A novel simulator for extended Hodgkin-Huxley neural networks

Sotirios Panagiotou*, Rene Miedema†, Harry Sidiropoulos†,
George Smaragdos†, Christos Strydis† and Dimitrios Soudris *
* MicroLab, School of Electrical and Computer Engineering, National Technical University of Athens
Athens, Greece
Email: [spanagiotou, dsoudris]@microlab.ntua.gr
† Neuroscience Department, Erasmus Medical Center
Rotterdam, The Netherlands
Email: [r.miedema, c.sidiropoulos, g.smaragdos, c.strydis]@erasmusmc.nl

***Abstract*—Computational neuroscience aims to investigate and explain the behaviour and functions of neural structures, through mathematical models. Due to the models' complexity, they can only be explored through computer simulation. Modern research in this field is increasingly adopting large networks of neurons, and diverse, physiologically-detailed neuron models, based on the extended Hodgkin-Huxley (eHH) formalism. However, existing eHH simulators either support highly specific neuron models, or they provide low computational performance, making model exploration costly in time and effort. This work introduces a simulator for extended Hodgkin-Huxley neural networks, on multiprocessing platforms. This simulator supports a broad range of neuron models, while still providing high performance. Simulator performance is evaluated against varying neuron complexity parameters, network size and density, and thread-level parallelism. Results indicate performance is within existing literature for single-model eHH codes, and scales well for large CPU core counts. Ultimately, this application combines model flexibility with high performance, and can serve as a new tool in computational neuroscience.**

***Index Terms*—Biological neural networks, extended Hodgkin-Huxley neuron model, gap junctions, parallel processing, OpenMP, electrophysiology, in silico medicine**

I. Introduction

Computational neuroscience studies how neural structures demonstrate systemic and computational aspects. Advances in this field have yielded innovations in neurophysiology, brain function and interfacing, and artificial intelligence.

Exploring large neural networks with rich, conductance-level models, like the extended Hodgkin-Huxley (eHH) formulation, is increasingly gaining traction in brain research. This turn toward increasingly more sophisticated models is due to the superior simulation quality they offer. Most importantly, real neurons exhibit subtle features that radically affect network activity, such as spike synchronization and subthreshold oscillations. Simplified models often cannot express such features, due to their inherent assumptions. Additionally, the physiological quality of simulation results greatly facilitates model validation through experimental data, such as fluorescence imaging. Another benefit of physiological modelling is that simulation results are directly associated with the biological processes taking place. This way, off-nominal physiological factors, such as drug effects or medical conditions, can be explored in silico without having to design new neuron models.

Mathematical modelling of neuron physiology is an open question, due to the extent and complexity of the underlying biochemical processes, and the difficulties in inspecting these processes as they happen. Current research attempts to eliminate this knowledge gap, by proposing new models for these processes, indirectly determining their parameters through experiments, and augmenting the existing neuron models with the resulting model components. In silico simulation is widely used in the research process, in order to constrain the parameters being searched and investigate model limitations, in turn making wet-lab experimentation more efficient.

The neuroscientific methods described above present an important requirement for neural simulation tools; to support an extensible, general class of models, instead of supporting only a particular neuron model or brain region. This way, the value, usability and long-term relevance of the tools can be maintained, as the cutting-edge models evolve with new components and dynamics[1].

Simulation of neuron models is a computationally expensive task, due to the non-linear functions and the amount of parameters and state variables involved.This computational load becomes even greater when networks of multiple neurons are considered, and computational effort is also added for each synapse present in the network. The scale of the computational load, combined with the models' inherent parallelization opportunities and high scientific relevance, makes simulation of biophysically-detailed neural networks an appropriate application for high-performance computing.

The immense known and unknown complexity of the human brain has motivated researchers to adopt various simplifications when modelling neural structures. The most common simplification is to ignore spatial variability of neurons and model them as single-compartment entities, with all activity taking place at an abstract point in space; models following this technique are called *point neuron models*, and almost all neural network simulators support only variations of this class of models.

2471-7819/20/$31.00 ©2020 IEEE
DOI 10.1109/BIBE50027.2020.00071

One simplification approach is to replace the (often obscure) physiological mechanisms with abstract, low-dimensional dynamical systems that mimic neuronal activity. Common examples are the Morris-Lecar[2] and Hindmarsh–Rose[3] models.

Considering that the action potential is the most salient part of neural activity, another simplification path can be taken; the continuous-time behaviour of a neuron can be reduced to a series of discrete spiking events. This way, the object of simulation is just the timing between each neuron's consecutive action potentials, and communication between neurons happens only when action potentials are generated and transmitted. However, this approach eschews electrical synapses, which effect continuous-time interaction between neurons.

