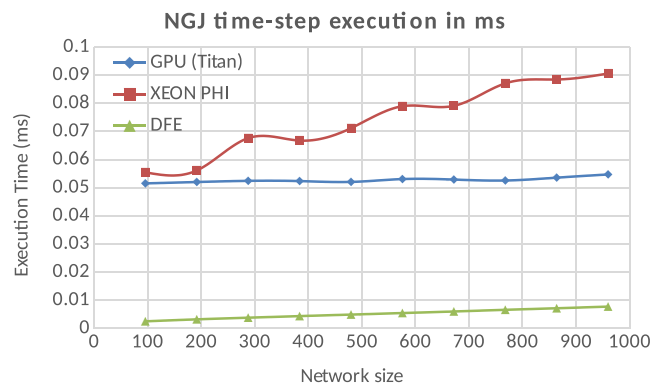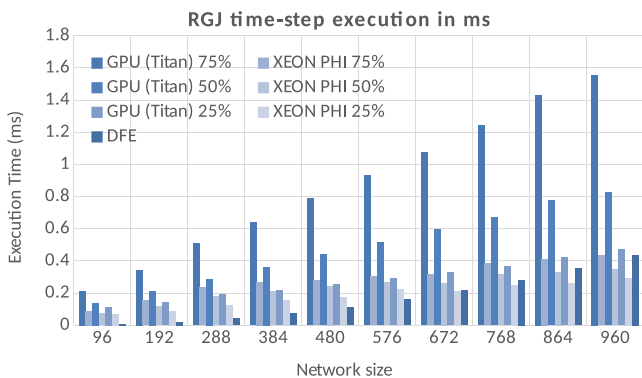**Figure 9.** SGJ execution time (TYPE I, 100% connectivity).

Figure 10. RGJ execution time (TYPE I, <100% connectivity).

higher operation frequencies, larger GJ computation parallelism and shorter pipelines). As a result, both latter accelerators show similar scaling properties to the RGJ case. In contrast, the GPU scores performance benefits in the SGJ case compared to the robust DFE for network sizes above 480 neurons.

Next, it is interesting to evaluate the three accelerators for connectivities of lower than 100% density. Although not relevant for the DFE which maintains the same implementation for any connectivity density, smaller densities can influence the Xeon Phi and the GPU performance considerably. In figure 10, we plot the execution time of a single simulation time-step for 25%, 50% and 75% connectivity densities, under the RGJ case. The GPU delivers significant gains but the inefficient GJ execution still causes it to perform worse than DFE, even though the latter operates as in a 100%-density

Figure 11. SGJ execution time (TYPE I, <100% connectivity).

Figure 12. NGJ execution time (TYPE I, no connectivity).

Table 3. RT-achievable network size (#cells) for each use case

Use case	DFE	Xeon Phi	GPU
RGJ (100%)	310	—	—
RGJ (75%)	310	—	—
RGJ (50%)	310	—	—
RGJ (25%)	310	—	—
SGJ (100%)	400	—	—
SGJ (75%)	400	—	—
SGJ (50%)	400	—	96
SGJ (25%)	400	—	96
NGJ	7680	96	500

simulation. The Xeon Phi, on the other hand, manages to achieve enough performance gains to become faster than the DFE for sufficiently large problem sizes; that is, sizes ⩾960 neurons for 75% density, ⩾864 neurons for 50% density and ⩾672 neurons for 25% density.

Under the SGJ use case (figure 11), we see similar trends as for the 100% SGJ use case: The GPU exhibits great scalability and is the best option for network sizes higher than 480 neurons. Besides, the DFE remains the most beneficial option for networks smaller than 480.

