Controlling Complexity of Cerebral Cortex Simulations—I: CxSystem, a Flexible Cortical Simulation Framework

Vafa Andalibi
vafandal@indiana.edu
Clinical Neurosciences, Neurology, University of Helsinki and Helsinki University Hospital, Helsinki 00029, Finland, and School of Informatics, Computing and Engineering, Indiana University Bloomington, IN 47408, U.S.A.

Henri Hokkanen
henri.hokkanen@helsinki.fi
Simo Vanni
simo.vanni@helsinki.fi
Clinical Neurosciences, Neurology, University of Helsinki and Helsinki University Hospital, Helsinki 00029, Finland

Simulation of the cerebral cortex requires a combination of extensive domain-specific knowledge and efficient software. However, when the complexity of the biological system is combined with that of the software, the likelihood of coding errors increases, which slows model adjustments. Moreover, few life scientists are familiar with software engineering and would benefit from simplicity in form of a high-level abstraction of the biological model.

Our primary aim was to build a scalable cortical simulation framework for personal computers. We isolated an adjustable part of the domain-specific knowledge from the software. Next, we designed a framework that reads the model parameters from comma-separated value files and creates the necessary code for Brian2 model simulation. This separation allows rapid exploration of complex cortical circuits while decreasing the likelihood of coding errors and automatically using efficient hardware devices.

Next, we tested the system on a simplified version of the neocortical microcircuit proposed by Markram and colleagues (2015). Our results indicate that the framework can efficiently perform simulations using Python, C++, and GPU devices. The most efficient device varied with computer hardware and the duration and scale of the simulated system. The speed of Brian2 was retained despite an overlying layer of software.

V.A. programmed the CxSystem and wrote the manuscript. H.H. contributed to programming the simulator and wrote the manuscript. S.V. conceived the study, contributed to programming the simulator, and wrote the manuscript.

However, the Python and C++ devices inherited the single core limitation of Brian2.

The CxSystem framework supports exploration of complex models on personal computers and thus has the potential to facilitate research on cortical networks and systems.

1 Introduction

1.1 Controlling Complexity. Numerical simulations of biological neural networks have a long and diverse history (Bower, 1992; Brette et al., 2007; Gerstner, Sprekeler, & Deco, 2012). Recently, the increase of computational power has enabled simulations of large-scale systems, which has highlighted the need to control the complexity of the simulation code. Some simulation engines abstract the neural system with different strategies: by adding a user-friendly GUI (Aisa, Mingus, & O'Reilly, 2008; Bekolay et al., 2014; Gleeson, Steuber, & Silver, 2007; Hines & Carnevale, 1997; Tosi & Yoshimi, 2016), creating a simple interface to a more complicated kernel (Davison et al., 2009; Eppler, 2009; Hoang, Tanna, Bray, Dascalu, & Harris, 2013), or simplifying the description and implementation of the neural network dynamics (Raikov et al., 2011). With these tools, large systems can be coded with a few lines of code and very little clutter. However, when we take the challenge of modeling cortical microcircuits or systems in a biologically meaningful way, we must go one step further.

The complexity of the cerebral cortex is palpable. First, the network is complex. For instance, one hemisphere of macaque monkey has on average $400 * 10^6$ nerve cells (Herculano-Houzel, Collins, Wong, Kaas, & Lent, 2008) in 130 to 140 functional areas (Van Essen, Glasser, Dierker, & Harwell, 2012) that are interconnected by about a 66% connection density spanning five orders of magnitude (Markov et al., 2011, 2014). Second, each neuron is complex and challenging to model realistically (Almog & Korngreen, 2016). Dendritic trees are complex in terms of both anatomy and biophysics (Larkum, Nevian, Sandler, Polsky, & Schiller, 2009; Rall, 1962; Sidiropoulou, Pissadaki, & Poirazi, 2006), and signal propagation is significantly affected by the dendritic morphology and ion channel distribution (Sjöström, Rancz, Roth, & Häusser, 2008). Third, the cerebral cortex is not homogeneous across cortical areas, developmental stages (Elston, Oga, Okamoto, & Fujita, 2010; Elston & Rosa, 1997), or species (Betizeau, Dehay, & Kennedy, 2014; Herculano-Houzel et al., 2008; Horvát et al., 2016). Finally, synapses and network homeostasis are heavily influenced by plasticity (Feldman, 2009; Turrigiano, 2008) and sleep (Rasch & Born, 2013).

When we started to model the cerebral cortex, we noticed that even a very good simulation engine resulted in a complexity nightmare, where a prohibitive model became mixed with a long and complex code. Complexity in software projects increases errors and prolongs development time, and managing complexity has become a primary technical imperative of software development (McConnell, 2004).

Here, we have come up with a simulation framework that simplifies numerical simulations by dividing the model construction into two parts. The first part addresses code complexity. First, the abstraction level of the biological model is decided. If the desired model contains elements that are not included in the framework by default, they need to be coded in the selected high-level simulation engine. Our approach implicitly mitigates code complexity since such programming tasks are isolated inside the framework.

In the second part, the framework completely hides the code and provides the user with two simple and restrictive interfaces. The simple user interfaces support experimentation with complex biological systems and aim at reduced development time without coding errors. With appropriate filters, a life scientist can start from an existing complex model with minimal need to consider the software implementation.

