Paper

A Novel GPU-Enabled Simulator for Large Scale Spiking Neural Networks

Paweł Szynkiewicz

System Research Institute, Polish Academy of Science, Warsaw, Poland

Abstract—The understanding of the structural and dynamic complexity of neural networks is greatly facilitated by computer simulations. An ongoing challenge for simulating realistic models is, however, computational speed. In this paper a framework for modeling and parallel simulation of biological-inspired large scale spiking neural networks on high-performance graphics processors is described. This tool is implemented in the OpenCL programming technology. It enables simulation study with three models: Integrate-and-fire, Hodgkin-Huxley and Izhikevich neuron model. The results of extensive simulations are provided to illustrate the operation and performance of the presented software framework. The particular attention is focused on the computational speed-up factor.

Keywords—GPU computing, OpenCL programming technology, parallel simulation, spiking neural networks.

1. Introduction

Simulation of biological-inspired Spiking Neural Networks (SNN) is generally a complex problem that involves cumbersome calculations, especially when processing of large scale networks. The restrictions are caused by demands on computer resources, i.e. processor and memory. As biological neural networks become larger and more complex, the required computational power grows significantly. However, the calculations performed by neural networks simulators can be easily partitioned into large number of independent parts and carried out on many cores or computers. It was observed that parallel implementation based on MapReduce programming model improves the efficiency of the simulator and speeds up a calculation process.

A low-cost, an alternative to supercomputers is the Graphical Procession Unit (GPU) – specialized massively-parallel graphics processor that can be used as a general purpose computational accelerator [1]. GPU-enabled parallel computing is a relatively new area of research that has become extremely popular over the last decade and is rapidly increasing its advance into different areas of technology. Last years a model for parallel computing based on the use of GPUs to perform a general purpose scientific and engineering computing was developed and used to solve complex scientific and engineering problems. Using Compute Unified Device Architecture (CUDA) or Open Computing Language (OpenCL) many real-world applications can be easily implemented and run significantly faster than on multi-processor or multi-core systems. GPU clusters are one of the most progressive branches in a field of parallel computing and data processing nowadays.

The paper addresses issues associated with parallel computing systems and the application of GPU technology to large scale systems simulation. During research a dedicated software framework have been developed and designed that can be used to extensive simulation of spiking neural networks on GPU accelerators. This framework has been implemented in the OpenCL programming technology and can be executed on various computing platforms. The relative benefits and limitations of presented software platform have been evaluated based on results of numerical experiments performed for various less and more complex models of neural networks.

The remainder of this paper is organized as follows. A brief survey of biological-inspired SNN models is presented in Section 2. The overview of parallel SNN simulators is provided in Section 3. The organization, implementation and usage of the software framework for SNN simulation on GPU is described in Section 4. The results of simulations of several complex networks on various hardware platforms are presented and discussed in Section 5. Finally, conclusions are drawn in Section 6.

2. Spiking Neural Networks Modeling

A spiking neural network is composed of a set of N spiking neurons and E synapses – links $< i, j > \in E$, $E \subseteq N \times N$ with weights $w_{ij} \geq 0$, [2]–[5]. Excitatory and inhibitory synapses are distinguished. Excitatory synapses are connections of all excitatory neurons, while inhibitory synapses are connections of all inhibitory neurons. A spike is produced when a condition on the state variables is satisfied, for example when the membrane potential exceeds a threshold value (see Fig. 1). In general, various linear or nonlinear threshold functions $V_{th} : \Re^+ \rightarrow \Re^+$ have been defined for various models of spiking neuron. Thus, a biological-inspired spiking neuron can be described as a hybrid system with one or several continuous state variables (membrane potential V, conductances C, etc.), and spikes received through the synapses that trigger changes in some of the variables. Continuous evolution of a number of state variables is usually modeled by a set of differential equa-

tions with discrete events. The hybrid system formalism is presented in [6].

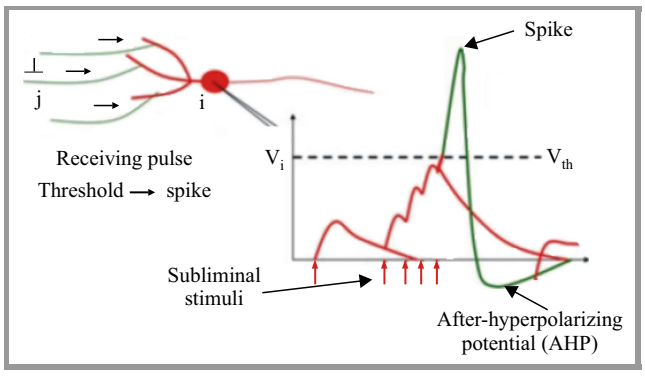

Fig. 1. A model of spiking neuron.

The range of computational problems related to spiking neurons is very large, especially in case when detailed biophysical representations of the neurons have to be used. The example is the reproduction of intracellular electrophysiological measurements. In general, modeling of biologically realistic spiking neural networks requires tuning the enormous number of parameters [7], [8]. In other cases, one does not need to realistically capture the spike generating mechanisms, and simpler models are sufficient.
A survey of models of a spiking neuron is provided in the literature [3], [4], [6], [9]. The research attention is focused on three commonly used models, simple Integrate-and-fire and models developed by Hodgkin and Huxley and Izhikevich.

