Neurocomputing 329 (2019) 370–383

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Multinode implementation of an extended Hodgkin–Huxley simulator

G. Chatzikonstantis[a,*], H. Sidiropoulos[a], C. Strydis[b], M. Negrello[b], G. Smaragdos[b], C.I. De Zeeuw[b], D.J. Soudris[a]

[a] National Technical University of Athens, School of Electrical and Computer Engineering, Iroon Polytechniou Str. 9, Athens, Greece
[b] Erasmus MC, Department of Neuroscience, Wytemaweg 80, Rotterdam, Netherlands

ARTICLE INFO

Article history:
Received 3 August 2018
Revised 28 October 2018
Accepted 30 October 2018
Available online 5 November 2018

Communicated by Dr. Muhammet Uzuntarla

Keywords:
Computational neuroscience
Intel Xeon Phi Knights Landing
Simulation
Multinode

ABSTRACT

Mathematical models with varying degrees of complexity have been proposed and simulated in an attempt to represent the intricate mechanisms of the human neuron. One of the most biochemically realistic and analytical models, based on the Hodgkin–Huxley (HH) model, has been selected for study in this paper. In order to satisfy the model's computational demands, we present a simulator implemented on Intel Xeon Phi Knights Landing manycore processors. This high-performance platform features an x86-based architecture, allowing our implementation to be portable to other common manycore processing machines. This is reinforced by the fact that Phi adopts the popular OpenMP and MPI programming models. The simulator performance is evaluated when calculating neuronal networks of varying sizes, density and network connectivity maps. The evaluation leads to an analysis of the neuronal synaptic patterns and their impact on performance when tackling this type of workload on a multinode system. It will be shown that the simulator can calculate 100 ms of simulated brain activity for up to 2 millions of biophysically-accurate neurons and 2 billion neuronal synapses within one minute of execution time. This level of performance renders the application an efficient solution for large-scale detailed model simulation.

1. Introduction

The last decade has witnessed a great amount of advances in the field of computational neuroscience. Interest has been peaking globally towards the human brain [1–3], marking it as an endeavour of paramount importance to the academia and industry alike. Neuroscientists have been gradually unveiling details of neuron operation. Using this knowledge, there is a wide research interest in studying the behavior of single-neurons, as well as small networks of neurons and eventually brain-sized populations of neurons. To this end, software tools exist aimed at simulating neuronal clusters ranging in size from a single neuron to networks matching a small animal's brain in size [4].

Simulating these neuronal networks on various platforms is an active field of research; a major challenge is the sheer computational complexity that many of these neuron models entail. Aiming at larger, more accurate neuron networks, neuroscientists require more memory and extended execution times to produce relevant results. Even the less complex models present significant challenges, in terms of computation, data-transfer rates and storage, that scale with the studied neuronal network size. Traditionally in the domain of neuroscience, the most common methods for simulating neuron models and studying their behavior were either through widely-known mathematical software suites such as MATLAB [5] or through special neuromodeling tools like NEURON [6], the NEuronal Simulation Tool (NEST) [7] and Brian [8]. While these tools have been used extensively to advance the field of computational neuroscience, simulating neuronal networks of realistic sizes in high detail remains a challenge; high-performance computing (HPC) has been recently recognized as a viable means for coping with this obstacle [9–14].

In this paper we develop a simulator for biophysically plausible neuron models, targeting a part of the human brain called the Inferior Olivary Nucleus, which specializes in the coordination and learning of motor function, among other crucial tasks [15]. The modeling accuracy is at the cell conductance level (as introduced by Hodgkin and Huxley models [16]), allowing us to expose fine details of the neuron mechanisms. The computational

* Corresponding author.
E-mail addresses: georgec@microlab.ntua.gr (G. Chatzikonstantis), harry.sidi@gmail.com (H. Sidiropoulos), c.strydis@erasmusmc.nl (C. Strydis), mnegrello@gmail.com (M. Negrello), g.smaragdos@erasmusmc.nl (G. Smaragdos), c.dezeeuw@erasmusmc.nl (C.I. De Zeeuw), dsoudris@microlab.ntua.gr (D.J. Soudris).

https://doi.org/10.1016/j.neucom.2018.10.062
0925-2312/

requirements of this biologically-accurate simulator make it an excellent candidate for parallelization on accelerator fabrics, such as the Intel Xeon Phi processors [17], due to the large inherent parallelism of the models. Additionally, it constitutes a realistic and challenging scenario in terms of model complexity, due to the model's mathematical stiffness and large number of floating-point operations required per simulation step, hence a benchmark for neuromodeling workloads. To tackle the computational complexity of the model, we utilize the Xeon Phi Knights Landing [18], an Intel processor with accelerator elements for massive computational power and parallelism potential. It supports traditional parallel-programming paradigms, such as MPI [19] and OpenMP [20], in contrast to Graphics Processing Units (GPU) requiring platform-specific programming paradigms [21]. The hardware assets present on the KNL are also found, in a smaller scale, on Intel Xeon processors commonly found in HPC centers and the two families of processors share binary compatibility. Thus, the simulator featured in this body of work is portable to other x86-based processors and conclusions derived can act as a general guideline for a wide class of similar computing platforms. In this sense, the Xeon Phi KNL is treated as an example platform.

The paper is organized as follows: in Section 2, we give an overview of the neuroscientific landscape and the research tools available today. In Section 3, we discuss the workings of our HH-based simulator, as well as the Phi processor architectural details. Following this information, we will delineate how the simulator is implemented on the KNL. More specifically, we will elaborate on how the simulator smoothly scales network computation across multiple KNL processors. In Section 4, we present the methodology of our experimentation and evaluate their results, followed by a discussion on system scalability. We explain how different network configurations affect the computational complexity of the simulation. In Section 5, we utilize the results of Section 4 to draw conclusions on the behavior exhibited by our developed system; we also discuss "lessons learnt" from this work that can be applied to other workloads of this neuromodeling class. Finally, Section 6 contains a concluding overview of the paper.

2. Related work

2.1. Domain overview

The different layers of abstraction in human brain studies are reflected in the usage of multiple models of neuronal functionality, with varying degrees of complexity and biochemical detail. Spiking Neural Networks (SNNs) [22] focus on input-current patterns and spike-transfer delays, attempting to replicate behaviors observed in their biological counterparts [23]. Some of the mechanisms exhibited by these models, such as the precise timing and frequency of their spiking patterns, are used by neuroscientists to study the human brain and verify (or reject) hypotheses that are much more difficult to recreate via *in vitro* and *in vivo* experiments. SNNs can be divided in two very broad categories known as Integrate and Fire (I&F) and as conductance-based models.

I&F models are the simplest SNN models, primarily focusing on receiving a spike input and determining the neuron's response based on a voltage threshold. They are widely used due to their simplicity and extensibility, resulting in a large range of I&F variants in the literature (e.g. leaky [24,25], adaptive exponential [26] and quadratic [27] I&F models). *Conductance-based models* lie on the opposite side of the spectrum, using complicated differential equations to represent the contribution of individual ion channels. They offer valuable insight into the electrochemical properties of the neuron and the machinations of its ion channels, but they come at the cost of significant computational complexity and difficulty in fine-tuning and studying. The Hodgkin and Huxley model [16], used in this work, can be considered as the most prominent example of this class; there is an extensive number of works in the neuroscientific literature that study the functionality and behavior of the Hodgkin and Huxley model [28–32]. A thorough classification of available neuron models and simulators has been made by Brette et al. [22].

Despite momentous achievements in the simulation of large scale neural systems, the path ahead is no less daunting. In the last decade, the computational neuroscience literature has seen the publication of brain scale models that include numbers of neurons comparable with those of biological systems, or patches of brain with high level of detail. Izhikevich and Edelman [33] simulated the whole thalamocortical system with quadratic 2D models and simple synapses, the Blue Brain Project has simulated detailed networks of a whole reconstructed cortical column with compartmental models and detailed synaptic models [34], as well as Erik De Schutter et al., who produced a highly detailed model of the cerebellar granular layer [35] . Going forward, it is the stated goal of the human brain project of expanding on the work of the Blue Brain Project and simulating a whole brain. Many other large scale reconstruction and analysis projects should be expected in the future, examining both larger neuron populations and more detailed neuron models [36].

The projects named above should be taken as isolated 'proofs of principle', and even if the authors have searched parameter spaces, the parameter space of possible networks has barely been scratched. The goal of computational neuroscience is not only to simulate a single column or even brain, but enormous classes of possible virtual brains. Making matters worse, it is likely that the future will demand that these brains be hooked to sensors and actuators and be required to function in real time and closed-loops.

This type of work pushes multiple boundaries of knowledge and technology. On the knowledge front, it commits the computational neuroscientist to a level of detail of the representation that exposes the free or unknown parameters of the system. This includes both the procurement of biological data, and the exploration of the gigantic parameter spaces. In fact, biological measurements of neuronal parameters can only take us so far, since neural network parameter spaces are far from convex [37,38], hence simply measuring biophysical properties of neurons will not be sufficient to recreate plausible neurodynamics. To make matters worse, biological neurons are in continuous change [39], and future brain models will need to tackle the problem of changeability as well, introducing yet another level of computational demands on the simulation.

A caveat of large scale simulations often put forth is that the correct level of detail for simulating brains is not known. This alone should be taken as justification for maintaining an agnostic view on the 'a priori' required level of detail of the simulation. It is not inconceivable that future models will continue to biological detail that is relevant in particular scientific domains, and hence this agnosticism is commendable. The best means to define that required level of detail is in the simulation of large scale systems and the comparison with reduced version, to gauge the contribution of the extra amounts of detail. The work of reducing a model to its essentials, often passes through understanding the implications of more complex assumptions, and hence, to simplify one often has to complexify.

Hence, we should predict that the computational requirements for future neurocomputational models will demand ever increasing computational resources, particularly in the problems of parameter space exploration, large network homeostasis and real time embedding of brain sized simulations.

Table 1
Prior art.

Reference	Platform	**Neuron model**	**Network size (neurons)**	Network density (connections /neuron)	Simulation speed (execution time/s of brain activity)
Hines et al. [40]	Supercomputer Intrepid BG/P	I&F	4 mil.	10,000	47 s
Kunkel et al. [41]	Supercomputer K	I&F	1.8 bil.	6000	270 h
Beyeler et al. [42]	GPU	Izhikevich	300,000	300	15 s
Hoang et al. 2013[43]	Multinode GPU	Izhikevich	1 mil.	100	1 s
Sripad et al. 2018 [44]	Multi-FPGA	Izhikevich	2,000	10	2 ms
Ananthanarayanan et al. [4]	Supercomputer Dawn BG/P	Izhikevich	900 mil.	10,000	300 s
Florimbi et al. [45]	GPU	HH-based	400,000	8	1.33 h
Nguyen et al. [46]	GPU	HH-based	1 mil.	8	400 s
Chatzikonstantis et al. [47]	Xeon CPU and Phi KNC	HH-based	1 mil.	100	24 min
Current work	**Multinode Phi KNL**	**HH-based**	**2 mil.**	**1000**	**10 min**

Table detailing the efforts in the literature to simulate neuronal models of varying complexity in high-performance computing platforms. The Table attempts to extract information from the cited studies concerning each work's best-effort network simulation and respective simulation speed. In order to evaluate performance, execution time per second of simulated network activity is reported. The studied neuron model, network configuration and hardware used are also reported.