1.2 Spreadsheet as a Development Environment. We implemented the idea of code/model division into CxSystem, a new framework that dynamically compiles a spreadsheet model into a simulation device and enables flexible biomimetic simulations of cortical systems. The framework works on the top of the Python-based Brian2 simulator (Goodman & Brette, 2009).

In Brian2 the user can implement membrane and synapse dynamics with straightforward model equations. Moreover, Brian2 inspects the units of the variables in the equation, reducing modeling errors. Brian2 provides the flexibility necessary for model exploration and is known to have the most compact code compared to other widely used simulation engines (Tikidji-Hamburyan, Narayana, Bozkus, & El-Ghazawi, 2017), such as NEST (Gewaltig & Diesmann, 2007) and NEURON (Hines & Carnevale, 1997). The CxSystem reuses components of earlier work (Heikkinen, Sharifian, Vigário, & Vanni, 2015), but the code was completely rewritten to be dynamically compiled from configuration files, to run on C++ and GeNN (Yavuz, Turner, & Nowotny, 2016) devices, and to support parallel parameter searches. CxSystem is available at GitHub[1] accompanied by online documentation.[2]

2 Methods

2.1 Framework

2.1.1 Interface of CxSystem. Two comma-separated values (CSV) files comprise the main user interfaces of the CxSystem. The first, the model and

[1] https://github.com/VisualNeuroscience-UH/CxSystem.
[2] https://cxsystem.readthedocs.io.

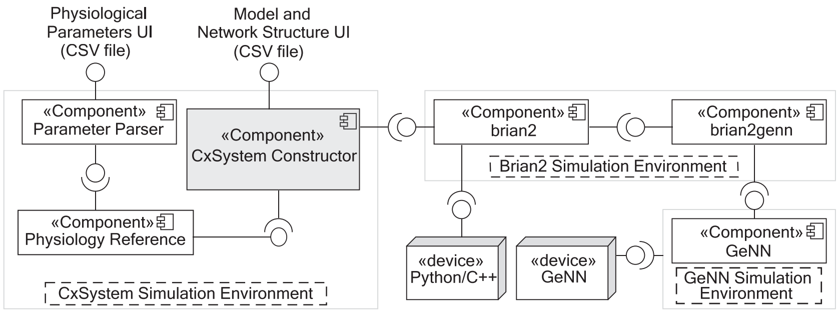
Physiological Parameters UI (CSV file)
Model and Network Structure UI (CSV file)
«Component» Parameter Parser
«Component» CxSystem Constructor
«Component» Physiology Reference
CxSystem Simulation Environment
«Component» brian2
«Component» brian2genn
Brian2 Simulation Environment
«device» Python/C++
«device» GeNN
«Component» GeNN
GeNN Simulation Environment

Figure 1: UML diagram of the CxSystem, illustrating the interaction of the CSV files, internal components of the CxSystem, Brian2 simulator, Brian2genn front end, and GeNN simulator.

network configuration file, houses the simulation setup parameters and the network-level anatomical structure of the model. This structure includes all connections between cell groups, as well as their connection probabilities and number of synapses and connection.

The second, the physiological configuration file, houses the spread of local connections, as well as all the neuron- and synapse-level biophysical parameters affecting the membrane voltage. Input to stimulate the system currently comprises the afferent volley of thalamocortical spikes with timing specified by the user.

2.1.2 Internal Components of the CxSystem. The CxSystem has three main components: the CxSystem constructor, the parameter parser, and the physiology reference (see Figure 1). The CxSystem Constructor reads the model and network configuration and builds up the anatomy. The parameter parser reads the physiological configuration file, extracting the physiological parameters. These are fed to the physiology reference, which collects all the elements that are required to create desired objects for Brian2.

Next, the physiology reference packs the required physiological parameters as a reference dictionary and passes it to the CxSystem constructor, the only component that interfaces with the Brian2 simulator.

C++ code generation is provided natively by Brian2, whereas compute unified device architecture (CUDA) code generation needs a GeNN simulator via the Brian2genn frontend (Nowotny et al., 2014). The CxSystem is built to cope with the minor limitations on Brian2 features that emerge from the stand-alone C++ and GeNN devices. The most important limitation is using a single network, the brian2 *magic* network, since multiple networks cannot be supported in the Brian2GeNN interface.

2.1.3 CxSystem Simulation Flow and Run Modes. When CxSystem runs a simulation, it executes the configurations in this order: network, input, neurons, and then synaptic parameters. The network parameters are set in the first step, preparing the environment for building up the neurons and synapses. Based on the input type, CxSystem prepares the network input either sequentially (e.g., vector of afferent action potentials) or in parallel (e.g., video through artificial retina). Next, the syntaxes for neuron groups, membrane equations, and corresponding monitors are dynamically formed and run. Similarly, the equations for synapses are formed based on the receptor type, compartments, and the pre- and postsynaptic neuron layers. CxSystem places the generated objects in the proper scope for Brian2. After running the simulation, CxSystem collects and saves the monitored data.

Simulations can be performed in two modes. In the local mode, the distance between the neurons does not affect the connection probabilities. In the expanded mode, the user can scale the connection probability with distance, by default with exponential decay (Markov et al., 2011, 2013).

The CxSystem can run simulations in parallel, in single-dimensional or multidimensional mode. In a single-dimensional array run, each parameter change is independent of another. In a multidimensional array run, CxSystem runs all combinations of all parameter dimensions. Both modes run each parameter set in one thread.