2.1. Integrate-and-fire Neuron Model

Integrate-and-Fire (I&F) is the simplest spiking neural model described in [3], [6]. Let us refer to V as the neuron's membrane potential (the system state) and to I as the input current. Assuming that I is a sum of excitatory impulses I_E, inhibitory impulses I_I and constant current offset I_{offset} the dynamics of a neural model is described by the following state equations:

$$\frac{dV}{dt} = \frac{1}{\tau}(V_{rest} - V) + \frac{1}{C}\left(I_E + I_I + I_{offset}\right), \tag{1}$$

$$\frac{dI_E}{dt} = -\frac{I_E}{\tau_{synE}}, \tag{2}$$

$$\frac{dI_I}{dt} = -\frac{I_I}{\tau_{synI}}, \tag{3}$$

where τ denotes a model time constant, τ_{synE} and τ_{synI} excitatory and inhibitory synapses time constants, C is membrane capacity, V_{rest} is initial potential of the membrane, V_{reset} is membrane potential after spike (reset potential) and τ_{refrac} stands for relaxation time (time after spike during which neuron is insensitive to stimulation). In I&F model all spikes are generated when $V \geq V_{th}$. Potential after spike is reset to $V \leftarrow V_{reset}$.

2.2. Hodgkin-Huxley Neuron Model

Integrate-and-fire is a simple model that can imitate some of the biological neuron behavior. However, it is unable to reproduce firing patterns like: bursting, chattering, etc. The model developed by Hodgkin and Huxley (H&H) and described in [10] is one of the most successful mathematical model of a complex biological process that has ever been formulated. The idea is that the semipermeable cell membrane separates the interior of the cell from the extracellular liquid and acts as a capacitor. If an input current I is injected into the cell, it may add further charge on the capacitor, or leak through the channels in the cell membrane. In the standard H&H model there are three types of channels: a sodium channel Na, a potassium channel K and an unspecific leakage channel l with resistance R. Assuming that an input current I is a sum of excitatory impulses I_E, inhibitory impulses I_I, constant current offset I_{offset} and externally injected current I_{inj} the model is yield by the state equations:

$$\begin{aligned}\frac{dV}{dt} &= \frac{1}{C}\left[-g_{Na}m^3h(V-E_{Na}) - \right. \\ &- g_K n^4(V-E_K) - g_l(V-E_l) - g_e(V-E_e) - \\ &- \left. g_i(V-E_i) + I_{offset} + I_{inj}\right], \end{aligned} \tag{4}$$

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(V)m, \tag{5}$$

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n, \tag{6}$$

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(V)h, \tag{7}$$

$$\frac{dg_e}{dt} = -\frac{g_e}{\tau_{synE}}, \tag{8}$$

$$\frac{dg_i}{dt} = -\frac{g_i}{\tau_{synI}}, \tag{9}$$

where g_e and g_i denote excitatory and inhibitory synapses conductivity, g_{Na}, g_K, g_l ion channels conductance, E_{Na}, E_K, E_l, E_e, E_i ion channels reverse potentials. The definitions of functions α_m, β_m, α_n, β_n, α_h, β_h are provided in [3]. In order to produce an action potential the membrane potential must be increased quickly enough to cross threshold ($dV/dt \geq V_{th}$).

2.3. Izhikevich Neuron Model

H&H model is computationally expensive and in case of huge networks requires many differential equations solutions. Another kind of formalism which is able to replicate different rich firing patterns achievable with H&H model, using two simple equations with only one super-linear term was proposed by Izhikevich in [11]. Izhikevich model (I) has a computational efficiency similar to I&F model.
Izhikevich reduced biophysically accurate H&H neuronal model to a two-dimensional system of ordinary differential equations of the form

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - U + I, \tag{10}$$

$$\frac{dU}{dt} = a(bV - U), \tag{11}$$

$$\frac{dI_E}{dt} = -\frac{I_E}{\tau_{syn}}, \tag{12}$$

$$\frac{dI_I}{dt} = -\frac{I_I}{\tau_{syn}}, \tag{13}$$

$$\text{if} \quad V \geq 30 \text{ mV}, \quad \text{then} \quad V \leftarrow c, \quad U \leftarrow U + d, \tag{14}$$

where I_E and I_I denote presynaptic currents from excitatory synapses and inhibitory synapses respectively, τ_{syn} synapse time constant (for excitatory and inhibitory $\tau_{syn} = 1$ ms), a, b, c, d model parameters, C membrane capacity.
After the spike reaches its apex (+30 mV), the membrane voltage and the recovery variable are reset according to the Eq. (14). The Eq. (10) was obtained by fitting the spike initiation dynamics of a cortical neuron (other choices also feasible) so that the membrane potential V has mV scale and the time t has ms scale. The resting potential in the model is between −70 and −60 mV depending on the value of b.

3. Simulation of Spiking Neural Networks

3.1. Parallel Simulation of SNN

The simulation of spiking neural networks can be naturally decomposed into three main phases:

- integrating the differential equations that describe the neuron models,
- propagating the spikes to target neurons,
- changing states of target neurons.

It is obvious that the bottleneck for large scale networks simulation is the propagation of numerous spikes across the network considered. Recent research has shown that modern simulators of spiking neural networks can be parallelized and executed on both multi-core CPUs and GPUs regardless of the network topology [12]–[15]. Parallel computation can be applied to all listed phases of the network simulation. The parallelization of the first phase, i.e. numerical integration is straightforward. It follows the Single Instruction, Multiple Data (SIMD) paradigm. The number of operations scale with the number of neurons in a network. The total computational cost for large scale networks is dominated by the second phase in which the operations scales with the number of synapses.
Parallel implementations of SNN are reviewed and discussed in [6]. Three parallelization strategies are proposed and discussed:

- N-parallel – spike propagation is parallelized over neurons. Each thread updates the total input of one neuron;
- S-parallel – spike propagation is parallelized over synaptic evens. Each thread implements the effect of a spike arriving at one synapse. The number of executed threads is limited by the total number of synapses in the network executed at each timestep;
- NS-parallel – combination of both aforementioned strategies N-parallel and S-parallel. This approach is recommended for GPU computing.