Simplification of spiking activity can also be combined with simplified neuronal dynamics, to produce even less computationally demanding models; the GLIF and Izhikevich[4] models are very prominent examples of this approach.

In this paper, we present a new simulator for conductance-based eHH, multicompartmental neural network models (called GenEHH from now on), that combines model flexibility with high computational performance. The simulator receives a human-readable description of the experiment to simulate, and runs the requested simulation, effectively utilizing multicore CPU resources.

The paper is organized as follows: In Section II, an overview of neuron models and simulation implementations for large-network simulation is presented. In Section III, our simulator's features are compared to existing conductance-based model simulators . In Section IV, we present our simulator's computational workload, the software architecture, and the parallelization techniques applied. In Section V, we present the experiments conducted to evaluate the simulator's computational performance and scalability using generic server-grade resources, and discuss the resulting performance data and the effect of computer hardware on performance. Finally, Section VI contains an overview of our work, experimental results, and the work's relevance to neuroscience research.

II. Related work

Many high-performance computing projects have produced neural network simulators; the most common runtime platforms are the versatile and ubiquitous x64 CPUs, and high-throughput but relatively cumbersome CUDA GPUs.

A Beowulf cluster of commodity general purpose computers has been used for simulations based on anatomical data, from slice scans of rat, cat and human brain tissue, containing 22 different types of cells as they are laminated on cortical columns, and spontaneously expressing behavioural regimes of normal brain activity.[6] However, the compartments of each modelled cell followed Izhikevich instead of conductance-based dynamics, and the implementation took tens of days to simulate a few seconds using the cluster, which is a prohibitive run time for research. In the desktop computer scale, a highly efficient GPU-based simulator is available for Izhikevich neuron models with plastic synapses[7].

In [8] authors simulated on a GPU a basic Hodgkin-Huxley point neuron model. The performance impact of electrical synapses was mitigated through a slow rate of voltage updates between neurons, and accelerating resting-state neuron simulation through telescopic projective integration, combining small transient timesteps with longer ones.

Simulation performance of Inferior Olive model cells with arbitrary gap junction connectivity has been investigated on clusters of Xeon Phi-accelerated nodes, as well as specialized architectures like Xeon Phi [5], GPUs and reconfigurable dataflow engines[9]. Those cells were multicompartmental cells and not point neurons.

Since performance requirements mandate extremely fast interaction between simulated neurons, many designs have also been tried out on bespoke, low-latency interconnection hardware. The SpiNNaker project[10] has developed a massive-scale ASIC-based neuromorphic architecture, where ARM processor cores send spike events each other through asynchronous digital logic, providing quick, broad and low-power exchange of spike event messages. Neuron dynamics can be customized through processor code, but still only spike-based, point-neuron models can be supported. Another class of ASIC implementations combines analog operation of single, simplified neurons, with electrical synapses between neurons in the same silicon die, and digital spike message passing between dies[11]. The analog implementation means only simplified Adaptive Exponential neuron models are supported.

Another approach to neural net simulation is extremely parametrizable, general purpose neural simulation software. Such packages provide simulation capability for an as diverse as possible range of neural models. This way, researchers can run simulations on the most detailed and highest quality neural models available, and propose new extensions to the existing types of models, without the need to create a new software or hardware application from scratch.

Common such packages are NEURON, NEST, BRIAN, MOOSE, and GENESIS. Many of these packages have been extended to support large-scale computer systems. However a lot of the parallelization burden is passed to the model designers, making simulation of a large network a whole project in itself, and precluding rapid model prototyping.

As shown in this section there is high trade-off between performance and level of model detail in the domain of neuronal simulators. The research works that focus on acceleration and high performance, support simulation of only simplified models, while the more generic, parametrizable simulators lack performance. In the following sections we will present our solution that bridges that gap.

III. Contribution

The neural network simulator introduced in this paper runs extended Hodgkin-Huxley (eHH) multicompartmental neuron models, simulates continuous-time gap junction interaction between neurons, supports a customizable set of compartments and a variable amount of highly customizable ion channels for

TABLE I
AN INDICATIVE COMPARISON OF SIMULATORS SUPPORTING CONDUCTANCE-BASED NEURAL NETWORKS.

	Simulator			
	NEURON	NEST	Chatzikonstantis et al. [5]	GenEHH
Model support				
Basic Hodgkin-Huxley dynamics	✓	✓	✓	✓
HH dynamics + Gap junctions (GJ)	✓		✓	✓
HH dynamics + Custom ion channels (CC)	✓	✓	✓	✓
HH dynamics + GJ + CC + Multicompartment model	✓			✓
Performance				
Automatic parallelization		✓	✓	✓
Multi-node parallelization		✓	✓	✓
Scalable performance		✓		✓

each compartments, and parallelizes the simulation workload across multiple processor cores.