The GeNN device automatically distributes the load of a single run across the graphics card, thus computing one parameter set in parallel. However, only one simulation should be run with a single GPU, since multiple simulations would use the same resources, degrading performance.

When the CxSystem is run on a cluster (tested with SLURM workload manager in Taito cluster[3] at IT center for science in Finland), it automatically divides the workload based on the requested number of nodes, generates the proper batch job scripts, transfers the required files to the cluster connection node, submits the simulations remotely, and, when the results are ready, automatically downloads the results and cleans both local and remote environments.

2.1.4 Performance Evaluation. For both run-time and stand-alone code generation (for C++ and CUDA via GeNN), the CxSystem was run on two computers. The first was a laptop equipped with an Intel Core i7-4702MQ CPU, 8 GB of DDR3 memory, and NVIDIA GK208M (Geforce 740M GT) graphic card with 2 GB video memory running the Linux operating system on HDD. The second was a workstation equipped with Intel Xeon Processor (E5 2640–2.6 GHz), 128 GB of DDR4 memory, and one NVIDIA GK104GL (Quadro k4200) graphic card with 4 GB video memory running the Linux operating system on a solids state drive (SSD). The simulation time step was

[3] https://research.csc.fi/taito-supercluster.

0.1 ms. The same laptop was also used for the conductance-based Hodgkin-Huxley (COBAHH) benchmark. In the Taito cluster, the CxSystem was run on HP SL230s G8 nodes with 16 cores and 64 GB memory per node.

For Python and C++ stand-alone devices, we tested the CxSystem on Linux, Windows, and MacOS systems. The GeNN stand-alone device was tested on Linux and Windows.

2.2 Simplified Model of Neocortical Microcircuit. The model details are described in the companion letter in this issue. In short, we copied the number of cells (∼31,000) and synapses (∼37 million) from a recently published comprehensive rat somatosensory cortex microcircuit model (Markram model; Markram et al., 2015). We made three main simplifications. (1) The morphological cell types were reduced from 55 to 16. (2) Instead of high-fidelity multicompartmental modeling, we used only up to seven compartments in pyramidal cells and point-like stellate cells. (3) We used exponential integrate-and-fire models (Fourcaud-Trocmé, Hansel, van Vreeswijk, & Brunel, 2003) instead of Hodgkin-Huxley-type neuron models.

The first part of the model was fixed and was coded directly into the CxSystem. This part comprised synaptic and neural equations for pyramidal cell (PC), spiny stellate (SS), basket cell (BC), Martinotti cell (MC), and layer I inhibitory cell (L1I) groups. In the case of compartmental PC neurons, the equations were a template that was automatically expanded based on the layers in which apical dendrites reside. As for the synaptic models, the user could choose between fixed synapses and synapses with short-term synaptic plasticity (STP; Markram, Wang, & Tsodyks, 1998; Zucker & Regehr, 2002).

The second part of the model was read from the CSV files and interfaced into the CxSystem. The connections and synapse probabilities were imported from the Markram model, converted to a CxSystem-compatible CSV, and imported as a model and network configuration file.[4] The physiological and biophysical parameters were introduced to the CxSystem in the physiological file.[5] The values of these parameters for the Markram model are presented in Table 2 of the companion letter.

2.3 Porting/Building a Model. The first step to port an existing model to CxSystem is to define the target network and the cell group characteristics. The CxSystem currently includes multicompartmental PC neurons and four types of point neurons, which provide templates for potential new neuron types. Each of the neuron groups will take a single line in the model

[4] https://github.com/VisualNeuroscience-UH/CxSystem/tree/master/config_files/Markram_config_file.csv.

[5] https://github.com/VisualNeuroscience-UH/CxSystem/tree/master/config_files/Physiological_Parameters.csv.

and network file. The physiological parameters of the neurons should also be modified in the physiological file. In the next step, the synaptic connections between the neuron groups are determined. Finally, the simulation parameters are configured by defining the parameters such as simulation duration, device, and system mode. Existing configuration files within the repository provide a useful starting point for porting new models.

We ported the 4000-neuron COBAHH benchmark model (Brette et al., 2007), which is based on network model of Vogels and Abbott (2005). In native Brian2, the network can be implemented with only one neuron group, which is divided into separate excitatory and inhibitory subgroups. We implemented this with two groups: an excitatory neuron group containing 3200 and an inhibitory neuron group containing 800 HH cells. Hodgkin-Huxley-type equations were added to a physiological reference Python module with fixed ion channel activation-inactivation parameters. The connection probability between neurons inside a group and between the two neuron groups was 2%, with each connection comprising one synapse; these were defined in the model and network file (see Table 1). The physiological details were then applied in the physiological file (see Table 2) including synaptic parameters, connection weights, synaptic delays, and neuron group parameters.

A new network can be designed by using a regular spreadsheet program such as Excel. First, plan the system architecture and select neuron groups, sizes, and locations. Next, it is useful to draw a connection diagram before implementing the connections. Finally, check the biophysical parameters for the selected neuron groups and connection types. A new project might need functionalities we have not implemented. New features can be programmed in Python in the physiology reference module, and they can then be referenced in the configuration file.