The software environments for neural networks simulation can implement two simulation modes:

- time-driven – all neurons are updated simultaneously at each timestep (tick of a global clock) – synchronous distributed simulation,
- event-driven – neurons are updated only when the event occur, i.e. they receive or emit a spike – asynchronous parallel simulation.

Time-driven and event-driven algorithms for SNN simulation are described in [6].

3.2. Survey of SNN Simulators

SNN Simulating on CPU. A survey of software environments for spiking neural networks simulation on CPUs is presented in [6], [14]. NEURON [16] is a commonly used, robust and efficient software platform that can support creation and evaluation of various models of biological neurons and neural circuits. It implements both time-driven and event-driven simulation modes. Moreover, NEURON supports parallel processing on multicore and multiprocessor machines employing threads and distributed processing in clusters using MPI standard. It is available on Unix, Linux and MS Windows platforms. It was executed on Cray and IBM Blue Gene supercomputers.
Neural Simulation Tool (NEST) [17] was created as a result of a long term collaborative project to support the development of technology for neural networks simulation. It is designed to large scale neural systems with heterogenity in neuron and synapses types simulation. It supports parallelization by multi-threading and message passing, and can be executed on multiprocessor machine and in a cluster of computers. NEST implements time-driven simulation mode, and is available on Unix, Linux, MS-Windows and Mac OS platforms. The software is provided to the scientific community under an open source license.
Brian [18], [19] is widely used, highly flexible and easily extensible simulator for spiking neural networks available on almost all platforms (Linux, MS Windows, Mac OS). It provides the implementations of I&F and H&H neuron models, and can be easily extended with the others. This software platform is written in the Python programming language. It is easy to learn and use. Various libraries of methods written in the Python can be used, e.g. NumPy and SciPy for numerical calculations, PyLab for results graphical visualization. Parallel Python can be employed to calculation parallelization. The sources, demos, manual

and publications can be downloaded from the project Web page [19]. It is released under the CeCILL license.

Mvaspike [20] is a general purpose tool for modeling and simulating large and complex biological neural networks. It is based on the event-based modeling and simulation strategy. The focus is on spiking neural networks simulation (integrate-and-fire and other spiking point neurons). A good balance between simulation efficiency and modeling freedom is provided. The core of the system is implemented in C++, however, the access from other programming languages is easy. A parallel implementation is available for multiprocessor machines and clusters.

SNN simulating on GPU. Several software environments for SNN simulation on GPU are described in literature.

NeMo [21], [22] is a high-performance environment for large scale spiking neural network simulation. It simulates systems of Integrate-and-fire and Izhikevich neurons on CUDA-enabled GPUs and uses a powerful scatter-gather messaging algorithm to provide further optimization for sparse random connectivities and supports real time simulation. It is a C++ class library. Moreover, NeMo has bindings for C, Matlab, and Python. The software is provided under an open source license.

GPU-enhanced Neuronal Networks (GeNN) [23] is another framework for simulating SNN on GPU. It is an open source library that generates code to accelerate the execution of network simulations on NVIDIA GPUs. It is entirely based on CUDA and C/C++. It is flexible and easily extended software – any neuron model can be simulated. In GeNN users can introduce their own neuron models, synapse models, post-synaptic integration models and synaptic plasticity rules by providing code snippets that are substituted into the network simulation during code generation. GeNN is available for Linux, Mac OS and Windows platforms.

The Myriad [24] CUDA GPU-enabled simulator focuses on realistic biophysical models using H&H neurons and densely integrated network models that scale poorly on clusters of computers. These models have many analogue interactions such as gap junctions and graded synapses that require many model elements to update one another at each timestep. Myriad provides a flexible and extensible interface through a Python module, which is then translated into a C-based implementation layer by code generation.

Table 1
Summary of selected SNN simulators

	Brian	NEST	NEURON	NeMo
Time-driven	Yes	Yes	Yes	Yes
Event-driven	No	No	Yes	No
GPU	Yes	No	No	Yes
Linux	Yes	Yes	Yes	Yes
Windows	Yes	No	Yes	No
Easy installation	Yes	Yes	No	No

SNN simulators – a summary. Selected parallel environments for SNN simulation were installed and tested by the author of this paper. Table 1 presents the summary of their evaluation. The focus is on their implementation and functionality.

4. SNNS – Spiking Neural Network Simulator

4.1. Description of SNNS

SNNS is a GPU-enabled software environment for spiking neural networks simulation using the OpenCL programming model. The aim was to provide a framework which allows performing effective experiments with less and more complicated models of spiking neural networks on various GPUs. It enables simulation study with three models: Integrate-and-fire, Hodgkina-Huxley and Izhikevich neuron model. SNNS implements time-driven simulation mode and NS-parallel parallelization strategy. Two of the system's principal goals are portability and usage in heterogeneous computing environments. SNNS can be executed on GPUs from many vendors.

During simulation experiment performed under SNNS one can distinguish three main stages: preparatory stage, experimental stage and recording test results. At the preparatory stage a neural model, presynaptic and postsynaptic neurons, spiking generator and all initial parameters (total number of neurons and synaps, initial values of state variables, etc.) are provided. The SNNS framework cooperates with the Brian simulator [19]. The neural network to be simulated is generated using tools from Brian. The special programme for recording the generated network into the disc file in the comprehensive SNNS format has been developed.

The experimental stage begins when all decisions regarding the simulated network are made. The corresponding computing modules (kernels) are executed in sequence. The spikes generated at each timestep are collected, neuron states are updated and new spikes are firing and propagating across the network.

Finally, all test results are recorded into a disc file in the defined format. They can be easy visualized in popular graphical programs. The Brian&SNNS system flow diagram is presented in Fig. 2.