Under the NGJ case (no connectivity), for TYPE-I experiments, the results point to the DFE as the uniformly best option. In the complete absence of inter-neuron connectivity, the application becomes a purely dataflow workload, fully compatible for acceleration on a DFE, which is tailor-made for such cases, providing significant benefits over both the Xeon Phi and the GPU (see figure 12).
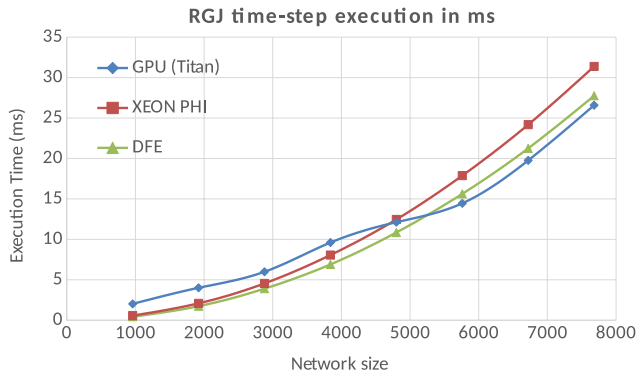
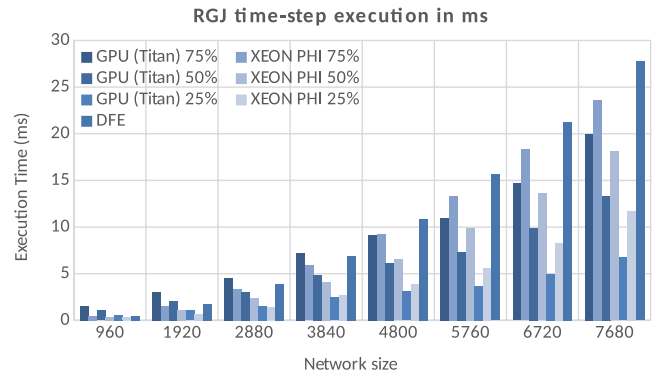**Figure 13.** RGJ execution time (TYPE II, 100% connectivity).

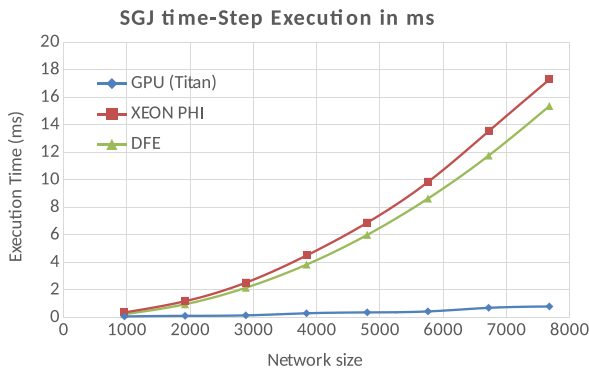**Figure 15.** RGJ execution time (TYPE II, <100% connectivity).

Figure 14. SGJ execution time (TYPE II, 100% connectivity).

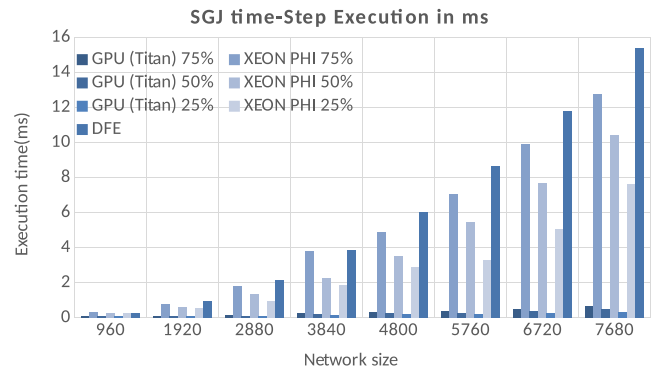**Figure 16.** SGJ execution time (TYPE II, <100% connectivity).

Figure 17. NGJ execution time (TYPE II, no connectivity).

Lastly, recall that for TYPE-I experiments, real-time speeds are often desired. Table 3 presents the real-time achievable networks for each use case. The results show that, for real-time experimentation, the DFE accelerator is the best option across the board. In contrast, and as mentioned in our previous analysis, the GPU and Xeon-Phi parallel threads tend to be underutilized at such small network sizes, even though most of the delays of using them are present. Thus the DFE—using fine-grain super-pipelined kernels—can achieve meaningful network sizes at real-time speeds under all use-case instances, according to the objective set in the introduction ($\geqslant 100$ cells). For low ($\leqslant 50\%$) or zero densities, the GPU and Xeon Phi come close to the real-time objective, yet it is interesting to note that the DFE can even support real-time experimentation for TYPE-II experiments under the NGJ case.