2.2. Neuronal simulation projects

Accelerators and many-core fabrics are an attractive option for neuroscientific workloads. One of the most fundamental software tools in the domain of neuroscience has been NEURON [6], an all-purpose neural simulator encompassing most widely-used neuron models to date [48]. It is still widely used and a pillar for neuroscientific research [49].

There is a number of notable modern neuroscientific simulators under development that can boast wide usage. Brian is a python-based lightweight simulator which is favored for its ease of usage and simple coding structure, significantly lessening the user's programming burden when trying to design an experimental run [50,51]. NEST [7] is a simulation tool that is designed to run efficiently on a large scale of computing systems and aims at simulating large networks of simpler neuron models [52]. Furthermore, there is a large number of recent attempts at simulating and visualizing the mechanisms of a designed neuronal network, particularly as parts of Matlab [53] toolboxes [54–56]. In addition to traditional software methods, a different approach is explored by the European research project FACETS [57], where analog neuromorphic hardware directly simulates complex neuron models. Other toolkits, aimed at the development of neuronal models, have been ported to accelerators. An FPGA toolbox for simulating SNNs in hardware has been developed by Qingxiang et al. [58]. CARLsim [42], on the other hand, is a GPU-oriented library for SNN simulation and model-testing; it is also primarily geared towards robotic control.

The domain's literature details multiple projects that study simulations of neuronal networks by utilizing various HPC-related tools. Fidjeland et al. [59], Ahmadi and Soleimani [60] and Hoang et al. [43] have successfully deployed densely connected neuronal networks of Izhikevich [61] neuron models on GPUs. The same model has been studied by Bhuiyan et al. [62] who use various platforms to scale up to millions of neurons, coupled with few HH neurons in a 2-level neuronal structure for pattern recognition. SNAVA is a multi-FPGA effort to simulate flexible neuronal networks comprised of multiple neuron models [44]. One of the largest simulation efforts has been carried out by Ananthanarayanan et al., where neuronal networks reaching the size of a billion of Izhikevich-model neurons have been simulated on a supercomputer [4] using MPI libraries [19]. Hines et al. have also used a Blue Gene supercomputer to simulate millions of simple spiking neurons [40].

Our work differs from the domain's existing literature by focusing on complex conductance-based models, contrary to the trend of focusing on simpler models for large-scale simulations. Furthermore, its programming is based on the easily-accessible x86 architecture and refrains from using GPUs or FPGAs; traditional parallel programming tools are easier to deploy than GPU- or FPGA-specific methods. In addition, the complexity of the studied models demand the superior single-threaded performance that KNL processors offer compared to GPUs, as well as previous versions of the simulator ported on the first generation of Xeon Phi Knights Corner [47]. These factors contribute to the development of a portable, scalable, easily maintainable and extendable simulator that attempts to expose greater neuronal detail than the literature's norm. Finally, in this work, we focus on the performance scaling behaviors that can be observed for workloads of this class and derive conclusions that can be beneficial to other projects facing similar neuromodeling challenges.

This section does not include a full review of all related work in the literature; an extensive survey is outside the scope of this paper. However, Table 1 attempts to relay an overview of the landscape to the reader by gathering information from the aforementioned works and reporting their achieved results. The Table also includes our previous and current work.

3. System description

The simulator described in this paper is written in C language and optimized for many-core x86 systems. Specifically, the simulator has been developed for Intel Xeon Phi line of accelerators. In addition, the micro-optimizations used to boost simulator performance are also beneficial to other x86 systems. "KNL-exclusive" hardware assets are configured in a fashion that can be encountered in other processors as well; for example, we use a special low-latency on chip memory called "MCDRAM" as shared last-level-cache, which is a commonly found in Intel Xeon processors. As such, we refrain from limiting our conclusions to the KNL family of processors and ensure effective portability to other platforms.

3.1. The Inferior Olive (InfOli) model

The model studied in the current paper is a conductance-based model of the Hodgkin–Huxley neuron, modified in order to better simulate the communication mechanisms of the human inferior olivary nuclei [64,65]. As mentioned in Section 2.1, conductance-based models are more complex models that feature biophysically accurate descriptions of the inner workings of each neuron, based on its biochemical properties. They tend to be very demanding in computational resources for simulation. For reference, a lighter version of the HH-based model discussed in this paper, featuring

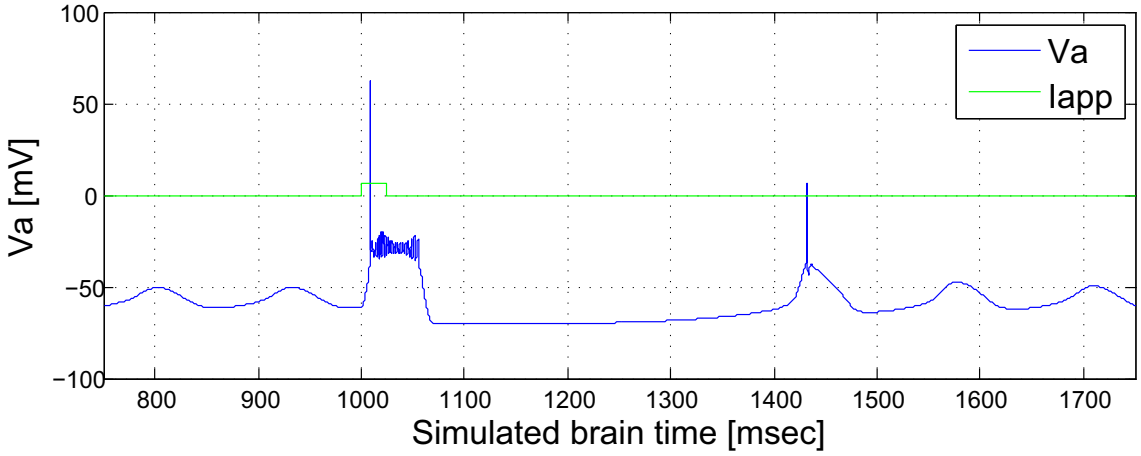


Fig. 1. Sample spike generation of the inferior olivary nucleus model studied in the present paper [63]. An externally applied electrical pulse (*Iapp*, denoted by the light-green line in the Figure) stimulates the neuron at $t = 1000$ ms. The axonal membrane voltage levels (*Va*) are recorded; the reader may observe the generation of an action potential, followed by a refractory period and a return to the neuron's normal oscillation.(For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

significantly less complicated inter-neuron communication mechanisms, has been ported on NVIDIA GPUs and scaled up to one million neurons [46].

The inferior olivary region is a small part of the brain linked to the cerebellum and is theorized to play a crucial role in the learning of movements and proper motor function. The model approximates the biological neuron with three different compartments: dendrite, soma and axon.

The dendritic compartment features a set of ordinary differential equations (ODEs) that simulate current exchange with other neurons of the inferior olivary network. This exchange happens between dendrites that have formed Gap Junctions (GJ), i.e. the electrotonic connections or synapses among them. Each dendritic compartment forms multiple such electrical synapses, allowing inter-neuron communication and introducing, for denser networks, a major source of computational complexity and multiprocessing synchronization overhead.

Most of the neuron ion channels are realized in its somatic compartment. These channels are crucial to evaluating the neuron's state in each simulation step. In sparser networks, the floating-point operations demanded by each somatic compartment dictates the majority of the simulation's computational workload. Finally, the axonal compartment is the part of the inferior olivary model that functions as the neuron's output stage towards other parts of the brain, such as the climbing fibers. It features less floating-point operations than the soma and its simulation is less complex than the other compartments of the neuron.

3.2. Xeon Phi Knights Landing

Intel Xeon Phi Knights Landing is a series of processors featuring an x86-based many-core architecture that specializes in servicing demanding HPC applications. The specific Knights Landing processor model examined in the present paper features 64 cores. Each core utilizes two 512-bit-wide vectorization processing units (VPUs) which enable AVX-512 instructions for parallel data processing. Furthermore, best practices indicate that each Knights Landing core can support the execution of up to four software threads in parallel [67]. These elements combined hint on the massive potential parallelism present on the processor. As such, codebases operate best on Knights Landing processors if they feature high degrees of parallelism, vectorization and ideally, well-designed accesses to memory.

The cores of the KNL processor each have access to a private 32KB L1 cache and pairs of cores have a 1MB L2 cache shared between the two cores. Via the L2 caches, the tiles are connected to each other in a mesh fashion. There are options available to the KNL user concerning the mode of operation followed by the processor's cache hierarchy. These options are referred to as "cache clustering modes", are configured at boot time and determine how the memory address space is distributed across the chip. The KNL features four modes: all-to-all, hemisphere/quadrant and sub-NUMA cluster modes of cache operation.

In all-to-all clustering mode, memory addresses are uniformly distributed across all of the tiles' tag directories. In hemisphere clustering mode, the 36 tiles of the KNL are divided into two spacial halves called hemispheres, ensuring that messages can be constrained within the hemisphere. The quadrant clustering mode follows the same mentality as the hemisphere but partitions the die's tiles in four spacial parts instead of two. Finally, the sub-NUMA cluster (SNC) modes are Non-Uniform Memory Access extensions of the hemisphere and quadrant cache operation modes; they are divided in SNC-2 and SNC-4, respectively. In our research, symmetrical networks act as well-balanced workloads evenly distributed throughout the KNL's cores. As such, we treat the KNL processor as a symmetrically-distributed multiprocessor and opt for quadrant mode of cache operation.

Another feature of the KNL processor aimed at reducing memory-access latency is the 16GB multi-channel dynamic random access memory (MCDRAM). This is an on-package high-bandwidth memory spacially located next to the processing cores that can deliver significantly higher (more than 400 GB/s) bandwidth than the chip's 384 GB DDR4 RAM (approximately 90 GB/s bandwidth). It also comes with three modes of operation chosen at boot time. When the MCDRAM operates in "flat" mode, it serves as a high-speed extension of the DDR4 memory. Alternatively, it can be configured to serve in "cache" mode, where it is treated as a last-level cache (LLC). Finally, it can be set up in "hybrid" mode where a pre-determined part of the memory is used in flat mode, while the remaining MCDRAM serves as an LLC. We utilize the MCDRAM entirely in cache mode, since some of the larger neuronal networks explored in this paper cannot be allocated on 16GBs of "flat" MCDRAM; additionally, "cache" mode is the most generalizable configuration for any other type of model we choose to port to the KNL and it bears resemblance to shared LLCs present in Xeon processors.