4.2. Architecture of SNNS

Programming for a GPU is rather specialised and needs additional effort from the programmer [1]. Due to the fact that the cores found on GPUs are less complex the programs that are executed should be especially tuned for minimizing their limitations and maximizing their potential to provide high level of parallelization. Algorithms that do not take into account the architecture of the GPU will not use it efficiently. In particular, the constraints on memory access patterns have to be considered. Therefore, tuning GPU algorithms to the specific hardware is highly recommended.

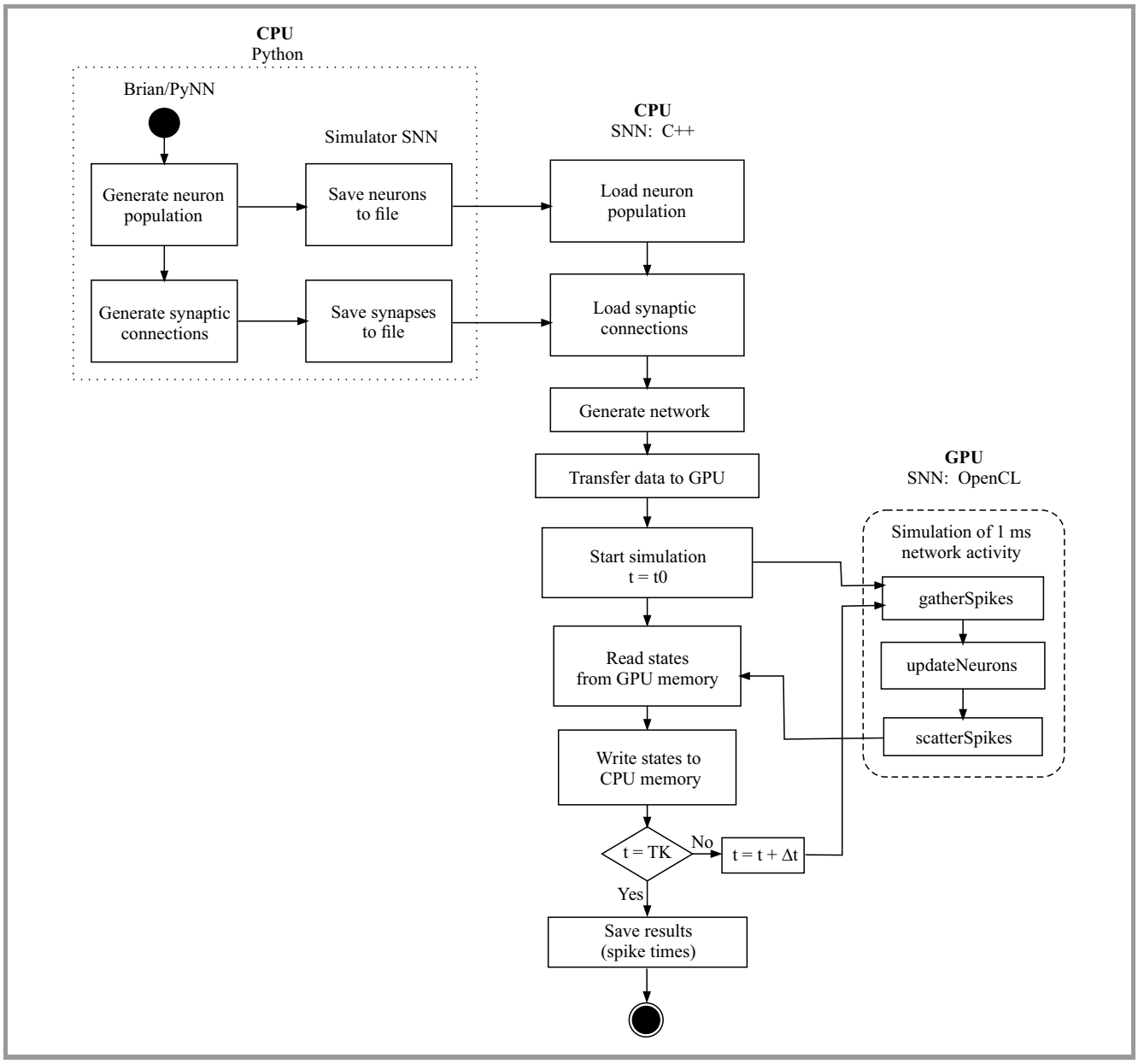

Fig. 2. The Brian&SNNS system flow diagram.

In order to take advantage of GPU accelerators from different vendors OpenCL [25], [26], which is a low level GPU programming toolkit was used. OpenCL is an industry standard computing library developed in 2009 that targets not only GPUs, but also CPUs and potentially other types of accelerator hardware. In OpenCL efficient implementation requires preparation slightly different codes for different devices, however it is much less complicated than writing code in many native toolkits for NVIDIA and AMD devices.

4.3. SNNS Components

SNNS consists of seven components. Its architecture is depicted in Fig. 3. All components have been implemented

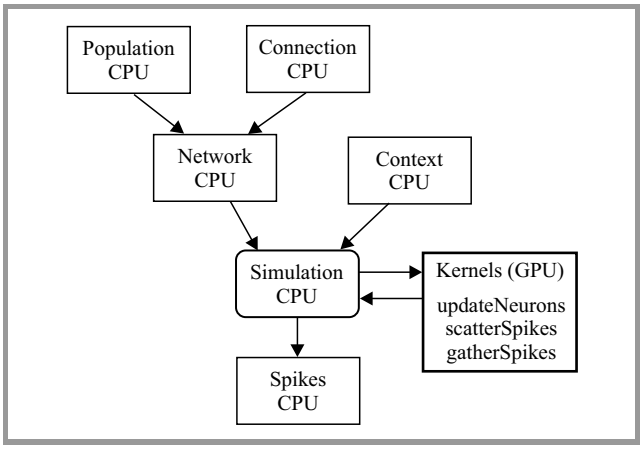

Fig. 3. The SNNS simulator components.