3.1.2. TYPE-II experiments. For TYPE-II experiments, the trends under the RGJ case with 100% connectivity change significantly (see figure 13). Here, the massive explosion of the GJ computations begins to stress the parallelization capabilities of both the Xeon Phi and the DFE. The DFE's efficient parallelization of the GJs relies mostly on its ability to unroll the GJ loop on the FPGA hardware, allowing for more iterations to finish per operation tick. However, the achievable unrolling factor is limited by the available chip area. For network sizes above 1000 neurons, the DFE compiler is forced to reuse a lot of resources in time (as the unrolling factor is reduced with increasing network sizes). In effect, the dataflow paradigm gradually degenerates to a sequential execution, making
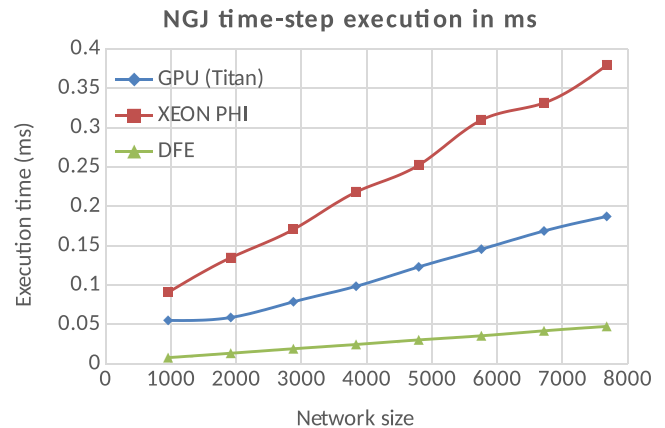
the application less scalable on the DFE. The Xeon Phi follows a similar trend, as the communication overhead between cores (which are interconnected through a moderately efficient ring topology [2]) increases, leading to similarly diminished scalability. Opposite to these accelerators, GPU scalability is largely improved. The GPU is underutilized until all CUDA cores are used (3072) simultaneously, so for experiments over 3000 neurons scalability is gradually improving. As a result, the GPU becomes the better-performing solution (surpassing the DFE) for network sizes of 4800 neurons or more.

For lower connectivity densities under the RGJ case, we observe similar trends, although the Xeon-Phi scalability is slightly better because of the lower interconnectivity (see figure 15). Thus, the Xeon Phi retains the advantages it has
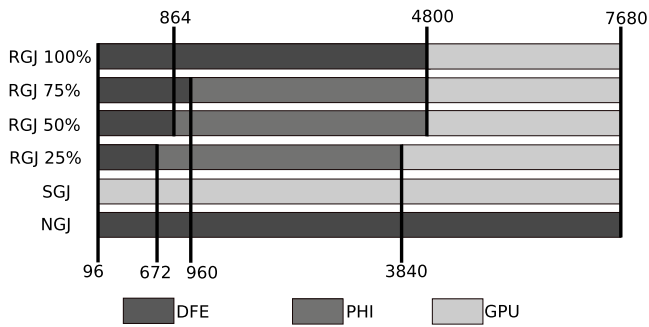
Figure 18. BrainFrame accelerator-selection map for TYPE-II experiments. Selection is heavily dependent on the experiment, involving all three accelerator fabrics. For TYPE-I experiments, the DFE is always the optimal choice (not shown).

for lower than 100% densities, compared to the DFE. Still, the effect of the inter-core communications is present allowing for the GPU to overtake the Xeon Phi for network sizes above 4800 neurons (for densities of 50% and 75%) and above 3840 neurons (for 25% density).

Under the SGJ case, the DFE and Xeon Phi follow similar trends, although they are less pronounced (see figures 14 and 16). As in the RGJ case, the GPU maintains its lead over the other two accelerator types for all tested network sizes and connectivity densities. Finally, in the NGJ case, the situation is the same as with TYPE-I experiments: The purely dataflow nature of the application allows the DFE to once more score the best performance across the board (figure 17).