As seen in the previous section, research has focused in simplified neuron models that are spike message-passing models, which require significantly less communication traffic than continuous-time interaction models with gap junctions.

To assess the existing simulation landscape, a brief representative set of conductance-based neural network simulators and supported features is compared in Table I. Each simulator is compared regarding the model types it supports: support for the classical Hodgkin-Huxley (HH) model, support for gap junction interaction, support for custom ion channel dynamics extending the HH model, and support for extensible multicompartmental models. In addition, the capabilities of each simulator to scale with available parallel resources are compared; namely, automatic distribution of computations to parallel threads, architectural support for parallelization in multi-node clusters, and scalability as model. Our simulator's scalability against parallel threads and network complexity has been confirmed through performance experiments, as described below. The system architecture also makes it easy to support a multi-node extension.

IV. PROPOSED SOLUTION

A. *Multicompartmental neuron and network modelling*

The GenEHH simulator assumes the compartmental level of neuron modelling. This level assumes that neural tissue consists of individual cells, and these cells are made up of distinct anatomical features(called compartments), across which the cell matter varies little.

Some anatomical features may be lumped into a single compartments, as long as simulation results remain accurate, to simplify analysis and reduce computational load. The dynamics of the internal state of each compartment are then explicitly defined directly through first-order ordinary differential equations; for each state variable present in the model, the corresponding rate of change is a directly evaluable function. In the case of coarse or lumped compartments, the models are empirical, and the model's coefficients are extracted by fitting against experimental data. In finer decompositions, the tubular parts of neurons are modelled after the cable equation[12], typically using its first-order approximation for discrete compartments. The active mechanisms present in neuron compartments are modeled after the eHH model[13].

Each neuron is also modelled to interact with adjacent neurons, through electrical synapses(also called electrical gap junctions). These are physical connections between adjacent neurons, adding a continuous, reciprocal interaction between the connected neurons. The intensity of this interaction can be described in terms of maximum electrical conductance between the two connected neuronal compartments.

Electrical flow through the cell's membrane passes through current probes, passive leaks, ion channels, and electrical synapses:

1) Current probes inject artificial current into cells, to induce or inhibit action potentials, or to simulate synaptic input from brain regions not included in the model.
2) Passive leaks behave equivalently to an ohmic leak between the membrane's two sides, in series with a reversal potential caused by the connection's molecular properties and polarity.
3) Each ion channel is modelled as a voltage source in series with a linear, time-varying resistor. Effective ion channel conductance is then a fixed maximum conductance coefficient, multiplied by gate variables as previously described.
4) Electrical synapse current is generally an antisymmetric, non-linear function of the local voltage difference between the connected cells, scaled by a conductance factor described above.

There are many other types of synapses also present in neural tissue; we chose to model specifically electrical synapses in the present simulator version. This is because electrical synapses tightly couple the behaviour of connected neurons, through continuous-time interaction; the continuous information transfer this interaction entails makes them by far the most computationally expensive synapse type. Also, simulation of event-based synapses has already been extensively studied and optimized in the existing literature; however, few works have tackled the computational issues of electrical synapses / gap junctions, and those that did either used fixed neuron models, or require the user to explicitly handle the parallelization of the simulation[14].

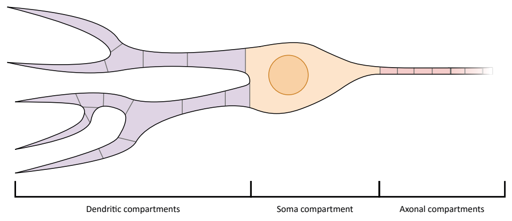


Fig. 1. A compartmental neuron decomposition example. A fine segmentation is denoted by the thin lines inside the neuron body, while the coarser parts of the neuron are coloured differently.

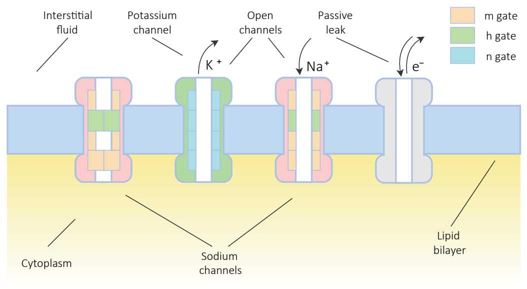


Fig. 2. Active and passive channels on a cell membrane and the effect of gates.