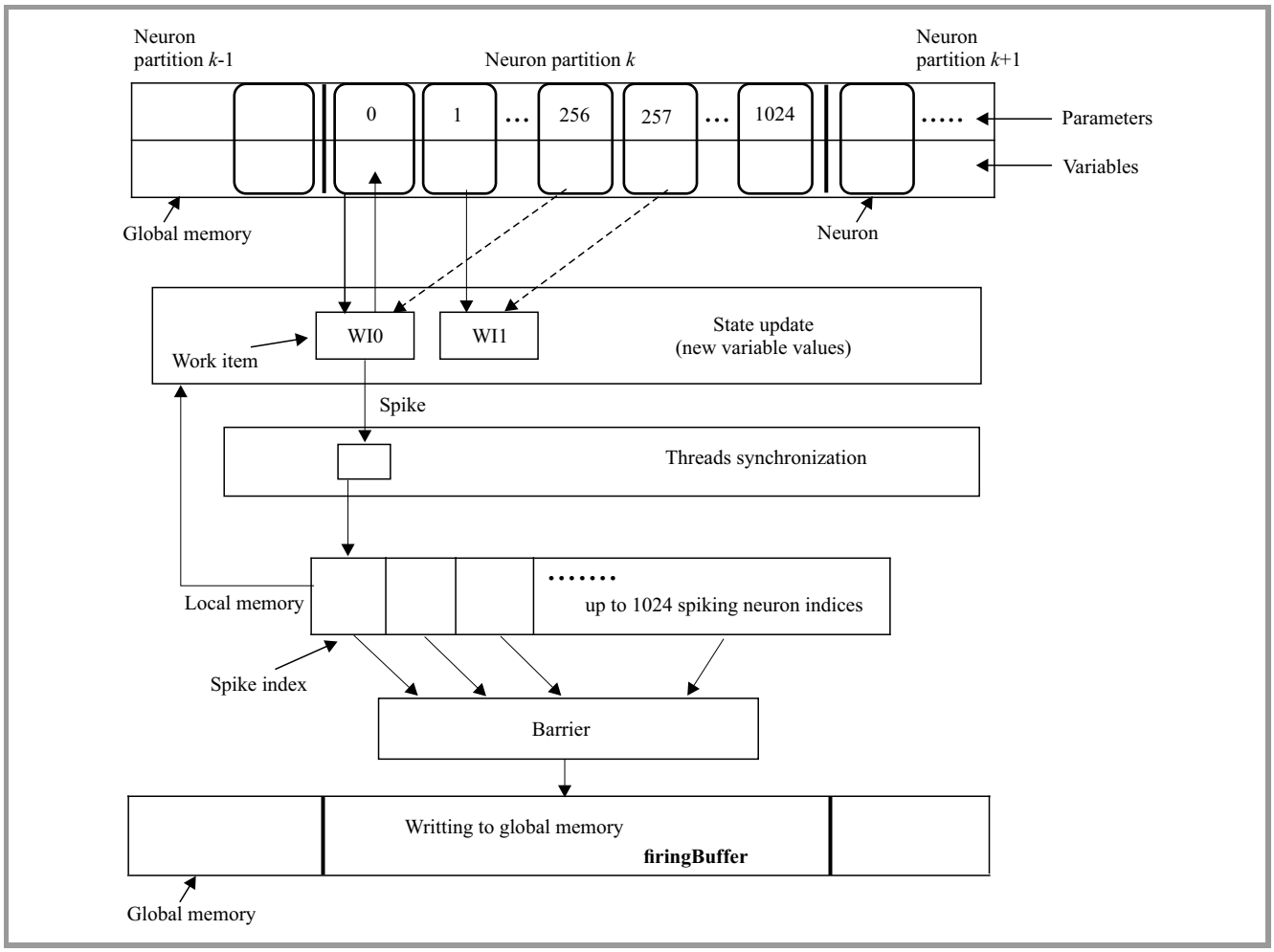

Fig. 4. The performance of the `updateNeurons` kernel.

in OpenCL and C++ and can be executed on CPU and GPU processors. The specification of components is as follows:

- Population – downloading neurons from the disc file (C++),
- Connection – downloading synaptic connections from the disc file (C++),
- Network – generation of network to be simulated (C++),
- Context – CPU – GPU communication mechanisms definition (C++),
- Simulation – a simulation scenario definition (C++),
- Kernels – network simulating (computations) (OpenCL),
- Spikes – test results recording into a disc file (C++).

The goal was to develop an effective, flexible and failure resistance software. Therefore, the main component of the system – *Kernel* – were decomposed into three kernels that perform the following operations:

- `updateNeurons` – neuron states updating,
- `scatterSpikes` – spikes propagating across the network,
- `gatherSpikes` – collecting spikes received by all neurons at each timestep.

The performance of the `updateNeurons` kernel is presented in Fig. 4. It implements the forward and exponential Euler methods for numerical integration [27].

