HHS Public Access

Author manuscript

J Neurosci Methods. Author manuscript; available in PMC 2023 January 31.

Published in final edited form as:

J Neurosci Methods. 2022 January 15; 366: 109400. doi:10.1016/j.jneumeth.2021.109400.

NeuroGPU: Accelerating multi-compartment, biophysically detailed neuron simulations on GPUs

Roy Ben-Shalom[a,b,c,d,*], **Alexander Ladd**[f], **Nikhil S. Artherya**[f], **Christopher Cross**[a], **Kyung Geun Kim**[f], **Hersh Sanghevi**[f], **Alon Korngreen**[h,i], **Kristofer E. Bouchard**[d,e,g], **Kevin J. Bender**[a,b]

[a]Weill Institute for Neurosciences, Kavli Institute for Fundamental Neuroscience, University of California, San Francisco, San Francisco, CA, United States

[b]Department of Neurology, University of California, San Francisco, San Francisco, CA, United States

[c]MIND Institute University of California, Davis, CA, United States

[d]Computational Research Division, Lawrence Berkeley National Lab, Berkeley, CA, United States

[e]Hellen Wills Neuroscience Institute & Redwood Center for Theoretical Neuroscience, University of California, Berkeley, Berkeley, CA, United States

[f]Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley, CA, United States

[g]Biological Systems and Engineering Division, Lawrence Berkeley National Lab, Berkeley, CA, United States

[h]The Leslie and Susan Gonda Multidisciplinary Brain Research Center, Bar-Ilan University, Ramat-Gan, Israel

[i]The Mina and Everard Goodman Faculty of Life Sciences, Bar-Ilan University, Ramat-Gan, Israel

Abstract

Background: The membrane potential of individual neurons depends on a large number of interacting biophysical processes operating on spatial-temporal scales spanning several orders of magnitude. The multi-scale nature of these processes dictates that accurate prediction of membrane potentials in specific neurons requires the utilization of detailed simulations. Unfortunately, constraining parameters within biologically detailed neuron models can be difficult,

[*]Correspondence to: University of California, Davis MIND Institute Wet Lab 2805 50th Street, Room 2460 Sacramento, CA 95817, United States., rbenshalom@ucdavis.edu (R. Ben-Shalom), kevin.bender@ucsf.edu (K.J. Bender).

CRediT authorship contribution statement

Roy Ben-Shalom: Conceptualization, Methodology, Software Development, Writing. **Alexander Ladd:** Software development, Data curation, GitHub management, Video recordings Editing. **Nikhil S. Artherya:** Software development, Writing. **Christopher Cross:** Software Development, Visualization, Editing. **Kyung Geun Kim:** Software development, Editing. **Hersh Sanghevi:** Software Development, Editing. **Alon Korngreen:** Conceptualization, Editing. **Kristofer E. Bouchard:** Conceptualization, Visualization, Editing. **Kevin J. Bender:** Conceptualization, Visualization, Editing.

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.jneumeth.2021.109400.

leading to poor model fits. This obstacle can be overcome partially by numerical optimization or detailed exploration of parameter space. However, these processes, which currently rely on central processing unit (CPU) computation, often incur orders of magnitude increases in computing time for marginal improvements in model behavior. As a result, model quality is often compromised to accommodate compute resources.

New Method: Here, we present a simulation environment, NeuroGPU, that takes advantage of the inherent parallelized structure of the graphics processing unit (GPU) to accelerate neuronal simulation.

Results & comparison with existing methods: NeuroGPU can simulate most biologically detailed models 10–200 times faster than NEURON simulation running on a single core and 5 times faster than GPU simulators (CoreNEURON). NeuroGPU is designed for model parameter tuning and best performs when the GPU is fully utilized by running multiple (> 100) instances of the same model with different parameters. When using multiple GPUs, NeuroGPU can reach to a speed-up of 800 fold compared to single core simulations, especially when simulating the same model morphology with different parameters. We demonstrate the power of NeuoGPU through large-scale parameter exploration to reveal the response landscape of a neuron. Finally, we accelerate numerical optimization of biophysically detailed neuron models to achieve highly accurate fitting of models to simulation and experimental data.

Conclusions: Thus, NeuroGPU is the fastest available platform that enables rapid simulation of multi-compartment, biophysically detailed neuron models on commonly used computing systems accessible by many scientists.