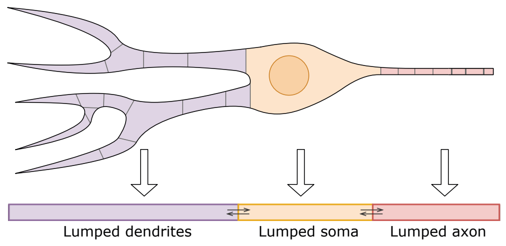


Fig. 3. A simplified catenary neuron model, supported by the simulator.

B. Computational model

The GenEHH simulator implements a certain subset of multicompartmental, eHH models. Some concessions were made due to the enormous variety of possible models, but the simulator implements a set of features that is common in literature, and captures the performance challenges this computational workload presents.

Each compartment's state is modelled as a set of scalar variables; its intracellular electrical potential, the concentrations of ions and other molecules that affect the cell's dynamics, and the gate variables of each ion channel.

The neural net model consists of a population of interconnected neurons. The network's neurons may all share the same neuron model, to quickly design an uniform population network. Alternatively, each neuron can be modelled differently. In this case, the model of each individual neuron is explicitly stated in the input files.

Each neuron is currently modelled in the simulator as a linear chain of interconnected cell compartments (Figure 1), each representing a portion of the cell. For example, dendritic tubes of a similar diameter may be lumped together (Figure 3), flattening the dendritic tree to a cylinder chain, as described by Rall [15]. A simple example is the commonly used ball-and-stick neuron model, that segments a neuron into the soma part, where most complex chemical processes take place, and the dendrites part, which roughly models the kinetics of all stimuli received and emitted by the cell.

Each ion channel gate has a set of parameters describing the behaviour of its activation variable. A gate variable may be dynamic, or static (Figure 2). A dynamic gate variable follows Hodgkin-Huxley dynamics: the variable's rate of change is a function of the variable's value and the compartment's membrane potential. Calcium concentration may also affect gate dynamics, to support the calcium activated ion channels common in literature. A static gate variable is a function of instant membrane potential, presenting no memory effect.

Hodgkin-Huxley gate variable dynamics follow the alpha-beta or tau-inf models, where alpha and beta, or tau and inf are functions of membrane potential. Some types of these functions are very common in published ion channel models, such as exponential, linear-by-exponential, and sigmoid fucntions. A web-accessible list of ion channel dynamics models is available at [16] . A set of these common function types was included in the simulator. To evaluate the rate of change of each ion channel gate variable, the alpha and beta or tau and inf functions it involves are evaluated, based on the supported set of function types.

The simulator supports connectivity between neurons, in the form of electrical synapses between lumped dendritic edge compartments. Since the compartments containing the synapses are lumped and gap junctions work symmetrically, gap junction connectivity between cells in the network can be described as a symmetric connectivity matrix W, with each element representing total gap junction base conductance between each pair of cells. (No connectivity between a pair of cells is equivalent to zero base conductance.)

In addition, the simulator supports external stimulation of neurons in the network, in the form of DC pulses injected into any neuron's lumped dendritic compartment.

C. Program architecture

The simulator presented in this paper is a standalone application running simulations in non-interactive, 'batch' mode. The simulator's target platform is the multicore x64 computer architecture. This platform suits the application best, since:

(a) it is ubiquitous in workstations and HPC settings alike;

(b) provides solid performance for the simulator's compute-heavy, random-access computational load under a wide variety of network models;
(c) allows efficient simulation of heterogeneous network models with disparate cell types, and changing the neuron models with no reconfiguration cost;
(d) places the least restrictions on what type of models can be efficiently simulated, making it easy to extend model support in future versions.

The GenEHH simulator is implemented as a non-interactive program, reading input files describing a simulation and producing data files containing the results of the specified simulation. Specifically, input files provide the model of the neural net to be simulated: the neurons present and their initial state, the synapses connecting the neurons to each other, the stimulation each neuron receives externally, and experiment duration and timestep used for simulation. All parameters of the model are defined through human- and machine- readable JSON[17] files, whose structure follows the well known -to field experts- Hodgkin-Huxley formulation. They also provide auxiliary simulator configuration such as input and output file redirection, performance hints for the simulator core, etc. Output files contain the model's state variables for each simulated timestep, and meta-information about the simulation, such as run time and memory usage.

Once the internal simulation data structures are constructed, the behaviour of the simulated model is calculated over the specified experiment time duration, through successive discrete timesteps. As simulation proceeds, its results (i.e. state variable trajectories) are also concurrently streamed to the output data files. When simulation is finished, experiment metadata and performance metrics are also stored for further analysis.

D. *Computational workload*

The main computational effort for a GenEHH run lies in the iterative progression of the model's state, through successive timesteps. For each simulation iteration, model state progresses by a fixed timestep; for each state variable in the model (i.e. each state variable of each compartment of each neuron in the net), its rate of change is evaluated, and then the for the next timestep is extrapolated using the Forward Euler method.