4.4. Memory Issues

It is obvious that mentioned above kernels need to access at each timestep a large amount of memory, since all neurons and synaptic variables corresponding to received spikes have to be accessed. Due to the fact that neuron and synaptic operations are often simple, the speed of memory access limits the efficiency of SNNS. The shared memory of GPU is fast but is very limited. The global memory is very slow. Therefore, the most critical issue is the optimization of read/write memory access to the values of synapses and neural variables at each timestep. The mem-

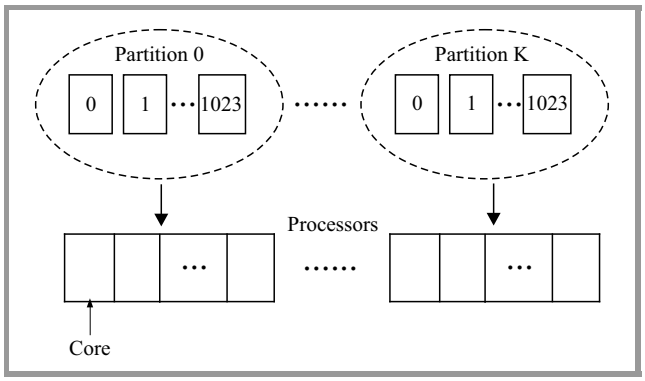

Fig. 5. Allocation of neurons to partitions and processors.

ory transfers on GPU are much faster if variables that are accessed at the same timestep are stored contiguously.
To maximize the speed of SNNS the particular attention was paid on the design of efficient data structures. The implementation is as follows. In each simulation experiment a population of neurons with unique identifiers is divided into partitions. To reduce the competition for memory access each partition is assigned to one GPU processor (see Fig. 5). Moreover, all synapses are divided into separated groups. Synapses with the same presynaptic neuron are collected to one group. Synapses from the same group are aggregated into the packages of fixed size and propagated across the network. Such an implementation allows to reduce memory usage.

5. SNNS Numerical Evaluation

The SNNS framework was used to simulate spiking neural networks with various size (1000 to 30000 neurons). Simulation experiments were conducted for following models:

- Network I&F – Integrate-and-fire model, spike average frequency 7 Hz.
- Network I – Izhikevich model, spike average frequency 15-25 Hz.
- Network H&H – Hodgkin-Huxley model, spike average frequency 15–30 Hz.

The experiments were performed on the following hardware platforms:

- P1: Intel Core2 Quad 2.83 GHz, Radeon HD 6700, 4 GB RAM, Linux x64.
- P2: Intel Core i5-2500K, 3.30 GHz, Radeon HD 6900, Linux x64.
- P3: Intel Core i5-2500K, 3.30 GHz, GeForce GTX 560T, Linux x64.

The results of simulations, i.e. times of calculations performed for various size of networks are presented in Figs. 6–12.

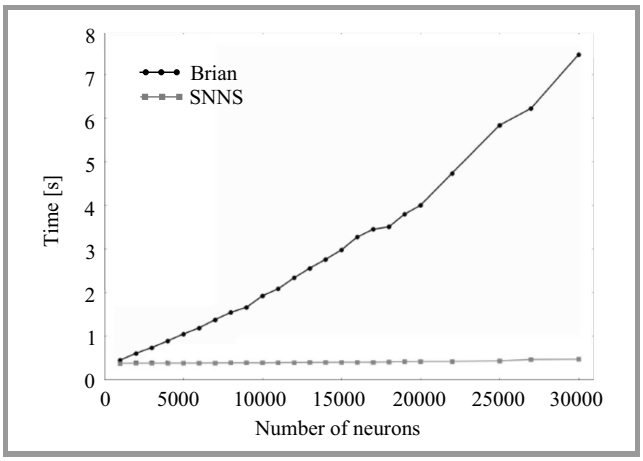

Fig. 6. Simulation time for Brian and SNNS simulators (Network I&F).

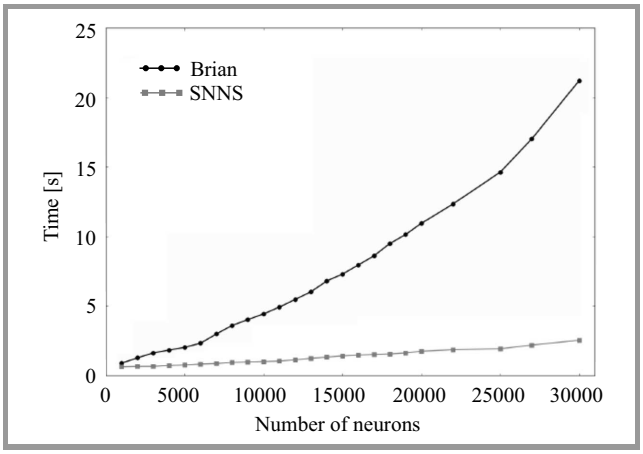

Fig. 7. Simulation time for Brian and SNNS simulators (Network I).

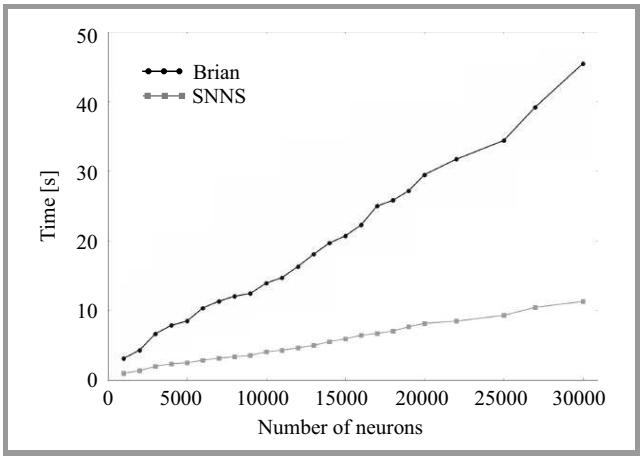

Fig. 8. Simulation time for Brian and SNNS simulators (Network H&H).

5.1. Comparative Study of CPU and GPU Simulators

The aim of the first series of experiments was to compare the efficiency of spiking neural networks simulation on CPU and GPU processors. The performance of the

CPU-enabled Brian simulator and the GPU-enabled SNNS framework was evaluated and compared. The experiments were performed on P1 and P2 hardware platforms. The results obtained for three neuron models are presented in Figs. 6–8.