Keywords

Compartmental models; Biophysical simulations; Conductance-based models; Electrophysiology; Graphical Processing Unit

1. Introduction

Electrical activity of single neurons is determined by the distribution of various ionic conductances arranged across complex morphologies (Mainen and Sejnowski, 1996; Hausser et al., 2000; London and Häusser, 2005; Spruston, 2008; Hay et al., 2013; Alonso and Marder, 2019). Our understanding of single neurons has long relied on the ability to construct biophysically rigorous models of how neuronal membrane potential [Vm], and, hence, action potentials (APs, spikes) are generated from currents [I] flowing across the membrane and through the cell (Fig. 1) (Hodgkin and Huxley, 1952). Wilfrid Rall described the biophysical theory of how membrane potential of a single neuronal segment (‘compartment’) depends on the conductance (e.g., gNa) and voltage dependent flow of specific ionic species [e.g., sodium (Na) and potassium (K)], as well as passive properties of the membrane (i.e., capacitance) (Fig. 1, top row) (Rall, 1962a). Using cable theory, Rall further described how to connect different compartments of a neuron, providing the foundation for modeling complex, spatially extended neuronal morphologies (Fig. 1A) (Rall, 1962b). Concomitantly, the membrane channels that mediate a specific ionic current exhibits large diversity of genetically defined conductances (e.g., gNav1.2, gNav1.6, etc.,), indicating that individual compartments are, in reality, quite complex (Fig. 1B). While just beginning

to be appreciated at the time, we now know that ion channels have their own voltage dependent kinetics that are determined by the transition probabilities amongst various states of the channel subunits (Fig. 1C) (Hille, 1984; Colquhoun and Hawkes, 1995). Finally, simulations of realistic neural network models require faithfully capturing the complexity of the individual neurons (Einevoll, et al., 2019). While the physical theories required to link these vastly disparate spatial-temporal scales exists, our ability to utilize them for basic understanding and clinical translation is impeded by the computational burden of the required simulations (Fig. 1D and E).

Our predictive understanding of several neuronal cell classes has benefitted from the repeated refinement of compartmental models that describe their activity in health and disease. For example, neocortical pyramidal cells have been modeled extensively, providing insight into the effects of morphology on AP firing characteristics (Mainen and Sejnowski, 1996; Hay et al., 2011; Almog and Korngreen, 2014), how synaptic input patterns affect spike output (Maršálek et al., 1997; Dlesmann et al., 1999; Destexhe et al., 2003) how APs initiate and propagate within axons (Kole et al., 2007, 2008; Shu et al., 2007; Hu et al., 2009; Hallermann et al., 2012; Cohen et al., 2020), and how neuronal activity is affected by alterations in ion channel function induced by genetic variation (Zamponi et al., 2010; Allen et al., 2014; Ben-Shalom et al., 2017; Spratt et al., 2019). Similar intensive studies have focused on other cell classes, including hippocampal pyramidal cells (Mainen et al., 1996; Magee and Cook, 2000; Poirazi et al., 2003; Narayanan and Johnston, 2008; Milstein et al., 2015), cerebellar Purkinje cells (De Schutter and Bower, 1994; Miyasho et al., 2001; Roth and Häusser, 2001), and midbrain dopaminergic neurons (Canavier, 1999; Canavier and Landry, 2006; Kuznetsova et al., 2010). In parallel with these advances in modeling, there has recently been an enormous improvement in experimental approaches to better understand the diversity of neuronal classes and their activity patterns. Within the general group of neocortical pyramidal cells, for example, exists a wealth of diversity. This includes not only differences in morphology and activity across laminae (Smith and Häusser, 2010; Deitcher et al., 2017; Kanari et al., 2019), but also within laminae depending on genetic makeup or axonal projection targets (Dembrow et al., 2010; Gee et al., 2012; Clarkson et al., 2017), or even within what was thought to be a homogenous cell class within a single layer as one samples across a large region of cortex (Fletcher and Williams, 2019).

Given the enormous complexity and vast spatio-temporal scales described above, generating models that accurately recapitulate neuronal activity across the true range of diversity present in nature can be a daunting task. Model fitting often requires one to tune individual parameters to best match empirical observations. This process can be aided by iterative rounds of parameter exploration and optimization that aim to minimize the differences between empirical data targets and their associated models. These procedures can be computationally demanding (Fig. 1D and E). Indeed, each linear improvement in model accuracy requires an exponential increase in computational resources (Nocedal and Wright, 2006; Gurkiewicz and Korngreen, 2007). Thus, model optimization is often done on supercomputers that parallelize these computations across massive number of central processing unit (CPU) cores. Unfortunately, the cost of constructing and operating supercomputing centers is similarly massive. As such utilization of these resources are typically restricted to large consortia, such as the Blue Brain Project (BBP) (Markram

et al., 2015) and the Allen Institute (Gouwens et al., 2018). For more restricted budgets, simulations must typically be compromised in scale, complexity, or accuracy, thus negatively impacting results (Almog and Korngreen, 2016).

