From Knights Corner to Landing: a Case Study Based on a Hodgkin-Huxley Neuron Simulator

George Chatzikonstantis[1], Diego Jiménez[2], Esteban Meneses[23], Christos Strydis[4], Harry Sidiropoulos[1], and Dimitrios Soudris[1]

[1] Microprocessors and Digital Systems Lab, National Technical University of Athens
[georgec, harry, dsoudris]@microlab.ntua.gr
[2] Advanced Computing Laboratory, Costa Rica National High Technology Center
djimenez@cenat.ac.cr, emeneses@cenat.ac.cr
[3] School of Computing, Costa Rica Institute of Technology
[4] Neuroscience Department, Erasmus Medical Center Rotterdam
c.strydis@erasmusmc.nl

Abstract. Brain modeling has been presenting significant challenges to the world of high-performance computing (HPC) over the years. The field of computational neuroscience has been developing a demand for physiologically plausible neuron models, that feature increased complexity and thus, require greater computational power. We explore Intel's newest generation of Xeon Phi computing platforms, named Knights Landing (KNL), as a way to match the need for processing power and as an upgrade over the previous generation of Xeon Phi models, the Knights Corner (KNC). Our neuron simulator of choice features a Hodgkin-Huxley-based (HH) model which has been ported on both generations of Xeon Phi platforms and aggressively draws on both platforms' computational assets. The application uses the OpenMP interface for efficient parallelization and the Xeon Phi's vectorization buffers for Single-Instruction Multiple Data (SIMD) processing. In this study we offer insight into the efficiency with which the application utilizes the assets of the two Xeon Phi generations and we evaluate the merits of utilizing the KNL over its predecessor. In our case, an out-of-the-box transition on Knights Landing, offers on average 2.4× speed up while consuming 48% less energy than KNC.

Keywords: Intel Xeon Phi, Knights Landing, Computational Neuroscience

1 Introduction

In recent years neuroscientists have been gradually revealing details of neuron operation. Using this knowledge, there is a wide research interest in studying the behaviour of single-neuron, a network of neurons and eventually study brain-wide populations of neurons. Simulating these neuronal networks on various platforms is an active field of research [3,19].

A major challenge is the sheer computational complexity that many of these neuron models entail. Even the less complex types have significant demands as

the studied neuronal network increases in size both in terms of computation and data transfer or storage. Traditionally in the domain of neuroscience, the most common methods for simulating neuron models and studying their behaviour were either through widely-known mathematical software suites such as MATLAB [24] or through specific neuromodeling tools like NEURON [13] and Brian [12]. It has become clear that these methods are not suitable for simulating neuronal networks of realistic sizes and high detail within a reasonable timeframe for brain research. High-Performance Computing (HPC) has been recently recognized as a viable domain for providing a variety of solutions to cope with this limitation [2, 5, 11, 18, 22, 25].

In our current case study we feature a simulator for biophysically plausible neuron models, targeting a part of the human brain named the Inferior Olivary Nucleus, which specializes in the coordination and learning of motor function [9]. The modeling accuracy is at the cell conductance level (Hodgkin and Huxley models [14]), belonging at an analytical and complicated class of models which allow us to expose fine details of the neuron's mechanisms. This workload is an excellent candidate for parallelization on HPC architectures, such as the Intel Xeon Phi system [10], due to the large inherent parallelism of the models. Additionally, it constitutes a realistic worst-case scenario in terms of model complexity, hence a benchmark for neuron modeling workloads.

In order to explore whether Intel's newest generation of the Xeon Phi computing platform, named Knights Landing (KNL), is a suitable platform for neuroscientific workloads, in the current paper we evaluate its performance and energy consumption compared to the previous version, Knights Corner (KNC). We utilize the aforementioned Inferior Olivary Nucleus simulator, named InfOli, which was developed for the KNC generation of Xeon Phi [6]. This comparison will highlight how the evolution of Intel's Xeon Phi architecture can improve the performance of a challenging application in the field of computational neuroscience. Since the application is fine-tuned to the previous version of Xeon Phi processors, we will, accordingly, explore the behaviour of an "out-of-the-box" application on the KNL.

In this paper, we shall first discuss the nature and parallelization method of our simulator. We will then briefly present the architecture of the two generations of Xeon Phi HPC architectures and highlight their significant differences in hardware. Furthermore, we will present the methodology of our experimentation and evaluate their results. Finally, we will conclude with remarks on the merits and shortcomings of each platform.