The presented results show that the speed of simulation strongly depends on the network size and neuron model considered. The usage of GPU enabled speed up the simulation from 4 times for Network H&H, 9 times for Network H&H to 19 times for Network I&F.

5.2. Comparative Study of GPU Simulators

The aim of the next series of experiments was to compare the efficiency of spiking neural network simulation conducted on GPs from different vendors. First, two hardware platforms P1 and P2 with different computing power were tested. The results obtained for I&F and I neuron models are presented in Figs. 9–10.

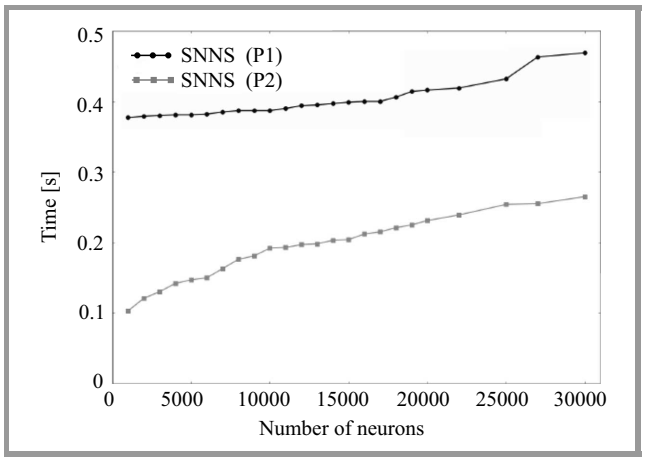

Fig. 9. Simulation time for SNNS simulator, P1 and P2 platforms (Network I&F).

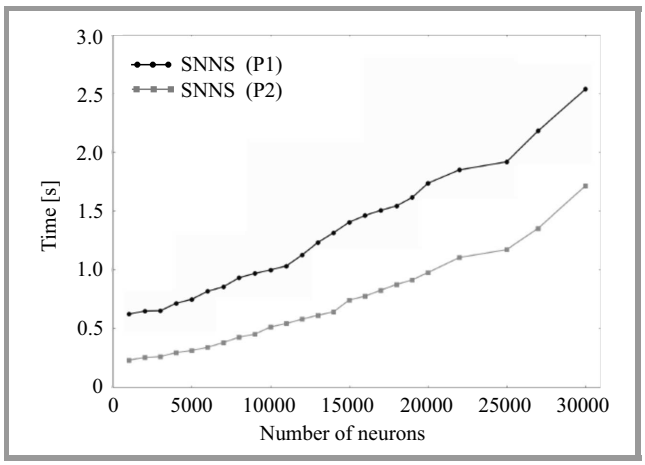

Fig. 10. Simulation time for SNNS simulator, P1 and P2 platforms (Network I).

The usage of more powerful GPU device enabled to speed up the simulation from 2.7 (Network I) to 3.7 (Network I&F) times. The acceleration was decreased with the size of the network.

Finally, the SNNS simulator was compared with the NeMo [21] CUDA-enabled framework for large scale networks simulation. Two series of experiments were performed for Network I&F and Network I. The tests for SNNS simulator were conducted on the P2 platform equipped with the AMD graphical processor Radeon HD 6900. The NeMo was executed on the P3 platform equipped with the NVIDIA graphical processor GeForce GTX 560T. The results of simulation experiments, i.e. times of calculations are depicted in Figs. 11 and 12.

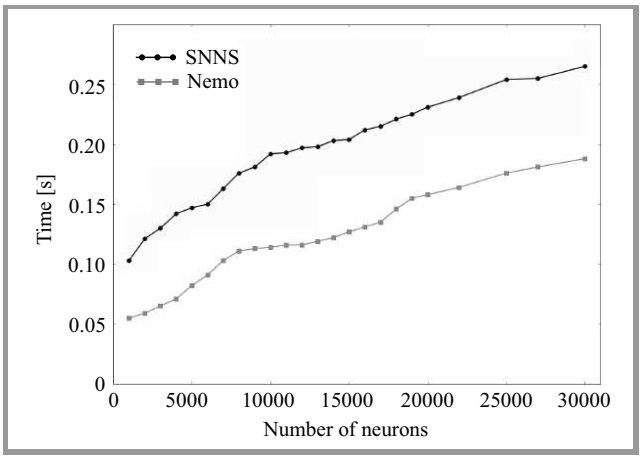

Fig. 11. Simulation time for NeMo and SNNS simulators (Network I&F).

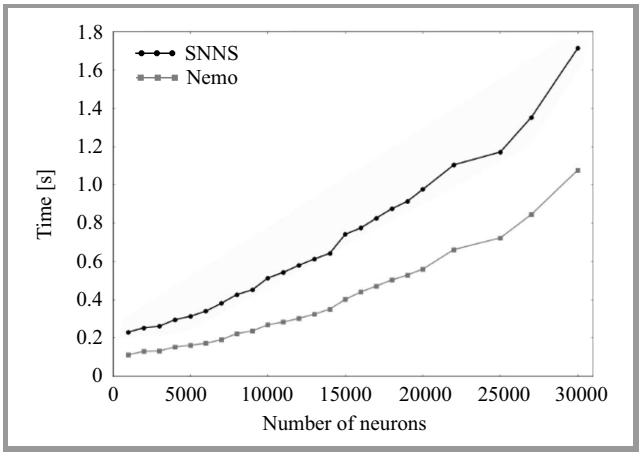

Fig. 12. Simulation time for NeMo and SNNS simulators (Network I).

It is observed that the CUDA-enabled simulator running on NVIDIA hardware gave a better results than the OpenCL one. The simulation time of neural network formed by 1000 neurons performed using the SNNS framework was decreased about 2 times for the NeMo framework and NVIDIA GPU. Such result was expected – CUDA is the technology dedicated to NVIDIA GPU. However, the numerical experiments showed that the acceleration level

with respect to OpenCL and AMD GPU decreases with the bigger size of a network.