3.2. Accelerator-selection algorithm

The performance analysis discussed above can now be used to formulate a simple accelerator-selection algorithm for BrainFrame to automatically choose the best-suited accelerator type based on the problem parameters: mainly, connectivity detail (biophysically realistic: RGJ, simple: SGJ and not present: NGJ), density and network size. Figure 18 shows the selection for our use-case instances. The RGJ case selection, which represents the most complex case in terms of accelerator choice, shifts between all three options depending on the connectivity density being simulated. For the SGJ case, the GPU is always the accelerator of choice, while for the NGJ case the DFE yields optimal results under all experiment parameters. Lastly, if the experiment is flagged as a real-time experiment, the algorithm exclusively chooses the DFE to accelerate the application, as it is the only viable accelerator for real-time experiments.

As a simple example of how this selection can speed up experiments, we can assume a scenario where several batches of RGJ experiments need to be executed for various network sizes. Let us assume that each batch includes 5 experiments each with gradually decreasing connectivity density (100%–75%–50%–25%–0%) and that each experiment in a batch simulates 40 s of brain time. The time savings in this example by using the BrainFrame system compared to homogeneous systems that integrate *only a single* accelerator type can be seen in table 4.

Table 4. Time savings (in minutes) with BrainFrame for the assumed experimental scenario compared to three homogeneous-accelerator systems. The % savings are shown in parenthesis.

	BrainFrame versus		
Network Size	DFE-only	Titan X-only	Phi-only
384	0.0 (0.0%)	24.2 (86.2%)	8.6 (68.7%)
960	3.2 (13.8%)	45.8 (69.5%)	3.0 (12.8%)
5760	1.9 (43.4%)	54.5 (27.0%)	10.7 (6.8%)
7680	591.7 (40.0%)	1.9 (0.2%)	246.6 (21.7%)
All batches	707.7 (40.0%)	126.4 (10.7%)	268.9 (20.2%)

Table 5. Energy savings with BrainFrame for the assumed experimental scenario compared to three homogeneous-accelerator systems. We assume nominal (TDP) power figures (see table 2).

	BrainFrame versus		
Network size	DFE-only (%)	Titan X-only (%)	Phi-only (%)
384	0.0	91.4	82.5
960	38	86.4	64.9
5760	51.3	60.9	55.1
7680	23	20.4	43.8
All batches	27.3	32.6	45.9

The BrainFrame system can achieve significant benefits compared to the single-accelerator systems that can lead up to 86% execution-time reduction. On average, assuming the total runtime of all batches, the BrainFrame system can achieve 40% savings compared to a DFE-only system, a 10.7% savings to a GPU-only system and a 20.2% savings compared to a Phi-only system.

If we consider the nominal scenario of TDP-level power consumption we can also present an estimation of the energy benefits of using BrainFrame compared to the single node accelerators for our example (table 5). The energy savings for specific batches of the example range between 20.4% and 91.4%. When running all experimental batches the energy savings ranges between 27% and 45.9%. Reduction in energy consumption can greatly reduce operation and maintenance costs especially within a datacenter environment.

Although these figures will vary based on the particular accelerator models used for the experiments, they give a rough estimate of the time savings that can be obtained by carefully selecting the accelerators for the various experiments. This selection can be easily extended/updated as new features and more generalized model libraries are added for acceleration (making the selection predictive for general cases) or as each acceleration technology is updated in the future.

4. Discussion

There are numerous related works that propose employing HPC solutions for the acceleration of SNNs. Such solutions include hardware-based approaches, like reconfigurable hardware, as well as software approaches using GPUs and less often many-core processors platforms, such as the Xeon Phi. Simpler modeling has found a good match on GPU-based systems, such as Izhikevich and I&F modeling [7, 21]. Higher

biophysically meaningful modeling, like the extended-HH model, seems to be a much more difficult problem to solve with GPUs, especially for real-time experimentation [3].

The Xeon Phi has also been used very successfully for bio-inspired neural networks, such as convolutional neural networks for deep learning systems [22]. On the other hand, similar difficulties to the GPUs in the acceleration of the complex HH models, are identified with Xeon Phi platforms even for less densely interconnected networks [2].

FPGA-based solutions have been especially prominent in accelerating neuron applications, with impressive results specifically for biophysically meaningful modeling and real-time performance for such networks [4, 5, 23]. It is also revealed in related works conducting performance analysis, that an FPGA's potential benefit varies greatly between SNN types, even without taking into account connectivity modeling that can decisively change the workload characteristics [6].