2 System Design

Software

The InfOli simulator, depicted in Figure 1, is a transient simulator; brain activity is calculated in simulation steps, with each step set to represent 50us of activity in a fixed manner. The steps are calculated sequentially, until the entirety of the requested brain activity is computed.

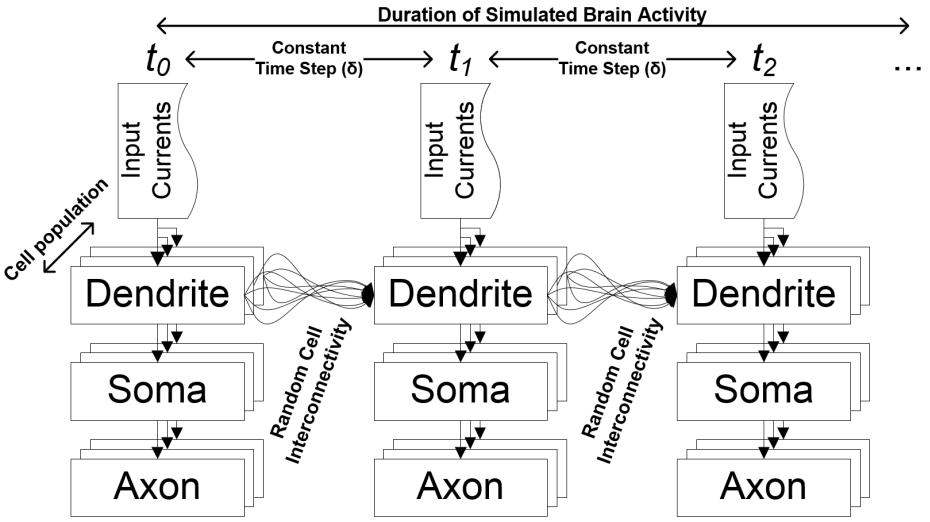


Fig. 1: The InfOli Simulator [6]

In each simulation step, the simulator has the task of updating the status of each neuron in a pre-defined network. The neurons are based on an elaborate, realistic model of the human neuron, derived from the work of Hodgkin and Huxley [14]. As such, they are comprised of 3 compartments, each modelling a different part of the neuron. The dendritic compartment holds the important task of communicating with the rest of the network; it forms connections with other neurons in the network, modelled as electrical synapses named Gap Junctions (GJ) [8]. The somatic compartment is the main body of the neuron, where most calculations for the neuron's inner state take place. Finally, the axonal compartment acts as the output port of the neuron (specifically, in our application, of the Inferior Olivary neuron) to other parts of the brain (such as, the cerebellum). In each step, the simulator processes the current flow in the GJs of the network and then, re-calculates the states of the three compartments of each neuron. This is achieved by solving the Ordinary Differential Equations (ODE) governing the model via the Euler forward method [20]. Each neuron may also receive an external stimulus by its environment, in each step of the simulation.

In order to boost simulation speed, OpenMP [7] has been employed to parallelize the application. Figure 2 relays how the simulator utilizes OpenMP threads. The network is divided in equal parts and assigned to different OpenMP threads, ensuring a balanced distribution of workload.

In each step, the threads read from the Xeon Phi's shared memory in order to calculate the state of their assigned neurons' GJs. This task requires that each thread accesses other threads' data concerning the dendritic compartments of their assigned neurons; these shared memory accesses enforce the flushing of cache lines that hold invalid data from previous simulation steps. After all relevant dendritic data is refreshed, the state of each neuron can be calculated independently from