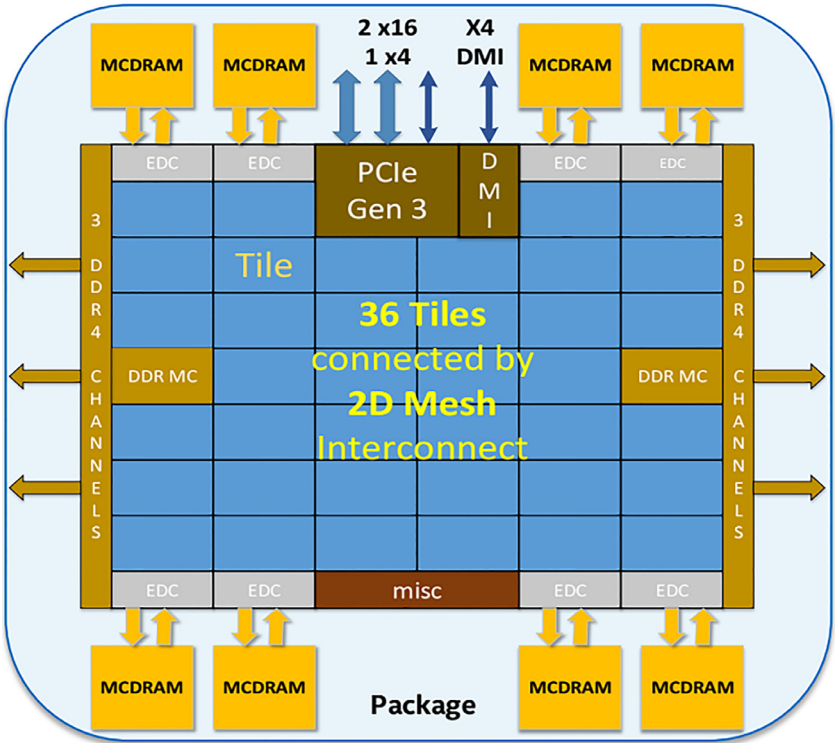


Fig. 2. The Knights Landing die organization [17]. Each tile consists of 2 cores that share an L2 cache. Communication between cores is orchestrated as a mesh, in contrast to the previous generation (Knights Corner) which employed bidirectional rings [66].

3.3. Multinode implementation

The simulator described in this paper has been previously ported on an Intel Xeon Phi 1st generation coprocessor (Knights Corner - KNC [66]) [9]. An advantage of using Intel Xeon Phi Knights Landing (KNL [17]) 2nd generation processors as a high-performance computing fabric over its predecessor, the KNC, as well as other accelerators, is the ease of employing a multinode implementation. The Xeon Phi line of products supports traditional parallel coding paradigms, such as MPI and OpenMP for task-level parallelism. These tools have been well-studied and are constantly improved upon, significantly reducing the difficulty and time-to-market of a scalable, highly-parallel implementation of the simulator's algorithm.

3.3.1. Resource partitioning

As mentioned in Section 3.2, the KNL processors utilized in this paper feature 64 cores, each able to dispatch up to 4 instruction streams simultaneously [17]. Thus, when employing n KNL processors, there is a degree of thread-level parallelism equal to $n \times 64 \times 4 = n \times 256$. In addition, it should be noted that each thread utilizes, when applicable, the AVX-512 instruction set which allows for vectorized instructions to operate on multiple data simultaneously.

In the case of our simulator, the thread-level computational capabilities of the ensemble of KNL processors are divided in groups and assigned to different MPI ranks [19]. An illustration of the method with which the computational resources of the KNL are partitioned across the studied neuronal network can be found in Fig. 3. Out of the possible MPI-to-OMP ratio configurations, we opt for 4 MPI ranks spawning 64 OpenMP threads. Fig. 4 shows that other configurations may have a small advantage performance-wise for one network setup, but exhibit significantly worse simulation speed in the case of different networks. A 4:64 MPI:OMP ratio works reliably well across multiple network configurations. This middle-of-the-road approach to the ratio of MPI ranks to OpenMP threads coincides with previous decisions on the 1st generation Xeon Phi KNC [9,47].

3.3.2. Algorithmic overview

On an algorithmic level, OpenMP threads operate on different parts of the neuronal network. Each neuron in the network is assigned to a single thread in order to be processed. Each thread handles an equal number of neurons, in order for the computational workload to remain balanced. Each neuron in the network is connected to others (except for special cases of zero connectivity) via the modeled Gap Junctions. This mechanism necessitates the usage of MPI collective communication in order to exchange data between processors that do not share memory. The amount of communication traffic between MPI ranks, whether on the same or on different KNL processors, depends on the amount of neurons in the network and the network's density, which indicates the average number of GJs each neuron has established.

During the simulation of any given neuronal network, each MPI rank is responsible for the message-passing needs of its assigned sub-network, which is processed in parallel by 64 threads. This procedure can be divided in two sub-processes: sending and receiving MPI messages. In each simulation step, Gap Junctions need the dendritic membrane voltage levels of the participating neurons in the connection in order to be computed. The MPI rank satisfies the other ranks' needs by packing the necessary values in a buffer after OpenMP thread calculations. The buffer is then distributed by using MPI's broadcast function (**MPI_Bcast**). The upper limit for this data-exchange instance happens when each MPI rank needs to broadcast voltage values for each neuron they handle.

After the MPI rank completes its **MPI_Bcast** function, it receives the other MPI ranks' broadcasts. The contents of each received buffer are processed by spawning 64 OpenMP threads which op-

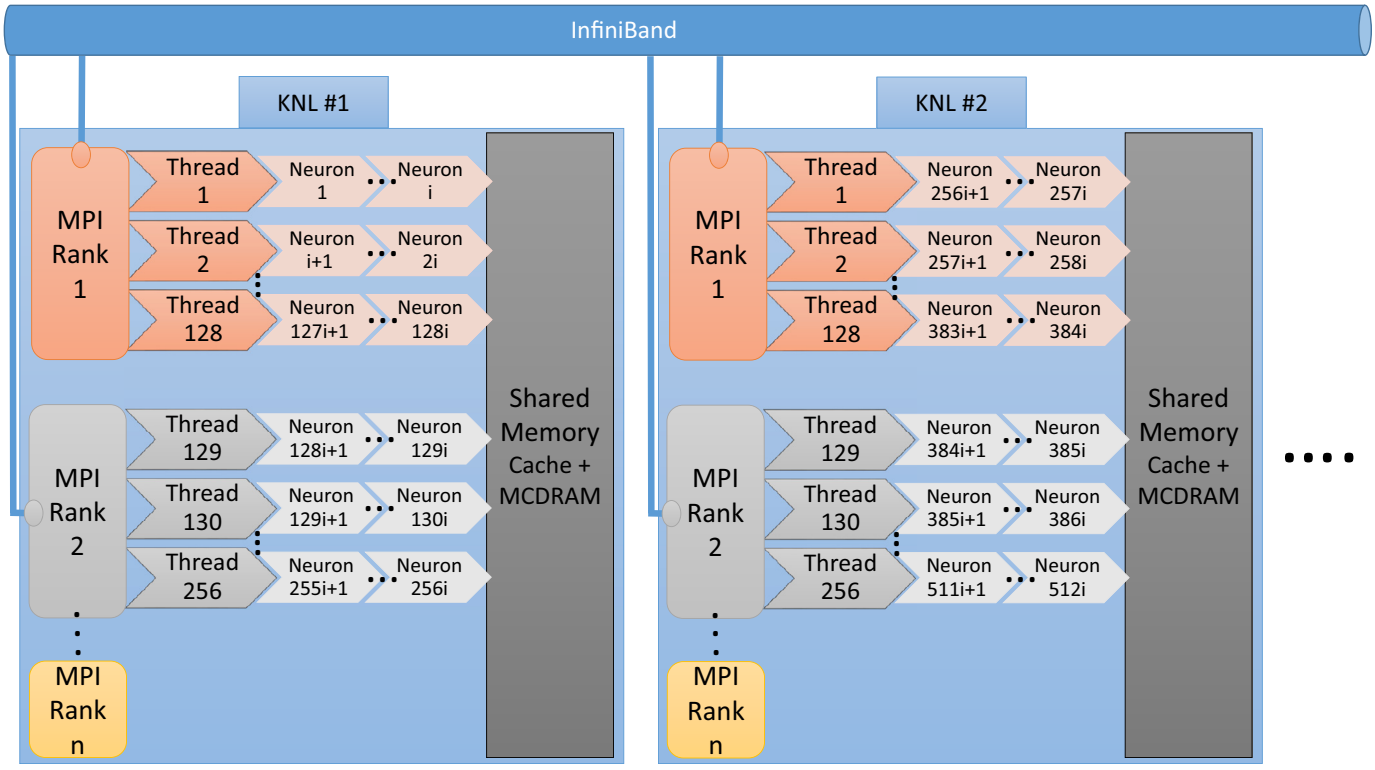


Fig. 3. Schematic of the simulator multinode implementation on the Knights Landing. In this example, each KNL processor operates at maximum capacity, meaning all of its $64 \times 4 = 256$ threads are employed, while a variable n amount of MPI Ranks are spawned per platform. It should be noted that in our work, we opted for spawning $n = 4$ MPI Ranks per KNL platform. A number of i neurons is assigned to each thread in this simulation, totalling a simulated network of $l = i \times 512$ neurons over two KNLs. The implementation schema can be extended to include as many KNL machines as necessary and available.