Recently, we have also seen use of DFEs for accelerating computational-neuroscience applications. On purely dataflow neuromodeling applications, the DFE can have great benefits for both large-scale networks and real-time networks performance [24]. Even in the cases of HH neurons that include highly accurate interconnectivity modeling (disrupting the purely dataflow nature), the DFEs can accomplish greater benefits than traditional control-flow-based FPGA acceleration [20].

These works, however, present just one-off implementations of specific application instances, on a specific acceleration platform and most also ignore the variety of synapse-modeling alternatives and its influence on the applications. Biophysically accurate models of biological systems, such as the ones using the HH formalism, are comprised mostly of a set of computationally challenging deferential equations often implementing an oscillatory behavior. If neurons are simulated as independent computational islands (NGJ case), then dependencies between the equations do not arise, allowing divide-and-conquer, data-flow and event-driven acceleration strategies to be used very efficiently. The moment interconnectivity between oscillating neurons is also modeled (like GJs, input integrators, STDP synapses etc), the cells become coupled oscillators. The embarrassingly parallel and dataflow nature of the application is then broken. All neuron states need to be completely updated at each simulation step to retain correct functionality. This requirement, in turn, enforces the use of cycle-accurate, transient simulators and forbids event-driven implementations. As a result, a single HPC accelerator type cannot be a universal answer to the problem since it is unable to cover all the aforementioned requirements efficiently, as our analysis also reveals.

The above difficulties strongly hint on why most of the computational-neuroscience community has so far meticulously avoided employing HH models and multi-compartmental models with complex connections on large problem sizes using conventional computing machines. The eventual use of biophysically plausible neurons and connections on a larger scale is anticipated to contribute substantially in explaining biological behavior. Even though the details of the most important system behaviors of the modeled systems must revolve around very specific characteristics of the networks, thus can possibly to be revealed by generally simpler representations, the computational neuroscientist cannot know beforehand which of the numerous dynamics revealed from the biological measurements (from which the models arise) can be safely abstracted. Studies seeking to reveal system behavior need to start with complex representations before they know enough to drop back to more simplified models.

Additionally, most related works seem to suffer from a limited reusability value due to their (often inexistent) user interface. They ignore the challenge of the neuroscientific community adopting the proposed platform and very few propose solutions to that end. Beuler *et al* [25] developed a graphical interface alongside their FPGA-based simulator. Although it does provide ease of use in experiments, it is still confined to only one platform and only one application with limited flexibility to be the basis of a more widely adopted system. Weinstein *et al* [26, 27] took the approach of developing their own modeling language to interface to their acceleration library, the DYNAMO compiler. Despite the limitation of using only FPGAs as the back-end platform, the DYNAMO compiler is a technically complete solution. Unfortunately, it failed to achieve wide adoption by the scientific community as it requires learning a new language and, additionally, the non-trivial process of porting all existing neuron models to the new coding paradigm.

PyNN has also been used in the past to tackle the issue of user interface. One such a system is SpiNNaker [28], which uses PyNN to interface to a neuromorphic network comprised by a many-core system based on ARM processors. SpiNNaker though is focused on simpler modeling paradigms that do not model specific biophysical neuron properties and have more tractable computational requirements. The most promising solution, both in terms of usability and computational ability, was proposed by Cheung *et al* [29] with NeuroFlow. In this work, the researchers integrated PyNN to their DFE-based hardware library. Neuroflow also provides a very complete library of IPs in the back-end, covering a great portion of possible applications. Yet, the system is still integrating a single acceleration platform. What is more, the performance and efficiency analysis is only presented for a single use case of a generally simpler model (Izhikevich) and with connectivity modeling of medium complexity (STDP) and relatively lower density (about 10%). The behavior and performance of the system for the rest of the supported features is not self-evident and is expected to be significantly different, especially for accurate modeling such as the HH and with high connectivity densities, as shown by our performance analysis on the DFE platform. Furthermore, many of the performance benefits are accomplished using event-driven simulations (neurons are evaluated only when their inputs are triggered), that cannot always be employed, as discussed earlier.