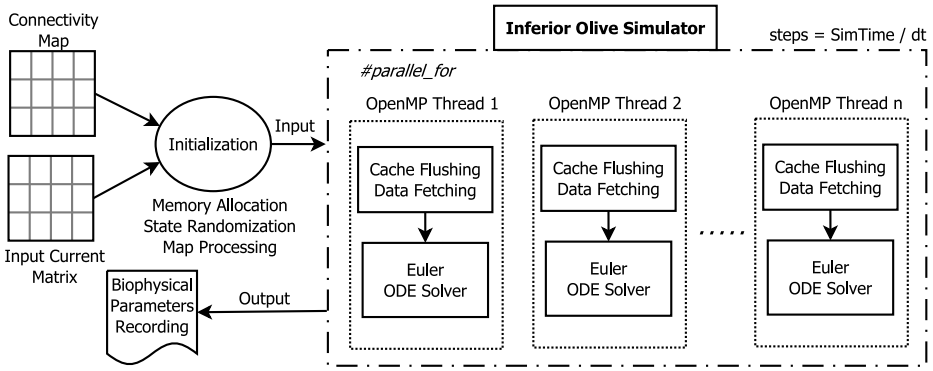


Fig. 2: OpenMP Implementation of the InfOli Simulator

the rest of the network. When the simulation step is completed, biological data that needs to be tracked, such as the voltage levels of the somatic membrane, is collected from each thread and recorded in the simulation's output file.

The simulator has been ported and tested primarily on the Intel KNC. An analytic methodology has been followed to boost vectorization processing unit (VPU) usage, in order to optimally utilize the platform's asset [6]. Data transformations (struct to arrays), aligning data to cache lines and loop transformations have been tested on the KNC, with the help of Intel's profiling tools (Intel VTune Amplifier). As such, the simulator is not expected to be optimal for the second generation of Xeon Phi, the KNL. However, due to their similar architectural design, the application is a good candidate for porting on both platforms.

Hardware

The first commercial generation of Xeon Phi products is named Knights Corner. This version of Xeon Phi is an Intel accelerator platform arranged in a host-and-coprocessor fashion and features up to 61 cores, each with four instruction streams [15]. It supports traditional parallel-programming paradigms, such as MPI [23] and OpenMP [7], in contrast to Graphics Processing Units (GPU) requiring platform-specific programming paradigms [1]. After the Xeon host boots a KNC-specific software stack on the Phi, named Intel Manycore Platform Software Stack (MPSS), the latter may be used independently, for native workload execution. The KNC accelerator features vectorization processing units (VPU) [15], which can parallelize multiple floating-point (FP) operations.

Intel's second generation of Xeon Phi processors introduced several architectural differences with respect to its predecessor. KNL is a standard Intel Many-Integrated Core (MIC) Architecture standalone processor that can boot stock operating systems and connect to a network directly via common interconnects such as Infiniband, Ethernet, etc. This is a significant differentiation over Knights Corner, which is a a PCIe-connected device and, therefore, could only be used when connected to a separate host processor. In KNL, cores are

integrated in couples into structures named tiles in which they share a 1 MB L2 cache. Each core is connected to two vector processing units as opposed to the single VPU unit per core present in KNC models, making vectorization a key aspect in this platform's computational power. KNL processors can have up to 36 tiles for a total of 72 cores, each capable of hyperthreading with up to 4 threads per core, and 144 VPUs. Communication between those 36 tiles is achieved through a cache-coherent 2D mesh interconnect which replaces the bidirectional ring bus used on the KNC coprocessor. This on-die interconnect allows for different clustering modes of operation which offer various degrees of address affinity to improve performance in HPC applications.

In addition to these features, KNL introduced a new memory architecture to provide both large memory capacity as well as high memory bandwidth. To do so, traditional DDR memory is complemented with what Intel named MultiChannel Dynamic Random Access Memory (MCDRAM). This on-package memory does not achieve higher single data access performance than main memory but supports a higher bandwidth [16]. As with the mesh clustering modes, MCDRAM can be configured in different memory modes: i) to serve as cache for the DDR memory (cache mode), ii) to be mapped as regular memory into the system's address space (flat mode) or, iii) to work as hybrid memory where part of the MCDRAM acts as cache and the rest is allocated to the address space (hybrid mode). KNL's characteristics and its high degree of customization make it a suitable platform for high perfomance computing applications like the Inferior Olive simulator.

3 Evaluation

Experimental Setup

The measurements presented in this section have been carried out using two different generations of Intel Xeon Phi. The Knights Corner co-processor's model is 3120P, featuring 57 cores at 1.1GHz, each supporting up to 4 threads running concurrently via multithreading technology. Cores run at 300W thermal design power (TDP). The application is designed to run natively on the co-processor, thus excluding any impact from its Intel Xeon host on its measured performance. Specifically, after compiling and transfering via Secure Copy Protocol (scp) all necessary binaries to the co-processor, the host remains idle throughout the experiment.