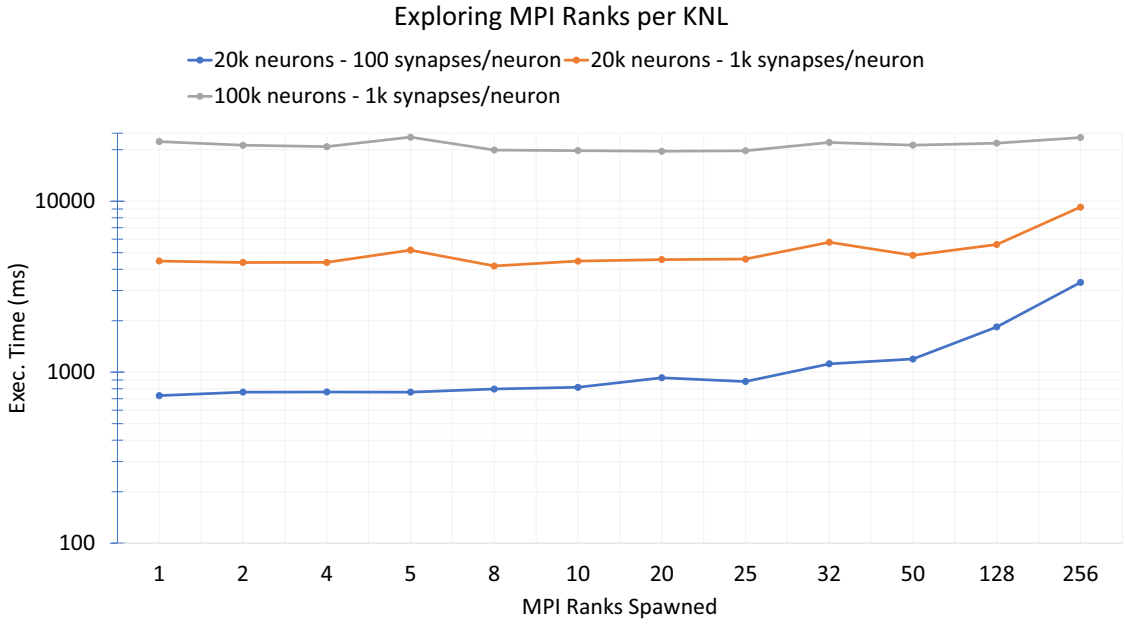


Fig. 4. Exploration of KNL's performance under different configurations of hybrid MPI-OpenMP clustering granularity. Three different networks of varying degrees in neuron population size and density are examined for 100 ms of simulated brain time. We alter the amount of MPI ranks spawned on a single KNL processor. Configurations employing a small amount of MPI ranks exhibit superior performance. In particular, using 4 MPI ranks spawning 64 OpenMP threads offers good, reliable performance for all neuronal networks.

erate on the buffer in parallel. In the worst-case scenario of 100% connectivity density, each of the 64 threads needs access to the full content of the received buffers; in this case, each rank gets updated on the entirety of the rest of the network in every simulation step. Following the processing of the received data buffers, the calculation of Gap Junctions, as well as the neuron compartmental states, can be carried out by the threads. Upon completion of these calculations, the OpenMP threads are joined, thus ensuring that the network state update is complete and ready to be processed in the next simulation step, which begins with a new **MPI_Bcast** function from the MPI rank.

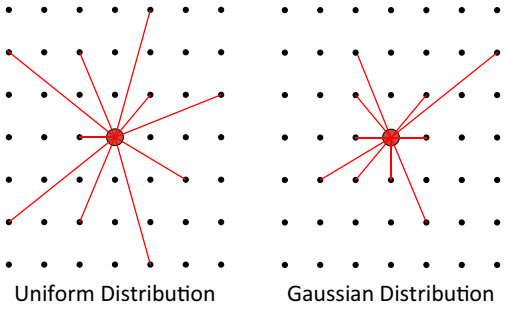


Fig. 5. Depiction of two different 7x7 2D neuron-meshes. In each case, the neuron in the center of the mesh forms 10 connections; the leftmost mesh follows a uniform distribution, whereas the rightmost features a Gaussian distribution. Uniform distribution creates spread-out connections, whereas the Gaussian distribution keeps the connections closer to their point of origin (i.e. neuron in the center of the mesh).

Table 2
Parameter space.

Variable name	Value range
Network size	1000–2,000,000 nrns
Network density	0–1000 syn/nrn
Synaptic pattern	Uniform and Gaussian
KNL nodes used	1–8 nodes

Range of explored parameters in this paper. The Table details network configurations considered, as well as the amount of hardware used during simulation.

4. Experimental evaluation

In order to evaluate the performance of the proposed simulator, we have run a number of experiments in neuronal networks of widely ranging connectivity patterns, density and size.

4.1. Experimental setup

We organize neurons in a 3-dimensional grid. For exploring the impact of network topology, we explore two different (and naturally occurring) distributions: a uniform distribution of synapses in the network; and a Gaussian distribution of synapses where neurons in proximate positions on the 3D grid are significantly more likely to form a bond. The differences of these distributions are visualized (in a 2D grid, for ease of reference) in Fig. 5.

These distributions represent different patterns of connectivity in the biological brain; neurons may exhibit local synaptic connectivity, as in the case of neocortical pyramidal neurons [68], while long-range synaptic patterns can also play an important role in neuron functionality [69]. Moreover, by exploring different connectivity patterns, it will be evident that synapse distribution affects performance in a definitive manner.

Networks are tested on varying degrees of size, as summarized in Table 2. The smallest networks evaluated are formed of 1000 neuron populations, whereas the largest are comprised of 2 million neurons. For exploring the impact of connectivity density, various (fixed) amounts of synapses per neuron have been used; configurations of no-connectivity, 10, 100 and 1000 synapses per neuron are tested. These particular configuration points match (and surpass) connectivity as encountered in biological inferior olivary nucleus and aim at revealing the simulator performance trends under increasing network density.

The measurements utilize the standard **gettimeofday()** C-function in order to evaluate execution time for the simulation of the network after it has been set up. In these measurements, input and output have been restricted to a minimum in order to measure pure simulation execution time. The experiments simulate 100 ms of brain time. Since this is a time-driven simulator with a steady, incompressible time-step of 50 μ s, brain activity during simulated brain time is not relevant to the simulator's performance, in contrast to event-driven simulators, whose performance is affected by neuronal spike generation frequency.

In addition to differing network sizes, connectivity policies as well as densities, we perform scalability experiments by employing multiple KNL nodes (1, 2, 4 and 8), with hardware assets as described in Section 3.2 and configured as in Section 3.3. A detailed discussion on the scaling behavior of the multi-KNL implementation is thus, also included in this evaluation.

4.2. Performance considerations

Multinode manycore systems are complicated. Analysis and pattern-detection for a heavy data-exchanging application, such as a Hodgkin–Huxley-model-based neuron simulator, is a challenge on such a system. We will first discuss impact factors that heavily influence the simulator's performance under different workloads and configurations. We will, then, discuss our experimental results with these factors in mind.

4.2.1. Manycore resource utilization

There is a price all manycore systems pay for utilizing their resources in parallel. Spawning and joining software threads and/or tasks, via the usage of libraries such as the OpenMP, requires an amount of preparation and core-time that constitutes a non-negligible overhead. In addition, unless examining an embarrassingly parallel application, parallelization resources of the manycore platform require synchronization at certain "checkpoints" in the algorithm. Simulations of biological neuronal networks entail exchange of bio-signals, which invariably result in some way of thread communication when using a manycore system with a shared memory hierarchy. Increasing the detail and complexity of the model scales the amount of such bio-signals the application simulates; as such, the biophysically realistic model studied in this paper is highly demanding in synchronization when employing a complicated, dense neuronal network.

In addition, contemporary manycore processors feature a wealth of parallelization resources for threading and vectorizing code. The KNL, for instance, by utilizing AVX-512 instructions by all of its available threads, can potentially execute more than 10,000 floating-point operations in parallel. This parallelization potential requires a suitable workload in order to be properly utilized. Since the simulator's unit of operation is the single neuron, a network's population size is bound by a lower limit; simulations under this size limit cannot be expected to utilize all of the manycore's assets, especially when investigating multinode systems.

As a result, under-utilization of the platform's resources can severely hinder the platform's performance during a biophysically-complex simulation. The manycore processor's parallelization assets go under-used, while still causing overheads of spawning/joining tasks. Even if the simulation is large enough to feature high degrees of asset utilization, stiff models, such as the one examined in this paper, enforce data synchronization between threads in every simulation step, further reducing the efficiency with which the processor's hardware is employed. In conclusion, in order to attain acceptable efficiency when using manycore processors such as the KNL, each of its threads need to be assigned with the computation of a suitably large workload.

4.2.2. Message exchange overheads

MPI-like communication between the nodes in a multinode system is materialized through Infiniband. This type of communication poses a significantly heavier overhead than intranode synchronization processes do. As such, locality of data exchange between

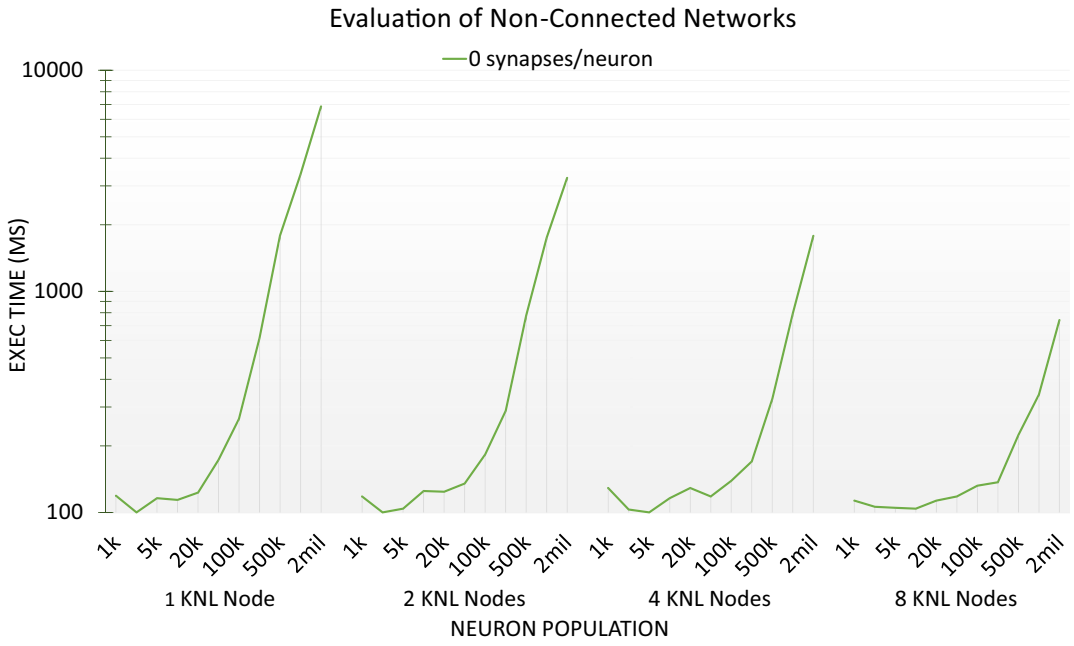


Fig. 6. Special use case of the simulator operating on non-connected networks. The neurons oscillate in a solitary environment. Due to the absence of communication between the cores' assigned subnetworks, this use case can be considered as one of the best cases for parallel processing from a scaling perspective. Utilizing increasing amounts of hardware scales simulation speed in an efficient manner; network simulation for 2 million isolated neurons requires execution time that is within the same order of magnitude as real time.

neurons in the simulator is particularly important. Real neurons in the brain exchange current (data) by being physically approximate to each other; this translates well for locality in the hardware. By partitioning the network in clusters of neurons which are physically close to one another, most messages between those neurons stay intra-node, avoiding using MPI functions to other cores or processors.