For each neuron, as seen in Figure 4, the required calculations can be separated into two groups. The first group concerns each neuron's internal mechanisms and state. This group consists of currents between compartments of the same neuron, passive leaks, current probe inputs, and ion channel dynamics. The required computational effort scales linearly with the number of compartments and associated mechanisms. The second group involves interaction between neurons through electrical synapses; the required computational effort per timestep scaled linearly with the amount of electrical synapses present in the network.

Computations for each neuron's internal dynamics need to be performed for each neuron independently from each other's state. Internal neuron complexity is given by the model definition and, for phenomenological compartment decomposition, is independent from neural net size. So the computational load for internal neuron dynamics calculations scales with $O(N)$ for increasing net size N.

For a given population size N, the number of possible synapses is $\frac{N\cdot(N-1)}{2} = O(N^2)$. The number of realized synapses, under the fixed probability model, is a fraction D of total possible synapses, where $D \in [0,1]$ is equal to the probability each synapse may exist.

Thus, when a neuronal network has a fixed synaptic density, the total computational load scales with $O(D * N^2)$ for large neuron populations.

In realistic network models, the amount of synapses per neuron is generally considered to reach a maximum value; thus the network becomes increasingly sparser as the network grows beyond a certain size. Thus, in order to keep computational load and memory requirements scalable, the simulator uses a sparse representation for the synaptic connectivity matrix described above.

E. *Simulation code parallelization*

The GenEHH workflow described above was enhanced with OpenMP directives, in order to utilize the parallel processing ability of multiprocessor systems.

For two or more computational tasks to be simultaneously performed on separate threads, the calculations in each task must not depend on the other's results in any way. At the same time, each separate task assigned to be performed in parallel with others incurs computational and synchronization overhead. Therefore, these tasks must be many enough to minimize load imbalance, keeping all processors busy, but also few enough to keep the OpenMP scheduler's overhead negligible. For example, when a loop is parallelized, the OpenMP scheduler may assign whole contiguous blocks of iterations, as single tasks, to processors. If the iterations take roughly the same time, the workload is balanced and this will maximize system efficiency.

For these reasons, parallelization was performed across simulation of entire neurons in the net, as seen in Figure 4. This analysis is expected to be efficient, since:

- the computational effort to simulate one timestep for a neuron is much larger than OpenMP task overhead;
- in large neural networks, the amount of neurons is much larger than the amount of concurrent threads, and the networks exhibit self-similar topologies. Thus the per-neuron work can be distributed evenly among threads.

Due to the conductance-based, continuous-time modelling of electrical synapses, each neuron must communicate its voltage state to all the neurons it is connected to, in every single timestep.

From a graph-theory perspective, neural networks are tightly connected, even when the direct connectivity matrix is sparse; the graph diameter (i.e. the upper bound for the closest chain of neurons, connecting any two neurons) is characteristically low. For fixed-probability connectivity models common in literature, maximum graph diameter can be shown to converge to 2 for large population sizes. For more realistic[18], fixed-expected-degree models, graph diameter is expected double logarithmic, and at most logarithmic to population size[19]. Since the communication paths among any pair of neurons are so small, no performance gains are expected from independently advancing the simulation state for selected groups of neurons, in the same network.

In addition, the Hodgkin-Huxley dynamics are highly non-linear and chaotic, therefore the model is unsuitable for algorithms that can parallelize across time for non-stiff models. Therefore, in this implementation, all computations per timestep are interlocked with a global synchronization barrier between timesteps.

In case the simulation results buffer is about to fill, the I/O operation flushing the results to the file can be performed in parallel with the new timestep. This operation is specified as an independent sibling task to simulation of all neurons inside a timestep, to mitigate possible I/O and file system delays.

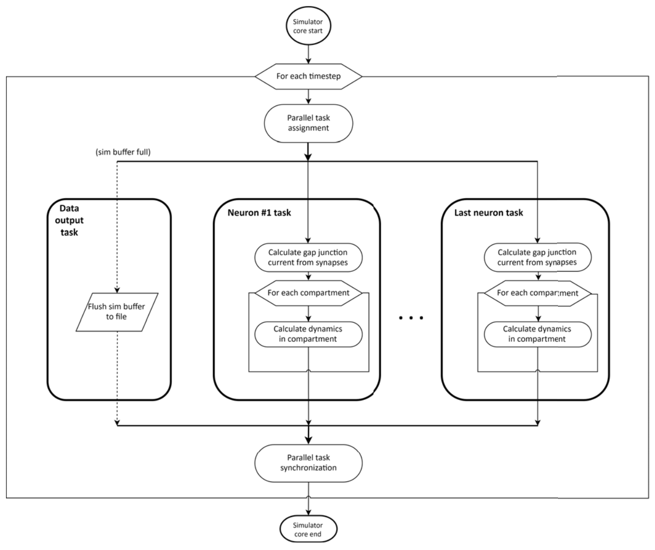


Fig. 4. The parallelized version of the simulator core.