3 Results

The performance of the CxSystem was evaluated in three ways. First, using the COBAHH model (Brette et al., 2007), the performance of CxSystem was compared with native Brian2. Second, we compared the performance of the three devices supported by the CxSystem: Python, C++, and GeNN. Finally, we tested weak scaling performance with several independent runs in a cluster. The latter two ran the simplified model of neocortical microcircuit.

3.1 Simulator Performance. We compared the total run time of three implementations of COBAHH example: in CxSystem and Brian2 with one and two neuron groups to get an approximation of the CxSystem overhead. The configuration file (see Table 1) along with the corresponding physiological configuration file (see Table 2) builds up and runs the COBAHH example in the C++ stand-alone device for 1000 ms.

Table 1: Tabular View of the Model and Network Configuration file for the COBAHH Example.

row_type	runtime	profiling	default_clock	do_init_vms	min_distance	scale
params	1000 * ms	1	0.1 * ms	1	1 * um	1
row_type	sys_mode	grid_radius	output_path_and_filename	device	number_of_processes	
params	local	210 * um	./output.gz	cpp	1	
row_type	idx	number_of_neurons	neuron_type	layer_idx	net_center	monitors
G	0	3200	HH_E	4	–	[Sp]
G	1	800	HH_I	4	–	[Sp]
row_type	receptor	pre_syn_idx	post_syn_idx	syn_type	p	n
S	ge	0	0	Fixed	0.02	1
S	ge	0	1	Fixed	0.02	1
S	gi	1	0	Fixed	0.02	1
S	gi	1	1	Fixed	0.02	1

Table 2: Tabular View of the Physiological Configuration File for the COBAHH Example.

Variable	Key	Value
Calcium_Concentration		2
Connection weights		
_weights	w_All_other_E-E_connections	6 * nS
	w_All_other_E-I_connections	6 * nS
	w_All_other_I-E_connections	11 * 6 * nS
	w_All_I-I_connections	11 * 6 * nS
cw	cw_HH_E_HH_E	_weights['w_All_other_E-E_connections']
	cw_HH_E_HH_I	_weights['w_All_other_E-I_connections']
	cw_HH_I_HH_E	_weights['w_All_other_I-E_connections']
	cw_HH_I_HH_I	_weights['w_All_I-I_connections']
Synaptic delays		
delay	delay_HH_E_HH_E	3.0 * ms
	delay_HH_E_HH_I	3.0 * ms
	delay_HH_I_HH_E	3.0 * ms
	delay_HH_I_HH_I	3.0 * ms
Neuron group parameters		
HH_E	C	200 * pF
	gL	10 * nS
	g_na	20. * usiemens
	g_kd	6. * usiemens
	ENa	50 * mV
	EK	−90 * mV
	taum_soma	C/gL
	EL	−60 * mV
	Vr	−60 * mV
	Vcut	20 * mV
	VT	−63 * mV
	V_res	−80 * mV
	Ee	0 * mV
	Ei	−80 * mV
	tau_e	5 * ms
	tau_i	10 * ms
HH_I	C	200 * pF
	gL	10 * nS
	g_na	20. * usiemens
	g_kd	6. * usiemens
	ENa	50 * mV
	EK	−90 * mV
	taum_soma	C/gL
	EL	−60 * mV
	Vr	−60 * mV
	Vcut	20 * mV

Table 2: Continued.

Variable	Key	Value
Calcium_Concentration		2
	VT	−63 * mV
	V_res	−80 * mV
	Ee	0 * mV
	Ei	−80 * mV
	tau_e	5 * ms
	tau_i	10 * ms

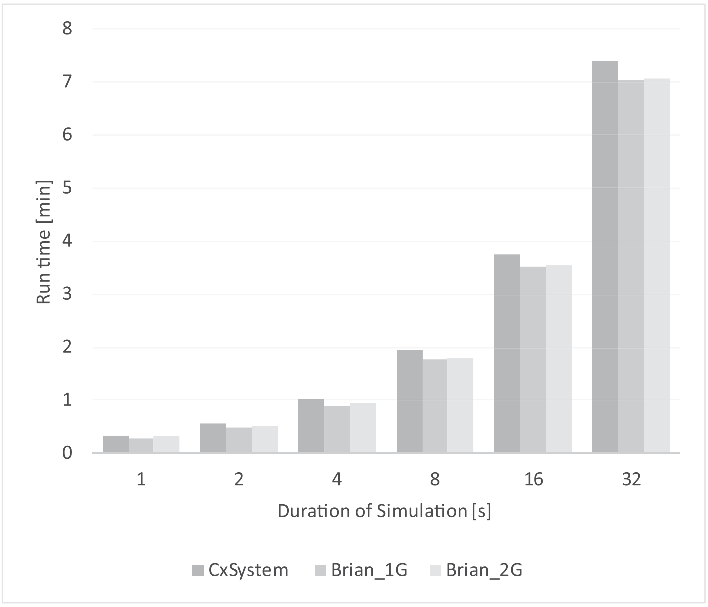

Figure 2: Proof-of-concept performance evaluation of the CxSystem running COBAHH example (Brette et al., 2007) in comparison with Brian2 implemented with one and two neuron groups.

The runtime of the single-core simulation of COBAHH example in CxSystem, as shown in Figure 2, is very close to Brian2 and the marginal overhead of the CxSystem over Brian2, i.e., the cost for simplicity, is negligible compared to total runtime.