As a result, simulations that do not allow for an efficient partitioning of the network in local sub-clusters will exhibit significantly less scaling potential. When examining different distributions for the network's connectivity map, it becomes evident that the overhead of inter-node communication is a limiting factor for utilizing multiple processors if connections are spread out throughout the network. These types of networks can be hard to partition in an effective manner.

4.3. Evaluation results

In this section we will present the results of the experiments carried out for this paper and assess the simulator's performance. We will analyze the behaviors exhibited in each case by referring to the factors impacting the manycore processors' performance, as mentioned in Section 4.2.

4.3.1. Non-connected networks

Fig. 6 depicts the special case of networks without the forming of GJ connections. In these cases, neurons operate in isolation to each other in the network. The absence of GJs relaxes communication needs as it translates to a lack of need for synchronization between OpenMP threads and communication between MPI ranks. Furthermore, the special conditions for these types of simulations permits the KNL to utilize its low-latency memory assets without overheads from the MESIF cache coherency protocol. Finally, there is also a considerable reduction in computational needs since the processor skips the calculation of the GJs in each simulation step, which would otherwise take up a major portion of CPU time.

These factors combined lead to overall low execution times which differ from real time by less than two orders of magnitude even for populations of 2 million neurons. Each performance curve in Fig. 6 exhibits similar trends. The initial part of the curve, which corresponds to low-population networks, is flat, since these simulations are "low-effort" and under-utilize the hardware's assets. This trend extends to higher-population networks as more KNL processors are added to the simulator. On the other hand, when simulating larger neuronal networks, there is a linear increase in simulation speed as the number of KNL processors used grows. These observations are consistent with how an application with minimal communicational needs should behave.

4.3.2. Uniformly distributed gap junctions

The uniform distribution of connections in the network, depicted in Fig. 7, is the worst-case scenario for the simulator. In this method of distributing each neuron's connections over the network, every neuron pair has a uniformly equal chance of being created. When examining a network of n neurons, each forming g connections, a neuron pair, regardless of its location in the network, has a probability $p = g/n$ of being formed. Furthermore, if the network is simulated by c cores, then each core is tasked with simulating n/c neurons and $g \times (n/c)$ GJs. Due to the uniform distribution of these Gap Junctions, the core stores data locally concerning only n/c neurons, thus lacking data necessary for the computation of $(g-1) \times (n/c)$ GJs. This scenario causes the simulation to be very "heavy" on utilizing MPI collective functions for data exchange. Memory accesses degrade the simulator's performance further, since L1 and L2 caches are unlikely to hold necessary data, forcing processor's cores to search in non-local caches.

This information explains the unsatisfactory performance exhibited by the simulator in Fig. 7. The application scales poorly, particularly when utilizing 8 processor nodes. Due to the system's lack of scalability, measurements of only up to 1 million neurons are depicted. Larger network populations cannot be simulated effectively, regardless of the amount of hardware utilized. The performance curves of 8 KNL nodes in Fig. 7 demonstrates that for larger net-

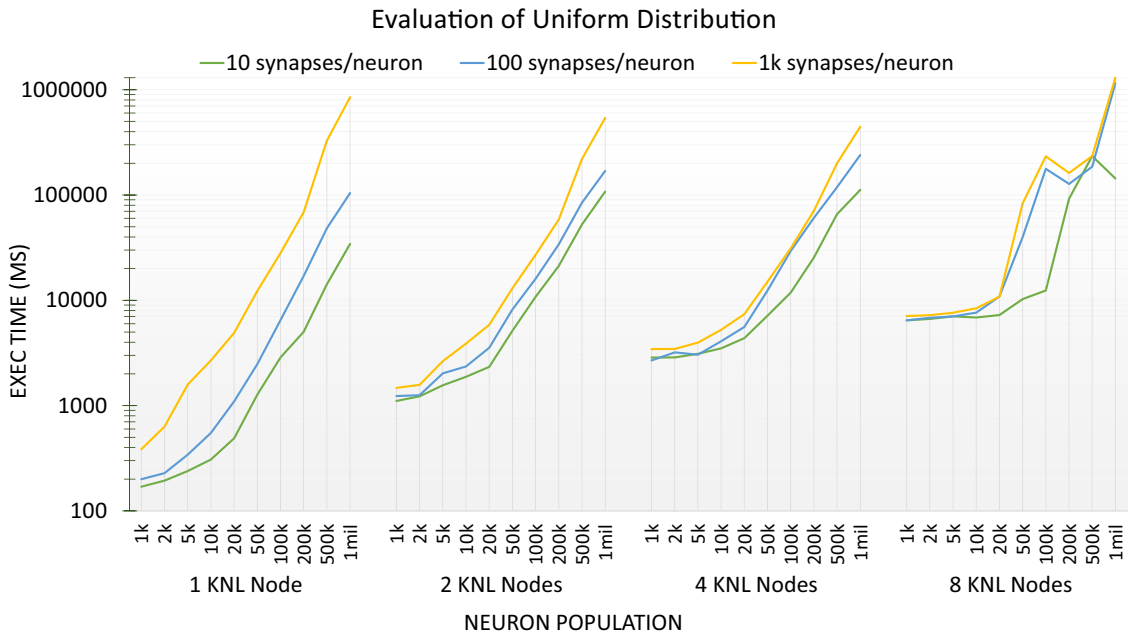


Fig. 7. Study of simulated networks following a uniform distribution of Gap Junction bonds. In this scenario, neurons show no preference over which neuron they form a bond with, resulting in GJ bonds being uniformly distributed across the entirety of the network. In this fashion, the application's performance and scalability is hindered due to data messages being exchanged between cores, especially between those belonging to different KNL machines. As such, only a small degree of speedup is attained by employing two KNL nodes instead of one. Furthermore, no further gains are observed when scaling to more hardware, particularly for heavier workloads.

works, execution times show a sharp increase and 8 KNLs perform worse than a single-node system, rendering the option of adding further hardware to the system ineffective.

The application's performance curves are significantly erratic and hard to interpret in this distribution case. A critical factor that determines simulation speed is the overhead of MPI collectives imposed during inter-node communication, as mentioned before in Section 4.2.2; this factor grows more dominant as the amount of machines employed during a simulation run increases. For any experiment consisting of a network of l GJs run on k different KNL machines, each processor needs to simulate the functionality of l/k GJ. The processor holds data capable of completing the calculation of a GJ without inter-node communication for l/k^2 GJs. Thus, the ratio of "expensive" inter-node communication versus "cheaper" intra-node data exchange directly correlates to the amount k of processors used in the case of uniform distributions of neuron connections.

In addition, Fig. 7 shows a qualitative difference between the performance curves of dense networks with 1000 GJs per neuron versus sparser networks. Dense networks, which exhibit a naturally heavier workload than sparser networks, depict a better tendency to benefit from using 2 KNL nodes over opting for single-node implementation. There is a small, but noticeable speedup for million-neuron dense networks, which is absent for similar in size, but sparser in connectivity populations.

This behavior can be attributed to the fact that in our simulator, data exchange between MPI ranks takes place with collective communication functions. MPI ranks exchange bundles with relevant dendritic voltage data concerning their respective subnetworks. In each simulation step, the MPI rank "builds" the bundle with data from neurons in its assigned subnetwork. A neuron in said subnetwork will be added to the bundle as long as there is a *single* GJ calculated by another MPI rank which needs this datum. Thus, in the case of uniform distribution, the probability of a neuron being added to the bundle grows quickly with the average amount of GJs formed by each neuron and "caps off" to 100% even for sparsely connected networks. When this probability reaches 100%, each MPI rank exchanges all of its subnetwork's dendritic data in each simulation step. In these cases, the maximum amount of data exchange between MPI ranks is achieved and, as explained, these cases are present even for networks of sparser density.

In conclusion, both sparse and dense networks must circulate large amounts of GJ-related data through the KNL's communication channels, both intra- and inter-node. However, denser networks have significantly more operations to perform in order to calculate GJ states, after acquiring all of the necessary data. These calculations happen in parallel, thus benefiting from employing more hardware and ultimately favour 2-KNL implementations over single-node. This benefit is "hidden" when employing more than 2 nodes due to "heavier" penalties to performance from communication-related overheads. It should be noted, however, that the performance curves of both sparse and dense networks follow the same trends when moving to 4-KNL simulations and that sparser networks show worse degradations in performance than the heavier experiments.

4.3.3. Gaussian distributions of gap junctions

The most realistic case of network connectivity, based on how real neurons in the inferior olivary region band together to form Gap Junction connections, is evaluated in Fig. 8, where neuronal proximity plays an important role in synapse forming according to a Gaussian distribution. A quick observation of the logarithmic Y-axis in the figure reveals that this scenario displays a decrease in overall execution times by nearly an order of magnitude when compared to the worst-case scenario of uniform distribution in Fig. 7.