The Knights Landing processor's model is 7210, with 64 cores at 1.3GHz and similar multithreading capacities. Its TDP is noticeably lower at 215W. MCDRAM for the KNL was set to cache mode as this setting is completely transparent to software and allows for "out-of-the-box" codes like the neuron simulator being tested, to take advantage of the high-bandwidth-memory technology. As for the clustering mode, quadrant configuration was chosen based on recognition that the cache-quadrant combination offers performance gain to HPC applications [16, 21].

Finally, in order to get a better grasp on the performance offered by the two generations of accelerator platforms, we include performance curves from

an Intel Xeon E5-2609-v2, a 4-core server-grade processor utilizing 4 threads concurrently. The processor's simulation speed acts as a baseline, with the added benefit that codebases developed for Xeon Phi accelerators are compatible with Xeon (or any generic x86) processors.

For the power measurements in this section, different methodologies have been followed for the two platforms. For the Knights Landing processor, the processor's power consumption was sampled via Intelligent Platform Management Interface (IPMI) [17] via a script running concurrently with each experiment's execution. Polling frequency was set to approximately 1 Hz. Energy consumption for each experiment was then calculated by integrating the power samples over the simulation's duration. On the other hand, power measurements on the Knights Corner co-processor is achieved by accessing the host's logs of information and errors regarding the co-processor. These logs are attained via a built-in tool named micrasd which can track the KNC's power in intervals of 5 milliseconds. The reports are generated from the beginning of the simulation and by summation of each report until the end of the experiment, an accurate estimation of total energy consumption can be attained.

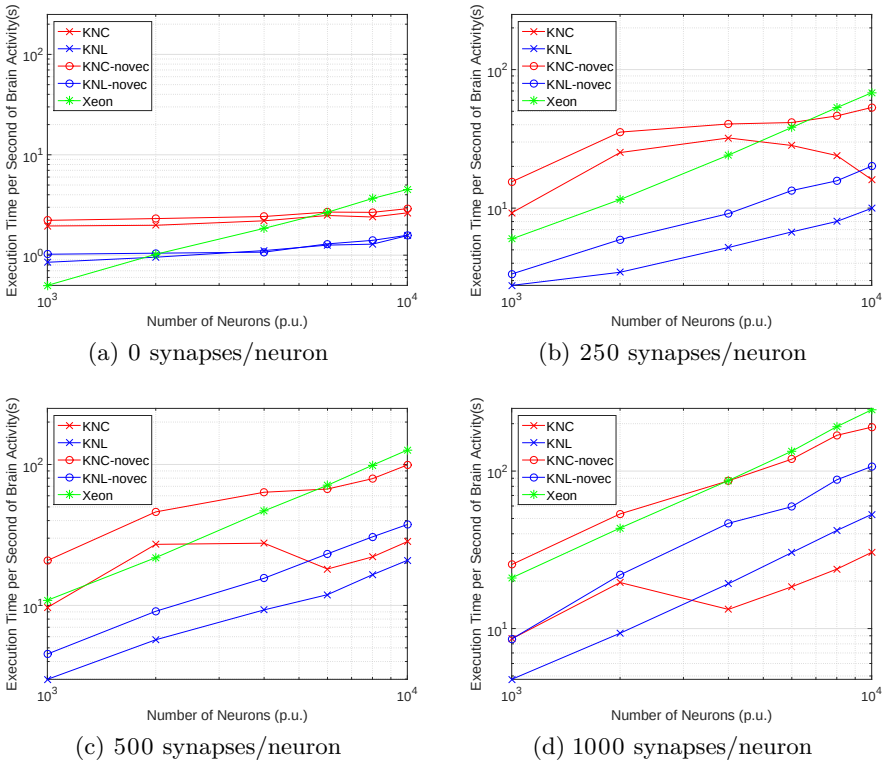
In each experiment, a network of neurons connecting to each other via the Gap Junction mechanism, explained in Subsection 2, is generated. The neuron connections are generated randomly, with each pair of neurons given a chance to form a bond regardless of their position on the neuronal grid. This chance is calculated based on the amount of connections each neuron is designed to have for each experiment, as well as the total neuronal network size; a division of the two variables calculates the network's average connection density, which, in turn, directly leads to the chance of a pair of neurons forming a bond.