3.2 Device-Level Performance Evaluation. We benchmarked the CxSystem using Python, C++, and GeNN devices in a laptop and workstation in two scenarios. First, the duration of run time was increased while the CxSystem was running in the local mode (see Figures 3A to 3D). Second, the size of the system was increased, so the CxSystem was running in expanded mode (see Figures 3E to 3H). We examined the integrity of C++ and GeNN code generation in comparison with the native Python. With an identical configuration file, neural positions, initial membrane voltages, and connections, the three implementations produced identical output.

For each of these scenarios, we measured two main metrics: Brian2 simulation run time and total computation time (including run time, Python code compilation, code generation performed by Brian2, and device-specific compilation for C++ and GeNN devices).

The first scenario simulated the simplified Markram model (related to as the EIF model in the companion letter) for durations ranging from 1 to 32 s. As expected, the run time was linearly associated with the simulation duration. In the laptop, the C++ device was always faster than the Python, and time saved with the C++ device increased with both run time and total computation time (see Figures 3A and 3B). The GeNN device followed a rather different trend. First (compare Figures 3A and 3B with Figures 3C and 3D), its performance was inextricably linked with the graphic card memory. The low-end GPU in the laptop with a low device memory demanded a higher number of memory transfers between host and device, which caused weak performance. In the workstation, the GeNN device performed best with long runs.

In the second scenario, the duration of simulations was set to 1 s, and the CxSystem was scaled from 0.5 to 4 (laptop) or 8 (workstation) times the Markram model size. Unsurprisingly, scaling the CxSystem up raised out-of-memory error much faster than increasing the duration of simulation (see the missing gray bars in Figures 3E to 3H). The out-of-memory error appeared first for the GeNN device and later for the Python or C++.

The Python and C++ outpaced the GeNN device, because GeNN suffered from too many device-host memory transfers. In contrast, reaching the out-of-memory state while using the Python and C++ device entirely depended on the host memory. With 128 GB of host memory on our workstation, we managed to scale our model up to 11 times the Markram model.

3.3 Parallel Scalability. Currently, single Brian2 simulations can be parallelized with GPU via a GeNN simulator. When exploring parameter spaces, the CxSystem supports independent simulations in different cores with array_run (workstation or laptop) and cluster_run (clusters) modes. In the future, CxSystem aims to follow Brian2 updates and implement possible new parallelization capabilities.

The two main approaches to measure the scalability of an application are weak and strong scaling tests, which present a measure of scalability for CPU-bound and memory-bound applications, respectively. In the

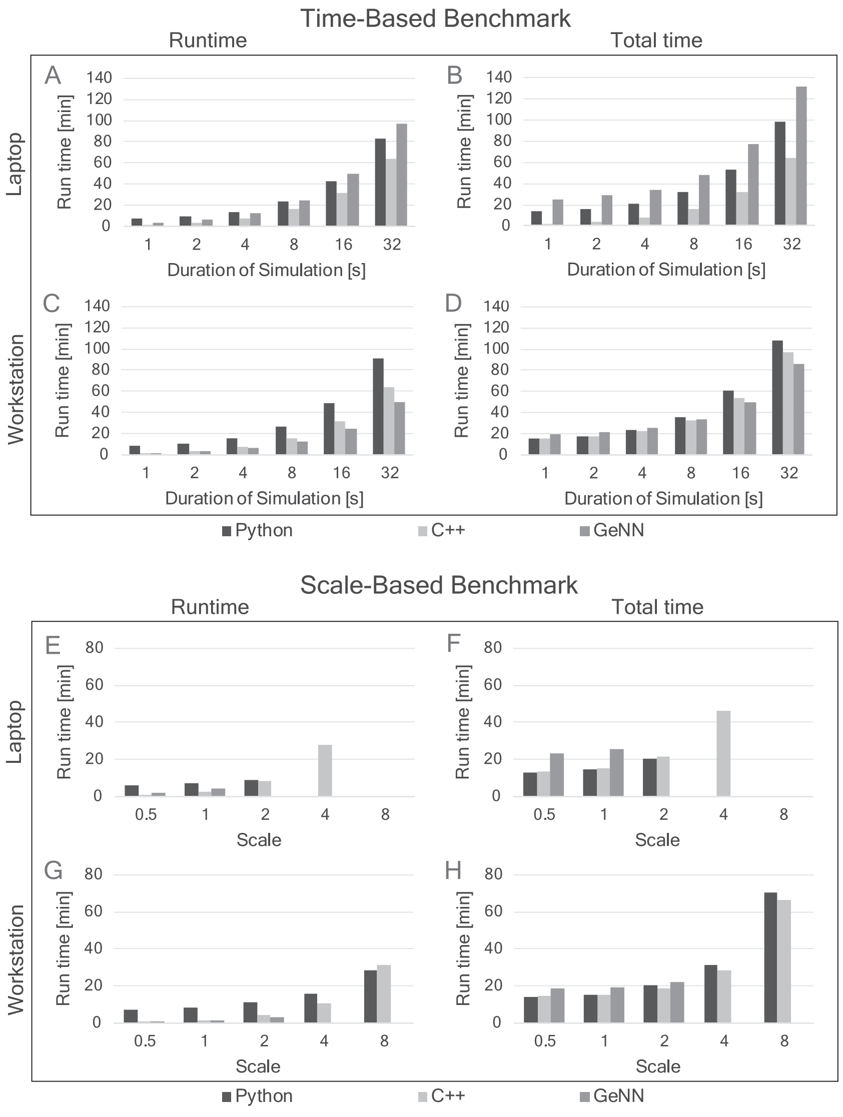

Figure 3: Top: Benchmark of the CxSystem devices with varying simulation times. Laptop (A) run time and (B) total time. Workstation (C) run time and (D) total time. Bottom: Benchmark with varying system scale. Laptop (E) run time and (F) total time. Workstation (G) run time and (H) total time. The missing bars in this figure indicate an out-of-memory state of the simulation. All data points indicate an average of three runs.