In this use case, a satisfactory amount of locality in message exchange is achieved by clustering neurons according to their coordinates in the 3D-mesh. Neurons within a small range of Cartesian distance are assigned to the same core. According to the Gaussian distribution, this allows the core to calculate most of its GJs without referring to external data, since most (but not all) of its neuron connections link to other neurons handled locally by the same core. Hence, we limit the amount of messages exchanged between

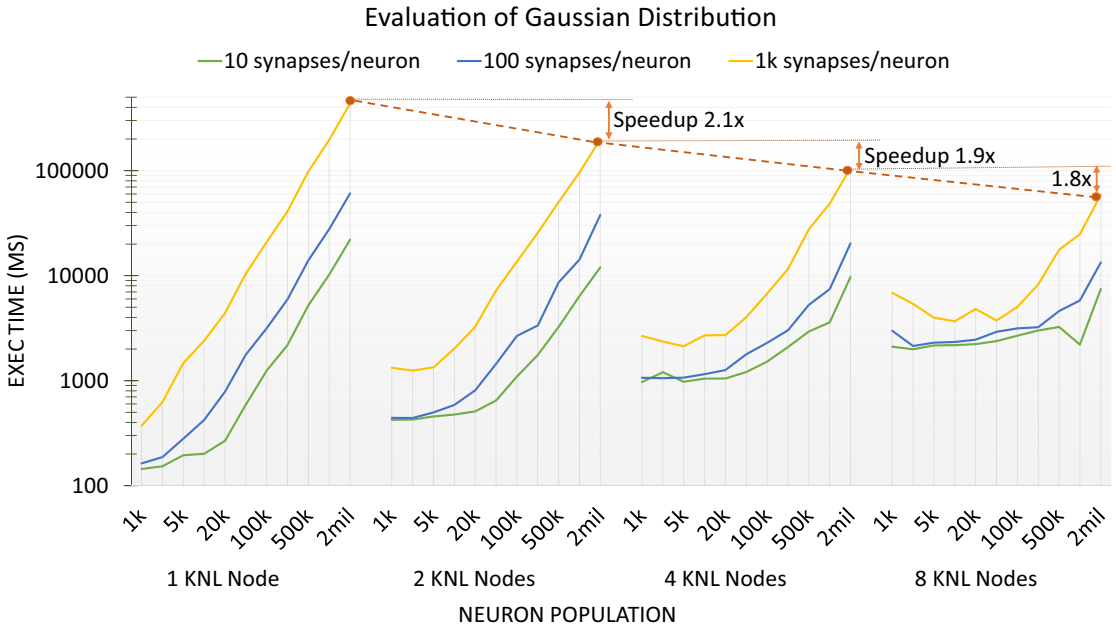


Fig. 8. Evaluation of the simulator's performance when computing networks of varying size and density. The network's connectivity map follows a Gaussian distribution. Neurons are imagined in a 3D space and form Gap Junction bonds between them. The likelihood of a neuronal pair forming is based on their proximity in the 3D space. The simulator's performance is evaluated when utilizing 1, 2, 4 and 8 KNL machines. Scalability is boosted by a significant factor due to the greater data locality. For large enough workloads, simulation speed increases in an almost linear fashion with the amount of hardware employed. Smaller speedups are attained for sparser, smaller networks.

cores intra- and inter-node, as well as reduce memory access latency by maximizing local cache usage.

Due to the favorable distribution, utilizing a multinode implementation yields positive results. There is a considerable speedup by adding more Knights Landing processors to the larger simulations. High efficiency is maintained for workloads that approach the 100k neuron-population mark in the case of dense network with 1k synapses per neuron. On the other hand, smaller networks do not exhibit favorable results when moving from single-node to multinode implementations. More specifically, Fig. 8 shows that there is a clear slowdown when employing 8 KNL nodes for relatively small networks of 5k neurons or less, when compared to the single-node's performance curve. Furthermore, an 8-KNL implementation for small and dense networks shows an improvement in execution speed when increasing the neuronal network size from 1k to 10k neurons.

These findings can be attributed to the factors mentioned in Section 4.2. When using a group of 8 manycore processors and spawning a large number of threads per processor, each capable of executing vectorization instructions, underutilization of the hardware assets causes considerable overheads that deteriorate performance based on how underutilized the processors are. This causes the simulator to execute *larger* neuronal networks *faster*, up to the point where the hardware's assets are utilized efficiently. The point at which the system's resources are *saturated* depend on the amount of processors used, as well as the network's density. Denser networks show a clearer, more impactful saturation point, as shown by comparing the performance curves of 100 versus 1k synapses per neuron. Furthermore, saturation is reached earlier when employing less manycore nodes due to less available resources to the system. When examining the performance curves of the densest network configurations in Fig. 8 (as noted with a golden yellow line), 2 KNL nodes retain stable execution times until the 5k neurons mark, whereas 8 KNL nodes show a true increase in execution times only past the 50k neurons mark.

Another point of interest is a super-linear speedup when moving from a single-node system to a 2-KNL configuration for 2 million neurons and 2 billion synapses. This behavior can be attributed to an increase of available low-latency assets. When using additional nodes of computational fabric, in addition to enhancing the system's potential parallel processing power, its total cache space (as well as the KNL's MCDRAM in our particular setup) is also expanded. By allowing a larger, if not whole, part of the network to be allocated in low-latency memory space, super-linear speedup can be observed in manycore multinode systems.

The multinode implementation allows the simulation of up to 2 million Hodgkin–Huxley-based neurons and 2 billion Gap Junctions for 100 ms within two minutes. As such, even in the case of the heaviest workload tested in this paper, the simulator exhibits a simulation speed that differs from real time by two to three orders of magnitude. In addition, networks of 5k neurons and 500k Gap Junctions, which represent sizable experiments in neuroscientific research, can be simulated in a single node at a rate that approaches 30–50% of a real brain's operational speed. Thus, the simulator can calculate workloads both light and heavy at satisfactory speeds; the single-node approach is recommended for smaller workloads, while multinode implementations are preferred for demanding networks.

5. Discussion

5.1. Optimal allocation of resources

One of the focal points in this paper is the concept of matching hardware utilization to the workload that requires calculating. The suggested amount of hardware to deploy for each network simulation varies according to network size and its corresponding connectivity map. By using data collected from the experiments presented in Section 4.1, with parameter ranges described in Table 2, Fig. 9 depicts a general guideline for allocating the minimal KNL instances necessary for achieving the best possible performance for workload instance. The Figure exhibits a number of interesting patterns.

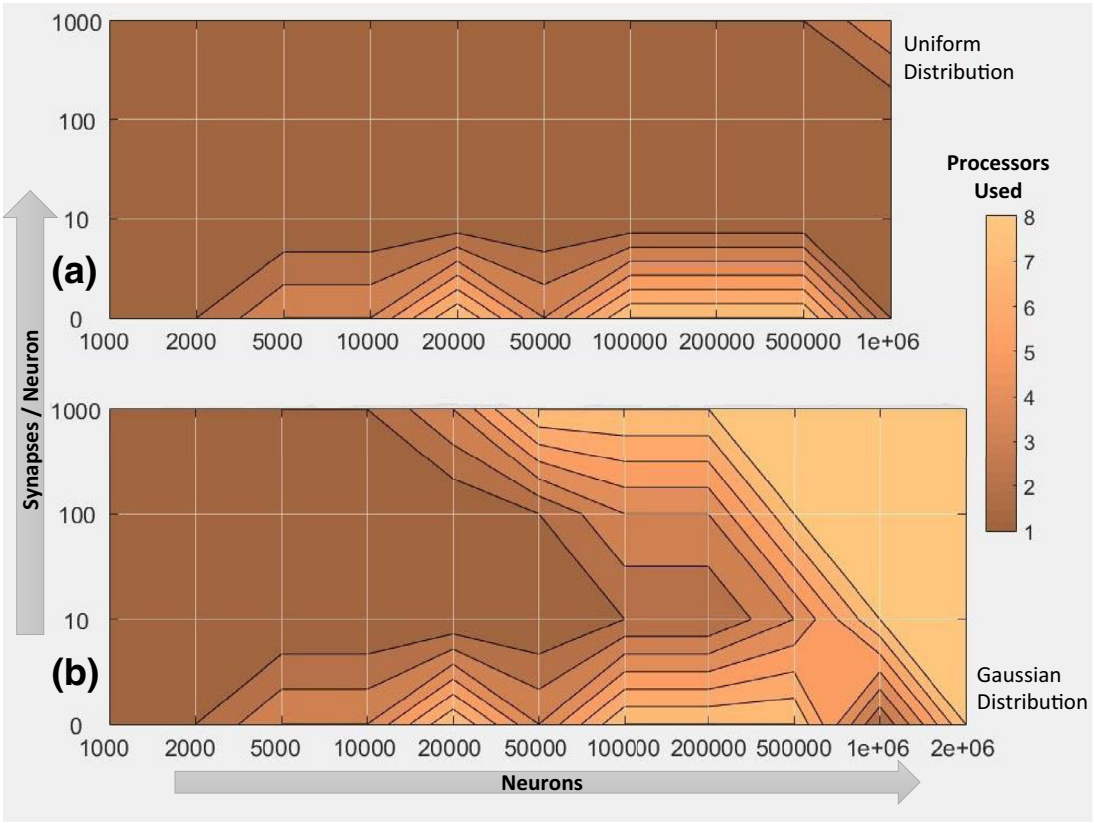


Fig. 9. Footprint of suggested implementation for simulated networks of varying sizes and connectivities. The colourmaps depict the amount of processors providing the best simulation speed for networks of uniform (panel **a**) and Gaussian (panel **b**) synaptic patterns. As derived from Section 4.3, single-node implementations dominate networks with uniform synaptic distributions due to poor scalability. In contrast, the case of Gaussian synaptic distributions varies with network density: highly dense networks require the maximal number of KNL nodes, whereas lower densities can be tackled with less nodes.

Fig. 9**a** depicts suggestions for networks with **uniformly distributed** connection patterns. We observe that:

- Uniformly-distributed connectivity maps force the simulator to become completely communication-bound, due to model complexity.

These types of network benefit mildly from 2-node implementations, while employing more hardware often yields no improvement. Fig. 9 shows that only networks with populations larger than 500,000 neurons and connectivity patterns denser than 100 synapses per neuron (thus, totalling more than 50 million synapses in the network) have an optimal configuration point of 2 to 4 processor nodes. In other cases, single-node implementations are recommended due to poor scalability.

Fig. 9**b** maps networks with connection patterns following a **Gaussian distribution**. We come to the following general conclusions:

- The notion that neighboring neurons are more likely to form bonds leads to significantly more scalable network configurations.

Connectivity maps based on the Gaussian distribution expose data locality better and support utilizing multiple KNL nodes. Fig. 9 shows multiple network configuration points where the maximal tested amount of processors is optimal for simulation speed. In this paper, up to 8 KNL processor nodes have been employed due to availability (as noted in Table 2); a larger amount of processors may yield further boosts to simulation speed.

Furthermore, network density directly correlates to the prevalence of cases where multinode allocations are recommended. For example, in Fig. 9b, networks featuring 10 synapses per neuron are suited for single-node solutions when population count is less than 100,000 neurons. On the other hand, when network density approaches 1000 synapes per neuron, network sizes of more than 10,000 neurons merit multinode configurations. This behavior can be attributed to the fact that network density affects the amount of floating-point instructions issued per simulation step; by increasing network density, the computational workload becomes heavier and thus, can be calculated more effectively by employing larger amounts of computational resources.

Both panels in Fig. 9 show that:

- When network sizes are large while network synaptic count is low, the neuromodeling problem becomes an embarrassingly parallel use case and utilizing a high amount of processors is recommended.
- Small, dense networks benefit from single-node allocations, otherwise computational resources are effectively wasted and simulator performance suffers.

Networks featuring low synaptic connectivity maps behave in a similar fashion, since there is negligible communication overhead for the simulator. In both panels of Fig. 9, multi-node configurations are encouraged when simulating less than 10 synapses per neuron in the network. This claim is challenged, to a degree, when simulating very high population counts (more than 500,000 neurons), since even a small amount of synapses per neuron can impose a non-trivial communication overhead.