Compilation for the KNC has been carried out using Intel's compiler icc version 16.0.1, whereas on the KNL, icc Version 17.0.0.098 has been used. On both platforms, the options used for vectorized code are `-O3` for the best available compiler optimizations, `-vec-report6` for a detailed analysis of vectorized code generated, `-opt-subscript-in-range` to inform the compiler that no integer in the main loop is calculated exceeding the value of 2^{31}, allowing more loop transformations and `-lm` to access math libraries needed throughout the model's calculations. For measurements that use unvectorized code, the options `-no-vec -no-simd -no-qopenmp-simd` have been utilized to ensure the compiler avoids all SIMD commands.

Experimental Results

In Figure 3, we can observe obtained simulation speed of each platform for networks of varying connectivity density. The measurements explore varying network sizes, where each neuron has a fixed average amount of connections to the rest of the network.

All experiments in Figure 3 have been carried out using approximately the maximum amount of threads available to each platform. For the KNC, we used 220 threads, whereas the KNL offered 256 threads. On average the KNL platform outperforms the KNC platform by 2.4× in terms of execution time. The maximum



(a) 0 synapses/neuron

(b) 250 synapses/neuron

(c) 500 synapses/neuron

(d) 1000 synapses/neuron

Fig. 3: Execution Time per second of simulated brain activity, comparison between Knights Corner (KNC) and Knights Landing (KNL) on different Simulator configurations. Performance on Xeon processor (4 threads) added as a baseline.

speed-up is 6×, while in some cases the KNC comes in front with up to 1.6× speed up over the KNL. More specifically, we can observe that, in the cases of low connectivity density, which translates to a low amounts of workload per thread, the KNL shows a superior performance to the KNC.

In cases of small workloads, the efficiency in usage of parallelization assets is diminished, thus single-threaded performance becomes much more important for overall simulation speed. The KNL demonstrates a considerably stronger single-threaded processing power and overtakes the KNC by a fair margin. For both the KNL and the KNC, we can observe that the difference between vectorized and unvectorized code is minimal when connectivity density is low; Gap Junctions represent a significant portion of the total workload and thus, when they are few or completely absent, vectorization fails to boost application performance. We can also observe that the Xeon processor, which excels at handling mostly serial code, may even surpass the KNC accelerator for small-scale simulations.

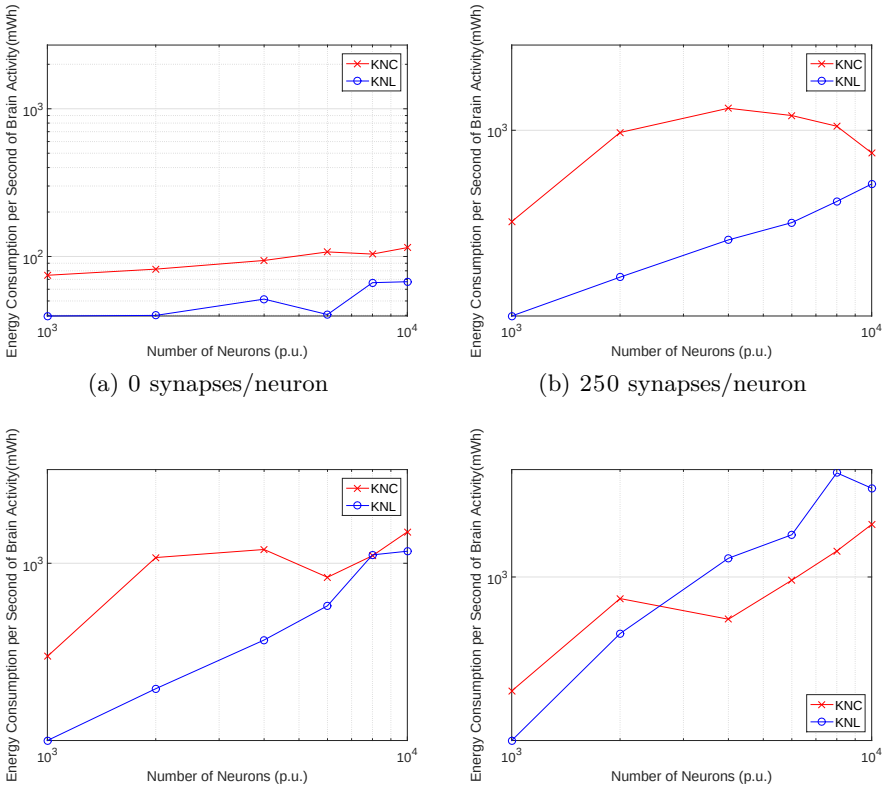
On the other hand, as the computational workload assigned to each thread increases for denser networks, the KNC performs significantly better. The performance gap between the two platforms lessens as the KNC can use its assets with increasing efficiency, since the application has been optimized with the KNC architecture in mind. The gap between vectorized and unvectorized code widens significantly for the KNC, whereas there is a more stable difference in the case of the KNL. Better usage of VPUs leads to the KNC outperforming the KNL; indeed, for workloads of more than 4,000 neurons, each forming approximately 1,000 synapses, the KNL is surpassed by the KNC. As expected, both platforms perform significantly better than the baseline Xeon processor; the KNL and the KNC simulate networks of 10,000 neurons, each with 1,000 synapses, approximately 4.6× and 8.1× faster than the server-grade processor, respectively.