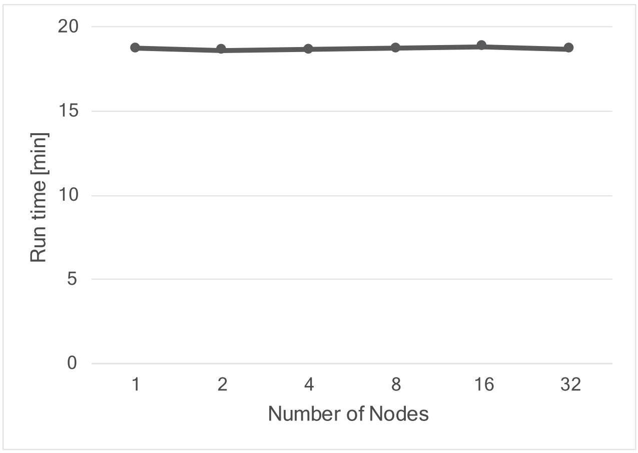

Figure 4: Result of the weak scaling test for the CxSystem running in a cluster at the national IT center for Science. The amount of work is fixed to six independent simulations of the simplified Markram model per node while increasing the number of nodes from 1 to 32, that is from 6 to 192 simulations in total. The waiting time due to the SLURM load manager was omitted.

former, the problem size stays fixed, while the number of processing units is increased, and in the latter, the computation assigned to each processing unit stays constant and additional elements are used to solve a larger total problem.

Currently, strong scaling is not applicable to the CxSystem since it does not parallelize an individual simulation on a controllable set of processing elements. Note that parallelization using GPU is performed automatically with GeNN, and therefore the number of CUDA cores used for processing cannot and should not be customized manually.

The weak scaling test of the CxSystem is instead trivial since by default, one simulation is assigned to one processing element, and it is expected that the total run time will stay constant while increasing the number of simulations in parallel. We tested this on the CxSystem cluster_run mode by running a set of six instances of a simplified model of neocortical microcircuit per node. As expected and shown in Figure 4, the run time stays almost constant while the workload is increased from the six simulations in one node to 192 simulations in 32 nodes.

4 Discussion

We have constructed a cerebral cortex simulation framework that operates on personal computers and tested the simulation software with a COBAHH example (Brette et al., 2007) and a simplification of a recently published comprehensive cortical microcircuit model (Markram et al., 2015). The CxSystem breaks the complexity of biomimetic simulations into separate coding and modeling parts, thus supporting testing and buildup of complex cortical models at a single cell resolution. Current implementation omits many well-known anatomical and physiological features with the aim of minimizing computation time and was used here for cross-device benchmarking purposes. The accompanying work in the companion letter (Hokkanen, Andalibi, & Vanni, 2019, this issue) validates the simplification of the Markram model and examines the computational and conceptual benefits simplification approach.

The CxSystem embraces the main goal of Brian2, minimizing development time, by providing the user with a simplified interface. The interfaces are easily modifiable with common spreadsheet programs and have a biologically meaningful syntax appropriate for life scientists unaccustomed to computer programming.

Benchmarking of the CxSystem showed that its performance is on par with Brian2. This suggests that more thorough comparisons of Brian2 to other simulation engines (such as in Tikidji-Hamburyan et al., 2017) are applicable with a negligible margin.

The device-based performance evaluation of the CxSystem revealed characteristics that are useful for selecting the proper device. Importantly, using the GPU via a GeNN device did not always result in the fastest run. For instance, the performance of the GeNN device on our laptop was always slower than the C++ device, while it outpaced the C++ device in the workstation for long runs. The reason was twofold. First, the Intel Core i7 processor in the laptop performed better than Intel Xeon in the workstation. Second, the higher amount of GPU memory and larger number of CUDA cores in the workstation pushed the run time down compared to the laptop. The GeNN device is not useful for parallel array runs because each parameter set reserves the same GPU resources slowing the simulation. Finally, the weak scaling test showed that using cluster run in the CxSystem is completely suitable for parameter search since the total run time stays constant.

The basic aim of this work has been to reduce the time to implement and run diverse models of the cerebral cortex using an interface that hides the code complexity of the model. Compared to neural mass models (Deco, Jirsa, Robinson, Breakspear, & Friston, 2008), which are used to compute population responses of large-scale networks, we simulate each neuron and synapse separately. This enables synapse-level studies of learning and plasticity, as well as studies on data processing in a population of partially independent nerve cells. We have tested a recent comprehensive cerebral cortex model, originally run on a supercomputer, on our workstations, laptops, and a remote computing cluster. Due to its simplified model structure, we could scale up the Markram model cell and synapse numbers with a factor of 11 with desktop workstations comprising 128 GB memory. This allows extension of the model to multiple interacting areas in a hierarchical fashion and with distinct timing. With fewer neurons per unit area of cortex or with more host computer memory, this could be expanded further.