5.2. Simulator sensitivity to workload parameters

It is clear that fully understanding the performance patterns exhibited by a biologically-accurate multinode simulator is not a

trivial task. The simulator presented in this paper works on x86-based processors, which are very well-documented and have been extensively studied. Furthermore, in this paper, the parameter space we explored relates to network size, density and connectivity distribution, as described in Table 2. In this strictly-defined parameter space, the simulator behavior, as depicted in Figs. 6–9, clearly shows that even small changes to its parameters can have a large impact on performance. Furthermore, this phenomenon is exacerbated by increasing the amount of available computational resources.

Since predicting simulator behavior in any given parameter space is hard, one is encouraged to create maps similar to the one featured in Fig. 9, in order to discern emerging trends. Such maps aid in choosing simulator configuration for future research in related areas. This map generation process can be efficiently deployed in a Cloud setting. Cloud services lend themselves to performing parameter-space explorations by offering processing resources that can be otherwise difficult to access [70,71]. In addition, the resources can be scaled to match problem size in a cost-efficient manner.

Furthermore, when mapping simulator behavior, one is encouraged to increase the scope of parameter exploration as much as resource availability allows. In this manner, the generated map is more effective at conveying hints related to simulator behavior trends. As an example, panel **a** of Fig. 9 partially resembles the image that panel **b** depicts for networks of 1000–20,000 neurons. It is possible that by further increasing the network size, trends that are already visible for Gaussian-distributed connectivity maps become manifest in uniformly-distributed maps as well. This could be attributed to the fact that uniformly-distributed networks face larger inter-node communication penalties; as such, they would require computing heavier workloads before additional computational resources prove to be beneficial.

A fundamental problem with extending parameter size is that heavier workloads demand larger execution times to be calculated. This, in turn, implies longer simulation times for evaluating optimal simulator configurations (here: number of nodes). Given that, for this type of cycle-accurate models, simulator behavior remains largely stable after a small amount of warm-up steps is performed. Thus, it can be beneficial to reduce the amount of simulation steps and increase the range of parameters explored.

6. Conclusion

This paper has discussed the performance and scalability of a computationally complex and biologically accurate neuron simulator on the Intel Xeon Phi Knights Landing processors. The simulator has been designed with a broader manycore architecture in mind, since the KNL hardware assets are found on most x86-based manycore processors [72] and the simulator was written using traditional parallelization techniques of OpenMP and MPI. This approach allows the portability of the simulator and the extraction of meaningful insight concerning the behavior of similar workloads.

The work proves that efficient usage of a small cluster of manycore processors, such as a system of 8 Knights Landing Xeon Phis, is able to achieve satisfactory performance even when facing a very demanding mathematical model of the human neuron, in network and synaptic sizes numbering in the millions and billions, respectively. It constitutes an efficient solution for studying demanding neuronal models in a pursuit of attaining deeper understanding of the human brain's intricate details.

Furthermore, it has been demonstrated that a biologically-accurate simulator exhibits performance patterns that are dictated by problem size and the nature of each network's connectivity map. A point of focus particularly in our analysis was the system's scalability in multinode setups. It has been highlighted that the system is highly sensitive to simulation parameters and as such, careful steps need to be taken in order to discern trends in performance behavior.

Acknowledgments

This research is supported by European Commission project H2020-687628-VINEYARD. The work is also partially supported by a machine allocation on Kabré supercomputer at the Costa Rica National High Technology Center.

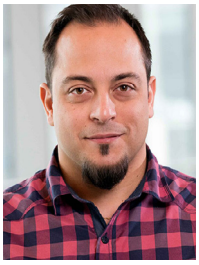
References

[1] H. Markram, The human brain project, Sci. Am. 306 (6) (2012) 50–55.
[2] T.R. Insel, S.C. Landis, F.S. Collins, The NIH brain initiative, Science 340 (6133) (2013) 687–688.
[3] H. Okano, E. Sasaki, T. Yamamori, A. Iriki, T. Shimogori, Y. Yamaguchi, K. Kasai, A. Miyawaki, Brain/minds: a japanese national brain project for marmoset neuroscience, Neuron 92 (3) (2016) 582–590.
[4] R. Ananthanarayanan, S.K. Esser, H.D. Simon, D.S. Modha, The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.
[5] P. Wallisch, M.E. Lusignan, M.D. Benayoun, T.I. Baker, A.S. Dickey, N.G. Hatsopoulos, MATLAB for Neuroscientists: An Introduction to Scientific Computing in MATLAB, Academic Press, 2014.
[6] M.L. Hines, N.T. Carnevale, The neuron simulation environment, Neural Comput. 9 (6) (1997) 1179–1209, doi:10.1162/neco.1997.9.6.1179.
[7] H.E. Plesser, et al., Nest: the neural simulation tool, Enc. Comp. Neurosci. (2015) 1849–1852.
[8] D.F. Goodman, R. Brette, The brian simulator, Front. Neurosci. 3 (2009) 192–197.
[9] G. Chatzikonstantis, D. Rodopoulos, S. Nomikou, C. Strydis, C.I. De Zeeuw, D. Soudris, First impressions from detailed brain model simulations on a Xeon/Xeon-Phi node, in: Proceedings of the ACM International Conference on Computing Frontiers, in: CF '16, ACM, New York, NY, USA, 2016, pp. 361–364, doi:10.1145/2903150.2903477.
[10] H.D. Nguyen, Z. Al-Ars, G. Smaragdos, C. Strydis, Accelerating complex brain–model simulations on GPU platforms, in: Proceedings of the Design, Automation, and Test in Europe, DATE, 2015.
[11] G. Smaragdos, S. Isaza, M.V. Eijk, I. Sourdis, C. Strydis, FPGA-based biophysically-meaningful modeling of olivocerebellar neurons, in: Proceedings of the 22nd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2014.
[12] B. Glackin, J.A. Wall, T.M. McGinnity, L.P. Maguire, L. McDaid, A spiking neural network model of the medial superior olive using spike timing dependent plasticity for sound localization, Front. Comput. Neurosci. 4 (18) (2010).
[13] M. Bhuiyan, A. Nallamuthu, M. Smith, V. Pallipuram, Optimization and performance study of large-scale biological networks for reconfigurable computing, in: Proceedings of the Fourth International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA), 2010, pp. 1–9, doi:10.1109/HPRCTA.2010.5670796.
[14] T. Yamazaki, J. Igarashi, Realtime cerebellum: a large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit, Neural Netw. 47 (2013) 103–111 Computation in the Cerebellum, doi:10.1016/j.neunet.2013.01.019.
[15] C.I. De Zeeuw, et al., Microcircuitry and function of the inferior olive, Trends Neurosci. 21 (9) (1998) 391–400.
[16] A.L. Hodgkin, A.F. Huxley, Propagation of electrical signals along giant nerve fibres, Proc. R. Soc. Lond. Ser. B Biol. Sci. 140 (899) (1952) 177–183.
[17] J. Jeffers, J. Reinders, A. Sodani, Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition, Morgan Kaufmann, 2016.
[18] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.-C. Liu, Knights landing: Second-generation intel xeon phi product, IEEE Micro 36 (2) (2016) 34–46.
[19] M. Snir, MPI–the Complete Reference: The MPI core, MIT, 1998.
[20] L. Dagum, R. Menon, Openmp: an industry standard api for shared-memory programming, IEEE Comput. Sci. Eng. 5 (1) (1998) 46–55, doi:10.1109/99.660313.
[21] D. Luebke, Cuda: Scalable parallel programming for high-performance scientific computing, in: Proceedings of the 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2008, pp. 836–838, doi:10.1109/ISBI.2008.4541126.
[22] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J.M. Bower, M. Diesmann, A. Morrison, P.H. Goodman, F.C. Harris, et al., Simulation of networks of spiking neurons: a review of tools and strategies, J. Comput. Neurosci. 23 (3) (2007) 349–398, doi:10.1007/s10827-007-0038-6.
[23] S. Ghosh-Dastidar, H. Adeli, Spiking neural networks, Int. J. Neural Syst. 19 (04) (2009) 295–308.
[24] Y.-H. Liu, X.-J. Wang, Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron, J. Comp. Neurosci. 10 (1) (2001) 25–45.