It should be noted that, in terms of performance predictability, the KNL is heavily favoured. Its performance is linear and very predictable. On the contrary, the KNC's performance is harder to anticipate, when operating with vectorization options enabled. The platform's capability to take advantage of its computational resources (threads, VPUs) increases with the supplied workload. Because of this behaviour, it forms a "plateau", during which simulation time for larger networks remains stable, or even lessens, due to better usage of the SIMD commands generated by compiler directives.

Beyond a certain point in network sizes, which differs based on how dense the network is, the aforementioned "plateau" ceases to exist and KNC's performance curve resumes its linear nature. The existence of such "plateaus" impacts the performance predictability of the KNC, whereas the KNL does not exhibit similar behaviour. This can be attributed to the less efficient usage of vectorized code in the KNL's case. For both platforms, unvectorized code, which omits the usage of VPUs, displays a very predictable behaviour.

In Figure 4, we present information regarding the energy required by each computing fabric in order to simulate a second of brain activity, measured in mWh. The Figure is directly linked to Figure 3, since energy consumption is dependent on execution time needed for simulation of each second of brain activity. As such, we can observe similar patterns between the two Figures. On average we have to note that the KNL consumes 48% less energy than the KNC. Because of the KNL's lower TDP and better performance for light workloads, there is a significant reduction in energy consumption when computing for small networks. To put this claim into perspective, whereas the simulation of one second of brain activity in a network of 4000 neurons, with a density of 250 synapses per neuron, requires over $1200mWh$ for the KNC, the KNL consumes under $300mWh$ for the same workload, improving on energy efficiency by a factor of 4×.

On the contrary, due to the KNC's smaller execution times for larger, denser networks, it is preferable from a power consumption standpoint to the KNL for such workloads. A network of 10,000 neurons, each forming 1,000 synapses with the rest of the network, requires 27% less energy on the KNC ($1600mWh$ per



(a) 0 synapses/neuron

(b) 250 synapses/neuron

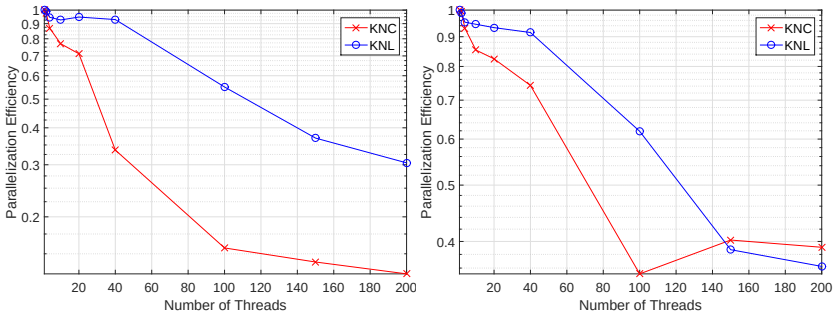
(c) 500 synapses/neuron

(d) 1000 synapses/neuron

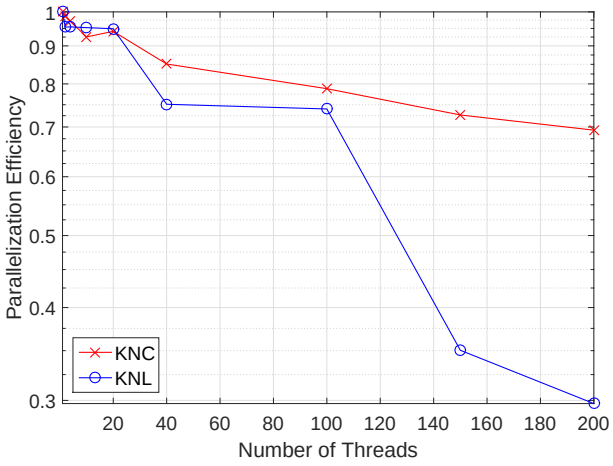
Fig. 4: Energy Consumption per second of simulated brain activity, comparison between Knights Corner (KNC) and Knights Landing (KNL) on different Simulator configurations