In comparison to PyNN, a multiplatform simulator (Davison et al., 2009), which aims at an easy shift and comparison between simulation platforms, CxSystem aims at model flexibility while using only the Brian2 simulation platform. Most parameters, including the network design itself, can be modified from the configuration files. This enables fast experimentation and reduces the need for coding skills for end users.

NeuroML is an XML-type language for translation of simulation engine-specific models from one engine to another (Gleeson et al., 2010). Brian does not directly support importing-exporting models to NeuroML. Brian has been our modeling language of choice because it is easy to use and enables scaling up to a systems level while allowing compartmental neural structures. However, in case NeuroML becomes the language of choice for model exchange in the future, we will indeed consider constructing a bridge from our model structure to NeuroML.

Conceptually, the CSV interface can be viewed as a declarative computer language. The user does not need to define the rules or the algorithm of simulation to get the results. In contrast, the CxSystem itself is coded with strongly procedural language, including a Brian2 interface and Python. This dual approach is trying to hide the coding and part of the low-level complexity of the model from the biological synthesis of the system. Instead, we aim at operating at a biological domain-specific level of abstraction while enabling easy parameter searches. The aim is to study the model and make the study as simple as possible. In addition, CxSystem optimizes the computational efficiency by turning the spreadsheet model to C++ or GeNN devices of Brian2.

Since the CxSystem is written on top of Brian2, it inherits most of its limitations too. Unlike NEST and NEURON, a model in Brian2 needs to be analyzed and compiled at every run. When Brian2 performance remains lower than NEST and NEURON, it can be significantly improved by setting the solver to the slightly less accurate exponential-Euler method (Tikidji-Hamburyan et al., 2017). Moreover, the performance of a simulator is inextricably linked with the model that it is simulating, as well as with some simulator-specific parameters, such as refractory period and numerical integration method. Because Brian2 currently lacks support for message passing interface, simulation of large networks consisting of high-fidelity multicompartmental models is probably unfeasible.

Several features are the object of future development. First, we plan to support a wider range of neuron models and improve the usability of the CxSystem in collaboration with life scientists. Second, interactive CSV configuration files could, for instance, guide users to choose the proper unit or parameter and thus reduce the rate of unexpected errors. In the near future, we plan to enable video input through artificial retina. Moreover, we plan to build a support for a system of cortical areas in spatially parallel patches of neural ensembles, mimicking distinct cortical areas.

Acknowledgments

We thank Hanna Heikkinen for programming the earlier version of the simulator, including to large extent the biophysics of the model neurons. We also acknowledge CSC–IT Center for Science, Finland, for computational resources. The work was supported by Helsinki University Hospital research funds to S.V. There are no competing financial interests.

References

Aisa, B., Mingus, B., & O'Reilly, R. (2008). The emergent neural modeling system. *Neural Networks*, *21*(8), 1146–1152.

Almog, M., & Korngreen, A. (2016). Is realistic neuronal modeling realistic? *Journal of Neurophysiology*, *116*(5), 2180–2209.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., . . . Eliasmith, C. (2014). Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*, 48.

Betizeau, M., Dehay, C., & Kennedy, H. (2014). Conformity and specificity of primate corticogenesis. In J. S. Werner and L. M. Chalupa (Eds.), *The new visual neurosciences* (pp. 1407–1422). Cambridge, MA: MIT Press.

Bower, J. M. (1992). Modeling the nervous system. *Trends in Neurosciences*, *15*(11), 411–412.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., . . . Destexhe, A. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *J. Comput. Neurosci.*, *23*(3), 349–398.

Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., . . . Yger, P. (2009). PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, *2*, 11.

Deco, G., Jirsa, V. K., Robinson, P. A., Breakspear, M., & Friston, K. (2008). The dynamic brain: From spiking neurons to neural masses and cortical fields. *PLoS Computational Biology*, *4*(8), e1000092.

Elston, G. N., Oga, T., Okamoto, T., & Fujita, I. (2010). Spinogenesis and pruning from early visual onset to adulthood: An intracellular injection study of layer III pyramidal cells in the ventral visual cortical pathway of the macaque monkey. *Cerebral Cortex*, *20*(6), 1398–1408.

Elston, G. N., & Rosa, M. G. P. (1997). The occipitoparietal pathway of the macaque monkey: Comparison of pyramidal cell morphology in layer III of functionally related cortical visual areas. *Cerebral Cortex*, *7*(5), 432–452.

Eppler, J. M. (2009). PyNEST: A convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, *2*, 12.

Feldman, D. E. (2009). Synaptic mechanisms for plasticity in neocortex. *Annual Review of Neuroscience*, *32*(1), 33–55.

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., & Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *J. Neurosci.*, *23*(37), 11628–11640.

Gerstner, W., Sprekeler, H., & Deco, G. (2012). Theory and simulation in neuroscience. *Science*, *338*(6103), 60–65.

Gewaltig, M.-O., & Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, *2*(4), 1430.

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., . . . Silver, R. A. (2010). NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Computational Biology*, *6*(6), 1–19.

Gleeson, P., Steuber, V., & Silver, R. A. (2007). neuroConstruct: A tool for modeling networks of neurons in 3D space. *Neuron*, *54*(2), 219–235.