[25] M.J. Chacron, et al., Interspike interval correlations, memory, adaptation, and refractoriness in a leaky integrate-and-fire model with threshold fatigue, Neural Comput. (2003) 253–278, doi:10.1162/089976603762552915.
[26] R. Brette, W. Gerstner, Adaptive exponential integrate-and-fire model as an effective description of neuronal activity, J. Neurophysiol. 94 (5) (2005) 3637–3642.
[27] E. Shlizerman, P. Holmes, Neural dynamics, bifurcations, and firing rates in a quadratic integrate-and-fire model with a recovery variable. i: deterministic behavior, Neural Comput. 24 (8) (2012) 2078–2118.
[28] X. Sun, M. Perc, Q. Lu, J. Kurths, Spatial coherence resonance on diffusive and small-world networks of Hodgkin–Huxley neurons, Chaos Interdiscip. J. Nonlinear Sci. 18 (2) (2008) 023102.
[29] Q. Wang, M. Perc, Z. Duan, G. Chen, Delay-enhanced coherence of spiral waves in noisy Hodgkin–Huxley neuronal networks, Phys. Lett. A 372 (35) (2008) 5681–5687.
[30] M. Ozer, M. Perc, M. Uzuntarla, Controlling the spontaneous spiking regularity via channel blocking on Newman-Watts networks of Hodgkin-Huxley neurons, EPL Europhys. Lett. 86 (4) (2009) 40008.
[31] M. Ozer, M. Perc, M. Uzuntarla, Stochastic resonance on Newman–Watts networks of Hodgkin–Huxley neurons with local periodic driving, Phys. Lett. A 373 (10) (2009) 964–968.
[32] M. Ozer, M. Uzuntarla, M. Perc, L.J. Graham, Spike latency and jitter of neuronal membrane patches with stochastic Hodgkin–Huxley channels, J. Theor. Biol. 261 (1) (2009) 83–92.
[33] E.M. Izhikevich, G.M. Edelman, Large-scale model of mammalian thalamocortical systems, Proc. Nat. Acad. Sci. 105 (9) (2008) 3593–3598.
[34] H. Markram, E. Muller, S. Ramaswamy, M.W. Reimann, M. Abdellah, C.A. Sanchez, A. Ailamaki, L. Alonso-Nanclares, N. Antille, S. Arsever, et al., Reconstruction and simulation of neocortical microcircuitry, Cell 163 (2) (2015) 456–492.
[35] S.K. Sudhakar, S. Hong, I. Raikov, R. Publio, C. Lang, T. Close, D. Guo, M. Negrello, E. De Schutter, Spatiotemporal network coding of physiological mossy fiber inputs by the cerebellar granular layer, PLoS Comput. Biol. 13 (9) (2017) e1005754.
[36] C. Eliasmith, O. Trujillo, The use and abuse of large-scale brain models, Current Opin. Neurobiol. 25 (2014) 1–6.
[37] J. Golowasch, M.S. Goldman, L. Abbott, E. Marder, Failure of averaging in the construction of a conductance-based neuron model, J. Neurophysiol. 87 (2) (2002) 1129–1131.
[38] A.A. Prinz, D. Bucher, E. Marder, Similar network activity from disparate circuit parameters, Nat. Neurosci. 7 (12) (2004) 1345.
[39] E. Marder, J.-M. Goaillard, Variability, compensation and homeostasis in neuron and network function, Nat. Rev. Neurosci. 7 (7) (2006) 563.
[40] M. Hines, S. Kumar, F. Schürmann, Comparison of neuronal spike exchange methods on a blue gene/p supercomputer, Front. Comput. Neurosci. 5 (2011) 1–15.
[41] S. Kunkel, M. Schmidt, J.M. Eppler, H.E. Plesser, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, M. Helias, Spiking network simulation code for petascale computers, Front. Neuroinform. 8 (2014) 78, doi:10.3389/fninf.2014.00078.
[42] M. Beyeler, et al., Carlsim 3: a user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks, in: Proceedings of the IJCNN, 2015, pp. 1–8.
[43] R.V. Hoang, D. Tanna, L.C. Jayet Bray, S.M. Dascalu, F.C. Harris Jr, A novel cpu/gpu simulation environment for large-scale biologically realistic neural modeling, Front. Neuroinform. 7 (2013) 1–10.
[44] A. Sripad, G. Sanchez, M. Zapata, V. Pirrone, T. Dorta, S. Cambria, A. Marti, K. Krishnamourthy, J. Madrenas, Snava-a real-time multi-fpga multi-model spiking neural network simulation architecture, Neural Netw. 97 (2018) 28–45.
[45] G. Florimbi, E. Torti, S. Masoli, E. D'Angelo, G. Danese, F. Leporati, The human brain project: parallel technologies for biologically accurate simulation of granule cells, Microprocessors Microsyst. 47 (2016) 303–313.
[46] H.A.Du Nguyen, et al., Accelerating complex brain-model simulations on gpu platforms, in: Proceedings of the DATE, 2015, pp. 974–979.
[47] G. Chatzikonstantis, D. Rodopoulos, C. Strydis, C.I. De Zeeuw, D. Soudris, Optimizing extended hodgkin-huxley neuron model simulations for a xeon/xeon phi node, IEEE Trans. Parallel Distrib. Syst. 28 (9) (2017) 2581–2594.
[48] W.W. Lytton, A.H. Seidenstein, S. Dura-Bernal, R.A. McDougal, F. Schürmann, M.L. Hines, Simulation neurotechnologies for advancing brain research: parallelizing large networks in neuron, Neural Comput. 28 (10) (2016) 2063–2090.
[49] M.J. Bezaire, I. Raikov, K. Burk, D. Vyas, I. Soltesz, Interneuronal mechanisms of hippocampal theta oscillation in a full-scale model of the rodent ca1 circuit, eLife (2016), doi:10.7554/eLife.18566.
[50] D.F. Goodman, R. Brette, Brian: a simulator for spiking neural networks in python, Front. Neuroinform. 2 (2008) 1–10.
[51] M. Stimberg, D.F. Goodman, R. Brette, M. De Pittà, Modeling neuron-glia interactions with the brian 2 simulator, bioRxiv (2017) 198366.
[52] S. Kunkel, W. Schenck, The nest dry-run mode: Efficient dynamic analysis of neuronal network simulation code, Front. Neuroinform. 11 (2017) 1–15.
[53] H. Demuth, M. Beale, MATLAB: The Language of Technical Computing; Neural Network Toolbox; User's Guide; Version 3, MathWorks, 1998.
[54] J.S. Sherfey, A.E. Soplata, S. Ardid, E.A. Roberts, D.A. Stanley, B.R. Pittman-Polletta, N.J. Kopell, Dynasim: a matlab toolbox for neural modeling and simulation, Front. Neuroinform. 12 (2018) 1–15.
[55] J. Senk, C. Carde, E. Hagen, T.W. Kuhlen, M. Diesmann, B. Weyers, VIOLA - A multi-purpose and web-based visualization tool for neuronal-network simulation output, arXiv preprint arXiv:1803.10205 (2018).
[56] S.G. Aleksin, K. Zheng, D.A. Rusakov, L.P. Savtchenko, Arachne: A neural-neuroglial network builder with remotely controlled parallel computing, PLoS Comput. Biol. 13 (3) (2017) e1005467.
[57] J. Schemmel, et al., A wafer-scale neuromorphic hardware system for large-scale neural modeling, in: Proceedings of the IEEE ISCAS, 2010.
[58] Q. Wu, et al., Development of fpga toolbox for implementation of spiking neural networks, in: Proceedings of the CSNT, 2015, pp. 806–810.
[59] A.K. Fidjeland, et al., Nemo: a platform for neural modelling of spiking neurons using gpus, in: Proceedings of the IEEE ASAP, 2009, pp. 137–144.
[60] A. Ahmadi, H. Soleimani, A gpu based simulation of multilayer spiking neural networks, in: Proceedings of the 19th Iranian Conference on Electrical Engineering (ICEE), IEEE, 2011, pp. 1–5.
[61] E.M. Izhikevich, et al., Simple model of spiking neurons, IEEE Trans. Neural Netw. 14 (6) (2003) 1569–1572.
[62] M. Bhuiyan, et al., Acceleration of spiking neural networks in emerging multi--core and gpu architectures, in: Proceedings of the IPDPSW, 2010.
[63] G. Smaragdos, S. Isaza, M.F. van Eijk, I. Sourdis, C. Strydis, Fpga-based biophysically-meaningful modeling of olivocerebellar neurons, in: Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays, in: FPGA '14, ACM, New York, NY, USA, 2014, pp. 89–98, doi:10.1145/2554688.2554790.
[64] C.I. De Zeeuw, C.C. Hoogenraad, S. Koekkoek, T.J. Ruigrok, N. Galjart, J.I. Simpson, Microcircuitry and function of the inferior olive, Trends Neurosci. 21 (9) (1998) 391–400.
[65] J.R. De Gruijl, P. Bazzigaluppi, M.T. de Jeu, C.I. De Zeeuw, Climbing fiber burst size and olivary sub-threshold oscillations in a network setting, PLoS Comput. Biol. 8 (12) (2012) e1002814.
[66] J. Jeffers, J. Reinders, Intel Xeon Phi Coprocessor High-Performance Programming, Elsevier, 2013.
[67] A. Sodani, Knights landing (knl): 2nd generation Intel® Xeon Phi processor, in: Proceedings of the Hot Chips 27 Symposium (HCS), IEEE, 2015, pp. 1–24.
[68] B. Hellwig, A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex, Biol. Cybern. 82 (2) (2000) 111–121.
[69] V. Senn, S.B. Wolff, C. Herry, F. Grenier, I. Ehrlich, J. Gründemann, J.P. Fadok, C. Müller, J.J. Letzkus, A. Lüthi, Long-range connectivity defines behavioral specificity of amygdala neurons, Neuron 81 (2) (2014) 428–437.
[70] K.R. Jackson, K. Muriki, L. Ramakrishnan, K.J. Runge, R.C. Thomas, Performance and cost analysis of the supernova factory on the amazon aws cloud, Sci. Prog. 19 (2–3) (2011) 107–119.
[71] Y. Shao, L. Di, Y. Bai, B. Guo, J. Gong, Geoprocessing on the amazon cloud computing platform - aws, in: Proceedings of the First International Conference on Agro-Geoinformatics (Agro-Geoinformatics), IEEE, 2012, pp. 1–6.
[72] J. Jeffers, J. Reinders, High Performance Parallelism Pearls Volume Two: Multi-core and Many-core Programming Approaches, Morgan Kaufmann, 2015.

George Chatzikonstantis obtained his Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 2013. His research interests focus on highperformance computing, multi-core/single-chip multiprocessors and bioinformatics. He is currently conducting research on neuromodeling applications in high-performance computing fabrics as a Ph.D. student in NTUA.

Harry Sidiropoulos graduated from the School of Electrical and Computer Engineering from the National Technical University of Athens in 2010. He then completed his Ph.D. studies on the topic of FPGA Architecture and Tool Suites, in the Microprocessors and Digital Systems Laboratory of NTUA. He has been working as a post-doctoral researcher since 2016, specializing in algorithm optimization, ongoing FPGA architectural challenges, cloud computing and computational neuroscience.



Christos Strydis obtained his Ph.D. Degree in Computer Engineering from the Delft University of Technology in 2011. His interests revolve around the topics of high-performance computational neuroscience and of nextgeneration implantable medical devices. Currently, he is an assistant professor with the Neuroscience department of the Erasmus Medical Center, the Netherlands, and is also a chief engineer with Neurasmus BV, the Netherlands.

Mario Negrello obtained a mechanical engineering degree in Brazil in 1997 and later served in the industry of VW until 2004. He then obtained his Masters degree in 2006 and Ph.D. (summa cum laude) in Cognitive Science at the University of Osnabrück, Germany in 2009. At the Fraunhofer Institute in Sankt Augustin of Germany, he researched artificial evolution of neural network controllers for autonomous robots. He acted as a group leader at the Computational Neuroscience laboratory in Okinawa Institute of Science and Technology and now leads a neuroscience lab that combines empirical research with computational method in Erasmus MC, Rotterdam.

George Smaragdos obtained his Diploma from the Technical University of Crete in the department of Electronics and Computer Engineering in 2008. He was then admitted to Delft University of Technology, Netherlands and obtained his M.Sc. in 2012. His master thesis presented an “Adaptive Defect-Tolerant Multiprocessor Array Architecture”. He now conducts research on reconfigurable hardware, computer architecture, fault-tolerant computing, scientific computing and embedded systems.

Chris I. De Zeeuw received his Ph.D. with a focus in brain and behavior in 1990 and his MD in 1991 from Erasmus University Rotterdam. He focuses on the nerve cells in the cerebellum responsible for learning and the effect of their electrical activity on movement. He is currently the director of Neurasmus BV, the Chairman of the Department of Neuroscience at Erasmus MC Rotterdam and the Project Director at the Netherlands Institute for Neuroscience in Amsterdam.

Dimitrios Soudris received his Ph.D. Degree in Electrical Engineering from the University of Patras in 1992. His research interests include embedded systems design, reconfigurable architectures, reliability and low power VLSI design. He is currently working as Associate Professor in School of Electrical and Computer Engineering, Dept. Computer Science of NTUA, Greece.