second of simulated time) than on the KNL ($2200mWh$ for the same amount of activity).

In Figure 5, information regarding the efficiency with which each platform manages its OpenMP threads is displayed. In HPC, the efficiency with which an application utilizes the underlying platform's resources can be calculated as the speedup yielded by employing said resources, compared to a single-threaded performance, divided by the amount of resources used, such as the number of processors used to run the application, or the number of threads spawned by the application. In our case, we calculated the efficiency metric by dividing execution speedup with the number of OpenMP threads spawned, with a range of OpenMP threads utilized from 1 to 200, on both platforms. In each subfigure, network density has been set to 1,000 synapses per neuron and we explore networks of different size.



(a) 1000 neurons (b) 4000 neurons



(c) 10000 neurons

Fig. 5: Threading Efficiency on the Knights Corner (KNC) and Knights Landing (KNL), for different Simulator configurations

For the KNL, we can observe that the efficiency of utilizing up to approximately 50 threads remains at satisfactory levels. In these cases, each core spawns either one or two threads (due to the selected balanced thread affinity) and, in contrast to the KNC, the KNL's cores operate significantly better when operating with only one thread [16]. The KNL maintains a reliable efficiency for low degrees of threading regardless of the simulated network's size, whereas the KNC's efficiency suffers for small workloads, such as for networks of 4000 neurons.

Larger networks, however, offer better opportunities for the KNC to utilize its computational assets efficiently, maintaining a speedup-to-threads ratio above 70% even for 200 threads. The KNL's threading efficiency sharply declines when employing massive degrees of parallelism, dropping below 40% when using more than 140 threads. The application's inability to utilize the entirety of KNL's

assets efficiently to tackle demanding simulations explains the performance gap between the two platforms for larger workloads. This inability is mostly attributed to the fact that the simulator has been fine-tuned to the KNC environment and has been tested "out-of-the-box" on the KNL.

4 Conclusion and Outlook

In this paper, a computationally demanding application from the field of computational neuroscience that had previously been extensively developed and optimized for the Intel KNC, has been tested "out-of-the-box" for the second generation of Xeon Phi, the KNL. The InfOli biophysically-accurate simulator's performance was tested using a range of workloads, from small, unconnected neuronal populations to larger, dense networks. The results were evaluated from both a simulation-speed and a power-efficiency standpoint. On average KNL offers a speed up of 2.4× while consuming 48% less energy. Smaller workloads, by taking advantage of the KNL's superior single-threaded performance, exhibit very significant gains in both speed and, even more so, energy consumption, with specific experiments demanding 75% less Wh of energy per second of simulated brain activity on the KNL. On the other hand, without further fine-tuning of the application to the architectural details of the KNL, OpenMP-thread efficiency suffers when running on the KNL, causing the simulator to handle more demanding networks poorly, relatively to the optimized KNC version. Furthermore, throughout the whole range of experiments, it has been shown that the KNL offers a more robust, dependable performance curve with little variability.

These findings are promising enough to warrant further optimization of the simulator for the new generation of the Xeon Phi. As future work, we would suggest using an optimized version of the simulator on a cluster of KNL processors, in order to simulate neuronal networks of much larger sizes and take advantage of Intel's OmniPath technology for inter-node communication [4].

Acknowledgments This work is partially supported by European Commission project H2020–687628–VINEYARD.