V. Performance experimentation and evaluation

A. Experimental setup

To evaluate simulation performance under generic sever-grade resources, a set of runtime performance tests was run on multi-processor, virtual machine instances, allocated from the Amazon Elastic Cloud infrastructure.

Allocations were made on the `c5` tier of instances. This specific tier guarantees the class of CPU architecture physically running the virtual machine, meaning it offers guaranteed performance. The allocation sizes used were `c5.xlarge`, `c5.2xlarge`, `c5.4xlarge`, `c5.9xlarge`, `c5.18xlarge` instances, each with 4, 8, 16, 36, 72 available vCPUs respectively. The physical CPUs used for the simulations were Intel Xeon Platinum 8124M @ 3.0 GHz (18 cores, 36 threads each). All simulations were run to utilize all available vCPUs on each instance.

In all runs, performance was measured through the wall time required to run 10,000 simulation steps of the simulation. (Preliminary tests showed that data structure initialization had an insignificant impact on total run time.) We measured the core simulation time, and the internal matrix representation was set to sparse in all experiments, so that algorithmic performance could be directly compared. In order to investigate neural structure processing performance, synthetic truncated models were generated for the given compartments/neuron and ion channel gates/compartment parameters. A simulation was run and timed for each combination of the vCPU count, neuron population count, net density, compartments per neuron, ion channel gates per neuron parameters specified below:

- The simulation timestep was selected at 10 microseconds, to ensure model stability.
- Simulation time was 100 milliseconds (10,000 simulation steps) for all runs.
- The vCPU count is: 4, 8, 16, 36, 72 vCPUs running in parallel.
- Neuron counts considered are: 1000, 2000, 4000, 8000, 16000 neurons.
- Neuron connectivity densities considered are: 0%, 25%, 50%, 100% of total possible synapses.
- Neuron compartment counts considered are: 1, 2, 4, 8, 16 compartments per neuron.
- Compartment gate counts considered are: 1, 2, 4, 8 gates per compartment.

In total, 2000 simulation runs were performed to cover the explored parameter grid. Although we present results from homogeneous cell populations, if the neuronal network was composed of heterogeneous populations, the performance results apply also to these kind of populations.

B. Results and discussion

Due to the high volume and dimensionality of the parameter space explored, specific combinations of parameters will be presented, keeping the remaining parameters fixed. In most figures, internal neural complexity was set to maximum (16 compartments, 8 gates per neuron), since the simulator targets high-complexity models.

Figure 5 shows how run time increases with neuron population size, under various connection density factors. The results show clearly that run time increases quadratically with population size, as a naïve analysis of the computational workload would suggest. Under this type of analysis, which is also used in the algorithmic complexity and worst-case analysis domains, any fundamental calculation step takes constant time, regardless of the code path taken, cache misses, out-of-execution functional unit utilization, or other instruction execution context.

For an unconnected population, run time was measured to be linear to population size. That is also consistent with the type of analysis mentioned above, since in that case there are

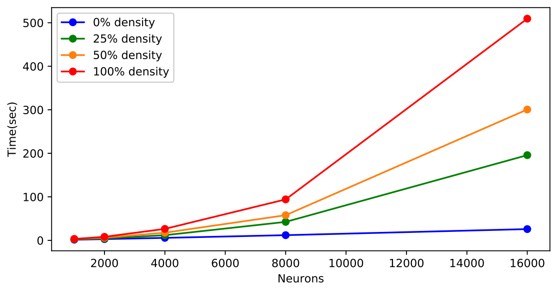


Fig. 5. Run time for 72 vCPUs, as a function of population size.

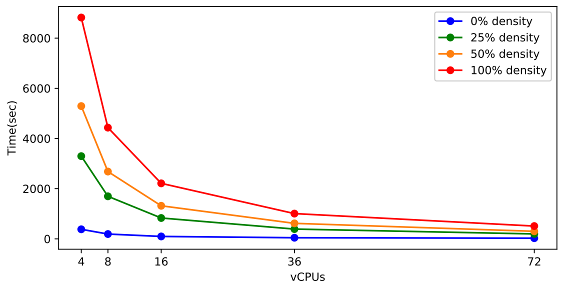


Fig. 6. Run time for a population size of 16000 neurons.

zero synapses, and computation is performed regarding each neuron's internal dynamics only.

Figure 6 shows how parallelization on a problem with maximum net population size scales ideally with the number of processors tested.