To the best of our knowledge, no prior work has considered an heterogeneous acceleration system for coping with the variability of the applications in the field. Additionally, the PyNN integration provides a familiar interface to the neuroscientific community, thus making BrainFrame a complete solution for a node-level heterogeneous system. Even though the current

work introduces BrainFrame in a single-node setup that integrates all three accelerator fabrics using the PCI-e interface, the BrainFrame paradigm is primary designed to support multi-node setups. Such setups can be facilitated in the now up-and-coming heterogeneous datacenters, provided crucial aspects such as low-latency interconnects are tackled. Such a development would lead to a dramatic increase in the size of network populations supported at tractable simulation times, while also providing a way for small-medium-sized labs to use BrainFrame as a service. Thus, enabling them to exploit the benefits of such an HPC platform without suffering the cost of creating and maintaining a local setup.

5. Conclusions

In this paper, we have proposed BrainFrame, an heterogeneous acceleration platform to serve computational-neuroscience studies in conducting the variety of real experimentation often required for the study of brain functionality. We have focused our analysis on biophysically-accurate neuron models, as such models are considered essential for the deeper understanding of the system properties of biological brain networks. In order for the BrainFrame system to cope with the demand for high ease of programming use as well as the computational requirements of the field, we have presented a proof-of-concept HPC platform that integrates three accelerator paradigms already proven in brain simulations. The performance analysis of the system employing use cases that take into account connectivity density and modeling complexity, has revealed that all three fabrics are essential within such a powerful simulation platform so as to optimally serve all possible experimentation cases. The platform, thus, achieves efficient large-network experiments as well as real-time performance for meaningful network sizes ($\geqslant$100 cells).

BrainFrame has been complemented, finally, with a PyNN front-end so as to tackle the much sought usability objective. The PyNN front-end makes the heterogeneous platform immediately accessible to a multitude of prior modeling works, which is an essential strategy for the wide adoption of complex HPC platforms in the neuroscientific community. Furthermore, building on the elegant PyNN infrastructure, a simple accelerator-selection algorithm has also been developed for automatically identifying the most suitable HPC fabric (Xeon Phi, GPU, DFE) per neuroscientific experiment and has been integrated in BrainFrame. Last but not least, all accelerators use PCIe slots to connect to the host system, which greatly amplifies the platform flexibility and permits adjusting the platform hardware depending on the funds and hardware resources available to a research lab wishing to use BrainFrame.

Acknowledgments

ORCID iDs

Georgios Smaragdos ![ORCID] https://orcid.org/0000-0001-7969-9827

References

[1] De Gruijl J R, Paolo B, de Jeu Marcel G and De Zeeuw C I 2012 Climbing fiber burst size and olivary sub-threshold oscillations in a network setting *PLoS Comput Biol* **8** 12

[2] Chatzikonstantis G, Rodopoulos D, Nomikou S, Strydis C, De Zeeuw C I and Soudris D 2016 First impressions from detailed brain model simulations on a Xeon/Xeon-Phi node *Proc. of the ACM Int. Conf. on Computing Frontiers (New York, NY, USA)* (ACM) pp 361–4

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

[4] Smaragdos G, Isaza S, Eijk M V, Sourdis I and Strydis C 2014 FPGA-based biophysically-meaningful modeling of olivocerebellar neurons *22nd ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*

[5] Glackin B, Wall J A, McGinnity T M, Maguire L P and McDaid L 2010 A spiking neural network model of the medial superior olive using spike timing dependent plasticity for sound localization *Frontiers Comput. Neurosci.* **4** 18

[6] Bhuiyan M, Nallamuthu A, Smith M and Pallipuram V 2010 Optimization and performance study of large-scale biological networks for reconfigurable computing *Fourth Int. Workshop on High-Performance Reconfigurable Computing Technology and Applications* pp 1–9

[7] Yamazaki T and Igarashi J 2013 Realtime cerebellum: a large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit *Neural Netw.* **47** 103–11 (Computation in the Cerebellum)

[8] Smaragdos G, Chatzikostantis G, Nomikou S, Rodopoulos D, Sourdis I, Soudris D, de Zeeuw C I and Strydis C 2016 Performance analysis of accelerated biophysically-meaningful neuron simulations *IEEE Int. Symp. on Performance Analysis of Systems and Software Ispass* pp 1–11