References

1. CUDA C Programming Guide. Tech. rep., NVIDIA Corporation
2. Bhuiyan, M., Nallamuthu, A., Smith, M., Pallipuram, V.: Optimization and performance study of large-scale biological networks for reconfigurable computing. In: Fourth International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA). pp. 1–9 (nov 2010)
3. Bhuiyan, M. et al.: Acceleration of spiking neural networks in emerging multi-core and gpu architectures. In: IPDPSW (2010)
4. Birrittella, M.S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K.D., Zak, R.C.: Intel® omni-path architecture: Enabling scalable, high performance fabrics. In: High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on. pp. 1–9. IEEE (2015)

5. Chatzikonstantis, G., Rodopoulos, D., Nomikou, S., Strydis, C., De Zeeuw, C.I., Soudris, D.: First Impressions from Detailed Brain Model Simulations on a Xeon/Xeon-Phi Node. In: Proceedings of the ACM International Conference on Computing Frontiers. pp. 361–364. CF '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2903150.2903477
6. Chatzikonstantis, G., Rodopoulos, D., Strydis, C., De Zeeuw, C.I., Soudris, D.: Optimizing extended hodgkin-huxley neuron model simulations for a xeon/xeon phi node. IEEE Transactions on Parallel and Distributed Systems (2017)
7. Dagum, L., Enon, R.: Openmp: an industry standard api for shared-memory programming. IEEE CSE 5(1), 46–55
8. De Zeeuw, C.I., Hoebeek, F.E., Bosman, L.W., Schonewille, M., Witter, L., Koekkoek, S.K.: Spatiotemporal firing patterns in the cerebellum. Nature Reviews Neuroscience 12(6) (2011)
9. De Zeeuw, C. I. et al.: Microcircuitry and function of the inferior olive. Trends in neurosciences 21(9), 391–400 (1998)
10. Fang, J. et al.: Test-driving intel xeon phi. In: ICPE (2014)
11. Glackin, B., Wall, J.A., McGinnity, T.M., Maguire, L.P., McDaid, L.: A spiking neural network model of the medial superior olive using spike timing dependent plasticity for sound localization. Frontiers on Comput. Neurosci. 4(18) (2010)
12. Goodman, D.F., Brette, R.: The brian simulator. Frontiers in neuroscience 3, 26 (2009)
13. Hines, M.L., Carnevale, N.T.: The NEURON simulation environment. Neural computation 9(6) (1997)
14. Hodgkin, A.L., Huxley, A.F.: Propagation of electrical signals along giant nerve fibres. Proceedings of the Royal Society of London. Series B, Biological Sciences 140(899), 177–183 (1952)
15. Jeffers, J., Reinders, J.: Intel Xeon Phi Coprocessor High-Performance Programming. Elsevier (2013)
16. Jeffers, J., Reinders, J., Sodani, A.: Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition. Morgan Kaufmann (2016)
17. Kaufman, G.J., et al.: System and method for application programming interface for extended intelligent platform management (Jun 21 2011), US Patent 7,966,389
18. Nguyen, H.D., Al-Ars, Z., Smaragdos, G., Strydis, C.: Accelerating complex brain-model simulations on GPU platforms . In: Design, Automation, and Test in Europe, DATE 2015 (Mar 2015)
19. Nguyen, H. A. Du et al.: Accelerating complex brain-model simulations on gpu platforms. In: DATE. pp. 974–979 (2015)
20. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical recipes in C, vol. 2. Cambridge university press Cambridge (1996)
21. Rosales, C., James, D., Gómez-Iglesias, A., Cazes, J., Huang, L., Liu, H., Liu, S., Barth, W.: TACC Technical Report TR-16-03 KNL Utilization Guidelines. Tech. rep., University of Texas at Austin, Texas Advanced Computing Center (November 2016), https://portal.tacc.utexas.edu/documents/10157/1334612/KNL+Utilization+Guidelines/95cc0f23-1755-424d-8d29-64a91a09cf33
22. Smaragdos, G., Isaza, S., Eijk, M.V., Sourdis, I., Strydis, C.: FPGA-based Biophysically-Meaningful Modeling of Olivocerebellar Neurons. In: 22nd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA) (Feb 2014)
23. Snir, M.: MPI–the Complete Reference: The MPI core. MIT (1998)

24. Wallisch, P., Lusignan, M.E., Benayoun, M.D., Baker, T.I., Dickey, A.S., Hatsopoulos, N.G.: MATLAB for neuroscientists: an introduction to scientific computing in MATLAB. Academic Press (2014)
25. Yamazaki, T., Igarashi, J.: Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit. Neural Networks 47, 103–111 (2013), http://www.sciencedirect.com/science/article/pii/S0893608013000348, computation in the Cerebellum