For larger population sizes, measured run time is inversely proportional to the number of vCPUs used, modulo a very small constant factor (typically less than 50 seconds, no correlation to model size or complexity). This means the simulator displays ideal strong scalability, up to the maximum problem size and number of processors tested.

The conditions for this ideal scalability under the per-neuron parallelization implemented in the GenEHH simulator requires that for each simulation timestep, the computational load of neurons is evenly distributed among the processors, so that all processors finish their assigned work at the same time and no time is wasted in processor idling. The even distribution of work per neuron is an effect of uniformly distributing synapses across neurons and uniformly distributing neurons among parallel processors. Thus statistically the workload is balanced among processors. In addition, the delay between start of a timestep and start of parallel processing (due to per-timestep result buffer management and OpenMP parallel section initiation) and the delay of the synchronization process have to be negligible.

Analysis of the experimental results showed that the relative

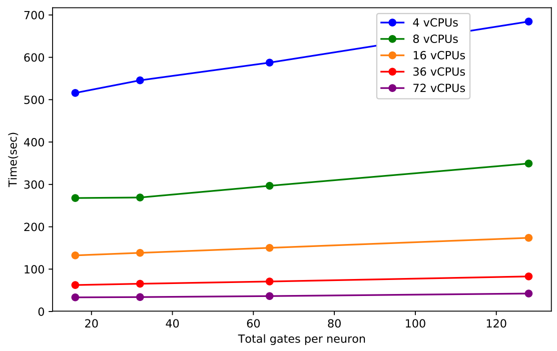


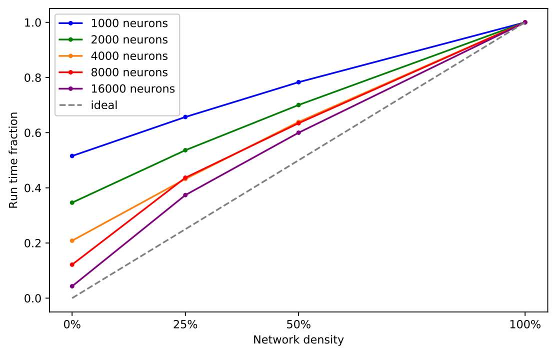
Fig. 7. Run time for 8000 neurons and 25% density, over total amounts of gates, for 8 gates per compartment.

effect of additional intra-neuron complexity on run time is an additive factor that is linear on the number of state variables added per cell. This is expected, since currently each variable depends on at most one other state variable for dynamics computations. Total run time remains inversely proportional to number of processors used. This is also expected, since the previous figures already show perfect distribution of per-neuron load, and no additional neuron interactions exist in the intra-neuron mechanisms. Figure 7 shows this linear scaling, for a representative neural net with maximum number of gates per compartment and varying the number of compartments per neuron and vCPUs in use.

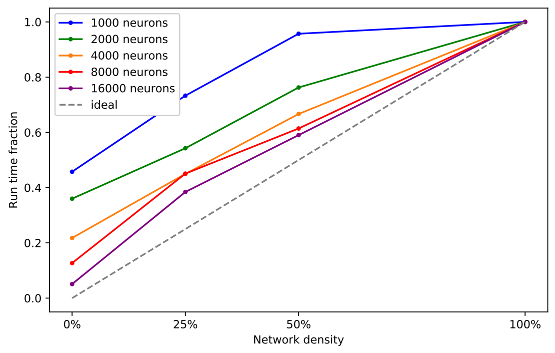
Figure 8 demonstrates the relation between network density and run time, under different amounts of thread-level parallelism. Run time is normalized against maximum, which naturally occurs at full synaptic connectivity. These figures demonstrate three distinct performance regions: density close to zero, density close to full, and low density . When connectivity is close to zero, efficiency increases as network size increases. This also expresses the proportion between synchronization overhead and useful work in each case. When connectivity is close to full, the simulator processes synapses more efficiently than under low network density . This could be explained by the cache and prefetch mechanisms of high-performance CPUs ; as density increases, there is a higher probability of following synapses already being in cache; in addition, at full density, the memory pattern becomes fully predictable, so automatic prefetching can optimize memory accesses. When connectivity is less than fifty percent, simulation efficiency decreases even more. This is to be expected since consecutive memory accesses are even less likely to share cache lines, but they are still not few enough for the lower amount of memory accesses to outweigh cache miss penalties.

VI. Conclusion

This paper has presented GenEHH, a highly-configurable, conductance-based neuronal network simulator, and discussed its computational performance on the multicore x64 platform.



(a) Relative run time for 4 vCPUs



(b) Relative run time for 72 vCPUs

Fig. 8. Normalized run time, over network density and population size, for maximum internal neuron complexity.