6. Summary and Conclusion

The paper provides a short overview of methods and tools for parallel spiking neural networks simulation on GPU accelerators. Spiking neural networks are natural candidates for massively parallel computations. SNN simulation requires complex calculations and parallel processing of large volumes of data, in which a speed of calculation and data decomposition are of essence. The attention of the paper is focused on the OpenCL and GPU-enabled software framework SNNS for simulating large scale networks. SNNS was designed to be powerful, effective, scalable, flexible, and easy to use. The experimental results presented in this paper demonstrate the effectiveness of the SNNS framework, and confirm that the direction to speed up complex systems simulation is to port it to GPU units.
As a final observation one can say that CUDA and OpenCL computing systems offer a new opportunity to increase the performance of parallel HPC applications in clusters, by combining traditional CPU and general purpose GPU devices. However, although much progress has been made in software and hardware for HPC computing simulation of large-scale neurobiologically inspired systems is still a challenging task.

Acknowledgment

The author would like to thank Dr. Paweł Wawrzyński for assistance with this research.

References

[1] W.-M. W. Hwu (Ed.), *GPU Computing Gems Emerald Edition*, 1st ed. Morgan Kaufman, 2011.

[2] N. Carnevale and M. Hines, *The NEURON Book*. Cambridge University Press, 2006.

[3] W. Gerstner and W. Kistler, *Spiking Neuron Models*. Cambridge University Press, 2002.

[4] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press Cambridge, 2007.

[5] J. Vreeken, "Spiking neural networks: An introduction", Tech. Rep., Artificial Intelligence laboratory, Intelligent Systems Group, University of Utrecht, 2003.

[6] R. Brette *et al.*, "Simulation of networks of spiking neurons: A review of tools and strategies", *J. Computat. Neuroscience*, vol. 23, no. 3, pp. 349–398, 2007.

[7] K. D. Carlson, J. M. Nageswaran, N. Dutt, and J. L. Krichmar, "An efficient automated parameter tuning framework for spiking neural networks", *Front. in Neuroscience*, vol. 8, art. 10, pp. 1–16, 2014 [Online]. Available: http://dx.doi.org/10.3389/fnins.2014.00010

[8] Z. Fountas and M. Shanahan, "GPU-based fast parameter optimization for phenomenological spiking neural models", in *Proc. Int. Joint Conf. Neural Netw. IJCNN 2015*, Killarney, Ireland, 2015, pp. 1–8.

[9] E. M. Izhikevich, "Which model to use for cortical spiking neurons", *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, 2004.

[10] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *J. Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[11] E. M. Izhikevich, "Simple model of spiking neurons", *IEEE Trans. Neural Netw.*, vo. 14, no. 6, pp. 1569–1572, 2003.

[12] R. Brette and D. F. Goodman, "Simulating spiking neural networks on GPU", *Network*, vol. 23, no. 4, pp. 167–182, 2012.

[13] M. Chessa, V. Bianchi, M. Zampetti, S. P. Sabatini, and F. Solari, "Real-time simulation of large-scale neural architectures for visual features computation based on GPU", *Network*, vol. 23, no. 4, pp. 272–291, 2012.

[14] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors", *Neural Netw.*, vol. 22, no. 5–6, pp. 79–800, 2009.

[15] R. Ananthanarayanan and D. S. Modha, "Anatomy of a cortical simulator", in *Proc. of ACM/IEEE Conf. Supercomput. SC'07*, Reno, NV, USA, 2007, pp. 1–12 (doi: 10.1145/1362622.1362627).

[16] NEURON Simulator [Online]. Available: http://www.neuron.yale.edu/neuron/

[17] NEST Simulator [Online]. Available: http://www.nest-initiative.org/

[18] D. Goodman and R. Brette, "The Brian simulator", *Front. in Neuroscience*, vol. 3, no. 2, pp. 192–197, 2009.

[19] Brian Simulator [Online]. Available: http://briansimulator.org/

[20] Mvaspike Simulator [Online]. Available: http://mvaspike.gforge.inria.fr/

[21] NeMo Simulator [Online]. Available: http://nemosim.sourceforge.net/

[22] A. Fidjeland, E. Roesch, M. Shanahan, and W. Luk, "NeMo: a platform for neural modelling of spiking neurons using GPUs", in *20th IEEE Int. Conf. Application-specific Syst., Architec. & Processors ASAP 2009*, Boston, MA, USA, 2009.

[23] GeNN Simulator [Online]. Available: http:genn-team.github.io/genn/

[24] Myriad Simulator [Online]. Available: http://cplab.net/myriad/

[25] OpenCL – The open standard for parallel programming of heterogeneous systems [Online]. Available: http://www.khronos.org/opencl/

[26] E. Bainville, "OpenCL multiprecision tutorial", Jan. 2010 [Online]. Available: http:// http://www.bealto.com/mp-opencl.html

[27] R. R. D. Stewart and W. Bair, "Spiking neural network simulation: numerical integration with the Parker-Sochacki method", *J. Computat. Neuroscience*, vol. 27, pp. 115–133, 2009 (doi: 10.1007/S10827-008-0131-5).

Paweł Szynkiewicz received his M.Sc. in Computer Science from the Warsaw University of Technology, Poland, in 2015. Currently he is a Ph.D. student in the Systems Research Institute, Polish Academy of Science. In 2015–2016 employed in Comarch. Since 2016 he is with BrightSolutions IT. His research area focuses on software technologies, HPC computing, neural networks and machine learning, genetic algorithms, computer networks security.
E-mail: pszynk@gmail.com
Systems Research Institute
Polish Academy of Science
Newelska st 6
01-447 Warsaw, Poland