Goodman, D., & Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*, *3*, 26.

Heikkinen, H., Sharifian, F., Vigário, R., & Vanni, S. (2015). Feedback to distal dendrites links fMRI signals to neural receptive fields in a spiking network model of the visual cortex. *Journal of Neurophysiology*, *114*(1), 57–69.

Herculano-Houzel, S., Collins, C. E., Wong, P., Kaas, J. H., & Lent, R. (2008). The basic nonuniformity of the cerebral cortex. *Proceedings of the National Academy of Sciences*, *105*(34), 12593–12598.

Hines, M. L., & Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Computation*, *9*(6), 1179–1209.

Hoang, R. V., Tanna, D., Bray, L. C. J., Dascalu, S. M., & Harris Jr., F. C. (2013). A novel CPU/GPU simulation environment for large-scale biologically realistic neural modeling. *Frontiers in Neuroinformatics*, *7*, 19.

Hokkanen, H., Andalibi, V., & Vanni, S. (2019). Controlling complexity of cerebral cortex simulations—II: Streamlined microcircuits. *Neural Computation, 31*(6), 1066–1084.

Horvát, S., Gămănuț, R., Ercsey-Ravasz, M., Magrou, L., Gămănuț, B., Van Essen, D. C., . . . Kennedy, H. (2016). Spatial embedding and wiring cost constrain the functional layout of the cortical network of rodents and primates. *PLoS Biology*, *14*(7), e1002512.

Larkum, M. E., Nevian, T., Sandler, M., Polsky, A., & Schiller, J. (2009). Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: A new unifying principle. *Science*, *325*(5941), 756–760.

Markov, N. T., Ercsey-Ravasz, M. M., Ribeiro Gomes, A. R., Lamy, C., Magrou, L., Vezoli, J., . . . Kennedy, H. (2014). A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex*, *24*(1), 17–36.

Markov, N. T., Ercsey-Ravasz, M., Van Essen, D. C., Knoblauch, K., Toroczkai, Z., & Kennedy, H. (2013). Cortical high-density counterstream architectures. *Science*, *342*(6158), 1238406.

Markov, N. T., Misery, P., Falchier, A., Lamy, C., Vezoli, J., Quilodran, R., . . . Knoblauch, K. (2011). Weight consistency specifies regularities of macaque cortical networks. *Cerebral Cortex*, *21*(6), 1254–1272.

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., . . . Schürmann, F. (2015). Reconstruction and simulation of neocortical microcircuitry. *Cell*, *163*(2), 456–492.

Markram, H., Wang, Y., & Tsodyks, M. (1998). Differential signaling via the same axon of neocortical pyramidal neurons. *Proc. Natl. Acad. Sci. USA*, *95*(9), 5323–5328.

McConnell, S. (2004). *Code complete* (2nd ed.). Seattle: Microsoft Press.

Nowotny, T., Cope, A. J., Yavuz, E., Stimberg, M., Goodman, D. F. M., Marshall, J., & Gurney, K. (2014). SpineML and Brian 2.0 interfaces for using GPU enhanced neuronal networks (GeNN). *BMC Neuroscience*, *15*(1), P148.

Raikov, I., Cannon, R., Clewley, R., Cornelis, H., Davison, A., De Schutter, E., . . . Szatmary, B. (2011). NineML: The network interchange for neuroscience modeling language. *BMC Neuroscience*, *12*(Suppl. 1), P330.

Rall, W. (1962). Theory of physiological properties of dendrites. *Annals of the New York Academy of Sciences*, *96*(4), 1071–1092.

Rasch, B., & Born, J. (2013). About sleep's role in memory. *Physiological Reviews*, *93*(2), 681–766.

Sidiropoulou, K., Pissadaki, E. K., & Poirazi, P. (2006). Inside the brain of a neuron. *EMBO Rep.*, *7*(9), 886–892.

Sjöström, P. J., Rancz, E. A., Roth, A., & Häusser, M. (2008). Dendritic excitability and synaptic plasticity. *Physiological Reviews*, *88*(2), 769–840.

Tikidji-Hamburyan, R. A., Narayana, V., Bozkus, Z., & El-Ghazawi, T. A. (2017). Software for brain network simulations: A comparative study. *Frontiers in Neuroinformatics*, *11*, 46.

Tosi, Z., & Yoshimi, J. (2016). Simbrain 3.0: A flexible, visually-oriented neural network simulator. *Neural Networks*, *83*, 1–10.

Turrigiano, G. G. (2008). The self-tuning neuron: Synaptic scaling of excitatory synapses. *Cell*, *135*(3), 422–435.

Van Essen, D. C., Glasser, M. F., Dierker, D. L., & Harwell, J. (2012). Cortical parcellations of the macaque monkey analyzed on surface-based atlases. *Cerebral Cortex*, *22*(10), 2227–2240.

Vogels, T. P., & Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, *25*(46), 10786–10795.

Yavuz, E., Turner, J., & Nowotny, T. (2016). GeNN: A code generation framework for accelerated brain simulations. *Scientific Reports*, *6*, 18854.

Zucker, R. S., & Regehr, W. G. (2002). Short-term synaptic plasticity. *Annual Review of Physiology*, *64*(1), 355–405.

Received September 28, 2017; accepted May 6, 2018.