Use of the OpenMP parallelization library and generic C++ code allows the simulator to be easily targeted for other multiprocessor and/or manycore platforms.

The GeneHH simulator presented in this work is flexible enough to support a wide variety of detailed, biophysically accurate models of neuronal network. Each neuron can be independently modelled, with a rich set of biophysical features and a variable amount of discretized compartments and ion mechanisms for each neuron. At the same time, the simulator exhibits high performance, by demonstrating strong scalability in distributing the computational load to multiple threads, and executes the computational load with high efficiency.

An analysis of the computational load factors and their impact on run time was conducted. Then the hypotheses made were juxtaposed with the results of the performance experiments, highlighting the emergent effects of the model characteristics onto the GenEHH performance.

Acknowledgments

This research is supported by the European Commission Horizon2020 Framework Programme Project EXA2PRO (Grant Agreement No. 801015).

References

[1] G. T. Einevoll, A. Destexhe, M. Diesmann, S. Grün, V. Jirsa, M. de Kamps, M. Migliore, T. V. Ness, H. E. Plesser, and F. Schürmann, "The scientific case for brain simulations," *Neuron*, vol. 102, no. 4, pp. 735 – 744, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627319302909

[2] L. H. Morris C, "Voltage oscillations in the barnacle giant muscle fiber." *Biophys Journal*, vol. 35, pp. 193–213, 1981.

[3] J. Hindmarsh and R. Rose, "A model of neuronal bursting using three coupled first order differential equations," *Proceedings of the Royal Society of London. Series B, Biological sciences*, vol. 221, no. 1222, p. 87—102, March 1984.

[4] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[5] G. Chatzikonstantis, H. Sidiropoulos, C. Strydis, M. Negrello, G. Smaragdos, C. D. Zeeuw, and D. Soudris, "Multinode implementation of an extended hodgkin–huxley simulator," *Neurocomputing*, vol. 329, pp. 370 – 383, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231218312906

[6] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and 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*, Nov 2009, pp. 1–12.

[7] 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 Networks*, vol. 22, no. 5, pp. 791 – 800, 2009, advances in Neural Networks Research: IJCNN2009.

[8] M. Wang, B. Yan, J. Hu, and P. Li, "Simulation of large neuronal networks with biophysically accurate models on graphics processors," in *The 2011 International Joint Conference on Neural Networks*, July 2011, pp. 3184–3193.

[9] G. Smaragdos, G. Chatzikostantis, S. Nomikou, D. Rodopoulos, I. Sourdis, D. Soudris, C. I. D. Zeeuw, and C. Strydis, "Performance analysis of accelerated biophysically-meaningful neuron simulations," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 1–11.

[10] S. Kunkel, M. Schmidt, J. M. Eppler, H. E. Plesser, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, and M. Helias, "Spiking network simulation code for petascale computers," *Frontiers in Neuroinformatics*, vol. 8, p. 78, 2014. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2014.00078

[11] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 431–438.

[12] R. D. Traub, R. K. Wong, R. Miles, and H. Michelson, "A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances," *J. Neurophysiol.*, vol. 66, no. 2, pp. 635–650, Aug 1991.

[13] E. Lewis, "Neuroelectric potentials derived from an extended version of the hodgkin-huxley model," *Journal of Theoretical Biology*, vol. 10, no. 1, pp. 125 – 158, 1966.

[14] M. Migliore, C. Cannia, W. W. Lytton, H. Markram, and M. L. Hines, "Parallel network simulations with neuron," *Journal of Computational Neuroscience*, vol. 21, no. 2, p. 119, May 2006. [Online]. Available: https://doi.org/10.1007/s10827-006-7949-5

[15] W. Rall, "Electrophysiology of a dendritic neuron model," *Biophys. J.*, vol. 2, no. 2 Pt 2, pp. 145–167, Mar 1962.

[16] R. Ranjan, G. Khazen, L. Gambazzi, S. Ramaswamy, S. L. Hill, F. Schürmann, and H. Markram, "Channelpedia: An integrative and interactive database for ion channels," *Frontiers in Neuroinformatics*, vol. 5, 2011. [Online]. Available: https://doi.org/10.3389/fninf.2011.00036

[17] "The javascript object notation (json) data interchange format." [Online]. Available: https://tools.ietf.org/html/rfc8259

[18] E. Bullmore and O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems," *Nat. Rev. Neurosci.*, vol. 10, no. 3, pp. 186–198, Mar 2009.

[19] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Proceedings of the National Academy of Sciences*, vol. 99, no. 25, pp. 15 879–15 882, 2002. [Online]. Available: http://www.pnas.org/content/99/25/15879