[9] Markram H *et al* 2015 Reconstruction and simulation of neocortical microcircuitry *Cell* **163** 456–92

[10] Hodgkin A L and Huxley A F 1954 Quantitative description of membrane current and application to conduction and excitation in nerve *J. Physiol.* **117** pp 500–44

[11] Davison A, Brüderle D, Eppler J, Kremkow J, Muller E, Pecevski D, Perrinet L and Yger P 2008 PyNN: a common interface for neuronal network simulators *Front. Neuroinform* **2**

[12] De Gruijl J R, Hoogland T M and De Zeeuw C I 2014 Behavioral correlates of complex spike synchrony in cerebellar microzones *J. Neurosci.* **34** 8937–47

[13] Hoogland T M, Gruijl J R D, Witter L, Canto C B and Zeeuw C I D 2015 Role of synchronous activation of cerebellar purkinje cell ensembles in multi-joint movement control *Curr. Biol.* **25** 1157–65

[14] De Zeeuw C I, Hoebeek F E, Bosman L W J, Schonewille M, Witter L and Koekkoek S K 2011 Spatiotemporal firing patterns in the cerebellum *Nat. Rev. Neurosci.* **12** 327–44

[15] Gao Z, van Beugen B J and Zeeuw C I D 2012 Distributed synergistic plasticity and cerebellar learning *Nat. Rev. Neurosci.* **13** 619–35

[16] Feenstra C 2011 A memory access and operator usage profiler framework for HLS optimization: using the lucas optical flow algorithm as case study *Master's Thesis* EEMCS, Circuits and Systems, TuDelft

[17] Pell O, Mencer O, Tsoi K H and Luk W 2013 *Maximum Performance Computing with Dataflow Engines* (New York: Springer) pp 747–74

[18] James J and Reinders J 2013 *Intel Xeon Phi Coprocessor High-Performance Programming* (Burlington, MA: Morgan Kaufmann)

[19] NVidia Corporation www.geforce.com

[20] Smaragdos G, Davies C, Strydis C, Sourdis I, Ciobanu C, Mencer O and De Zeeuw C 2014 Real-time olivary neuron simulations on dataflow computing machines *Supercomputing* (*Lecture Notes in Computer Science* vol 8488) ed J Kunkel *et al* (Berlin: Springer) pp 487–97

[21] Nageswaran J, Dutt N, Krichmar J, Nicolau A and Veidenbaum A 2009 Efficient simulation of large-scale spiking neural networks using CUDA graphics processors *Int. Joint Conf. on Neural Networks* (IEEE) pp 2145–52

[22] Viebke A and Pllana S 2015 The potential of the intel Xeon Phi for supervised deep learning *17th IEEE Int. Conf. on High Performance Computing and Communications*

[23] Shayani H, Bentley P and Tyrrell A M 2008 Hardware implementation of a bio-plausible neuron model for evolution and growth of spiking neural networks on FPGA *NASA/ESA Conf. on Adaptive Hardware and Systems* pp 236–43

[24] Cheung K, Schultz S R and Luk W 2012 A large-scale spiking neural network accelerator for FPGA systems *Int. conf. on artificial neural networks and machine learning* pp 113–20

[25] Beuler M, Tchaptchet A, Bonath W, Postnova S and Braun H A 2012 Real-time simulations of synchronization in a conductance-based neuronal network with a digital FPGA hardware-core *Artificial Neural Networks and Machine Learning—ICANN*

[26] Weinstein R K and Lee R H 2006 Architectures for high-performance FPGA implementations of neural models *J. Neural Eng.* **3** 21

[27] Weinstein R K 2006 Techniques for FPGA neural modeling *PhD Thesis* Georgia Institute of Technology

[28] Painkras E, Plana L A, Garside J, Temple S, Davidson S, Pepper J, Clark D, Patterson C and Furber S 2012 SpiNNaker: a multi-core system-on-chip for massively-parallel neural net simulation *Proc. of the IEEE Custom Integrated Circuits Conf.* pp 1–4

[29] Cheung K, Schultz S R and Luk W 2016 NeuroFlow: a general purpose spiking neural network simulation platform using customizable processors *Frontiers Neurosci.* **9** 516