In the past 10 years, graphics processing units (GPUs) have emerged as an alternative to CPU-based hardware that may offer comparable levels of performance at substantially reduced cost for some problems. GPUs utilize streaming multiprocessors with multiple simple cores that allow for distributed, parallelized computing for relatively small chunks of data. With software optimized for GPUs, GPU-based computing can often outperform CPU-based applications in processing speed and cost for some problems (Payne et al., 2010). Today, GPUs are being used in scientific simulations, including molecular dynamics (Go et al., 2012; Salomon-Ferrer et al., 2013) and climate modeling (Prein et al., 2015), and are the computational engine for most modern artificial intelligence applications (Schmidhuber, 2015). In neuroscience, GPUs are currently being used to accelerate complex imaging dataset processing (Eklund et al., 2013), spiking neural network analysis (Yavuz et al., 2016; Chou et al., 2018), clustering of activity from in vivo extracellular electrophysiological experiments (Pachitariu et al., 2016), and simulations of single ion channels (Ben-Shalom et al., 2012).

Recently, two platforms for neuronal biophysical simulations with GPU support were developed: Arbor (Akar et al., 2019) and CoreNeuron (Kumbhar et al., 2019). Both platforms focused on simulating large scale neuronal networks comprised of detailed multi-compartmental models. CoreNeuron supports previous NEURON models but is not implemented in CUDA, the fundamental operating language of NVIDIA's GPUs. As such, its ability to accelerate model runtimes with GPUs may not exploit the full potential of GPU computing. Arbor, instead, is implemented in CUDA via an entirely new simulation environment. Thus, while it does harness the speedup potential of GPUs, it is not clear how existing models, such as those found in ModelDB and the BBP portal (McDougal et al., 2015; Ramaswamy et al., 2015), could be ported to Arbor, thus impeding utilization. Here, we describe NeuroGPU, a computational platform optimized to exploit GPU architecture to dramatically accelerate the simulation of multi-compartmental neuronal models. Our goal with NeuroGPU was different than that of CoreNeuron or Arbor. Rather than focusing on neuronal network simulations, NeuroGPU is designed to optimize fitting of models that best recapitulate empirical data derived from single neurons, and to study the parameter space of such models by iterating parameter values. To do so, we developed new approaches to parallelize compartmental models, utilizing the GPU-based programming language CUDA to optimize memory handling on GPUs manufactured by NVIDIA. This resulted in simulation speedups of up to 200-fold on a single GPU and up to 800-fold using a set of 4 GPUs. Similar optimizations may be made for other GPU manufactures, but this was not considered here, given the broad usage of NVIDIA hardware. We found that NeuroGPU performed faster than NEURON using (Message Passing Interface) MPI by up to 10-fold and CoreNeuron up to 4-fold. Building on our previous efforts (Ben-Shalom et al., 2013), we developed an intuitive user interface that can import most compartmental models implemented in NEURON (Carnevale and Hines, 2006) deposited at the ModelDB (McDougal et al., 2017) or BBP portal (Ramaswamy et al., 2015). Further, we provide methods to explore model parameter space to study how each parameter of the model

contributes to its voltage output. The runtimes of NeuroGPU enables us to sample the parameter space in a very detailed manner, which were utilized here to reveal the response landscape of single neurons. Finally, we provide an interface to use NeuroGPU for fitting models to experimental data with evolutionary algorithms (DEAP and BluePyOpt) (Gagn, 2012; Van Geit et al., 2016). NeuroGPU implemented such model optimization algorithms in 2.5 h using a single GPU (see Fig. 5), compared to 26 h using 256 CPUs, which is the current standard (Nandi et al., 2020; Schneider-Mizell et al., 2020). This enables the use of more complex models that will better represent experimental data, as we demonstrate here in both simulated and experimental data. NeuroGPU therefore provides an open-source platform for neuronal simulation with increased speed and reduced cost, thus enabling the neuroscience community to perform high quality biophysically detailed simulations.

2. Methods

2.1. Hardware

NEURON and TitanXP-based simulations were run on a PC with Intel Core I7–7700 K 4.2 GHz with 16 GB of RAM. Tesla V100-based simulations were run using the NVIDIA PSG cluster. Here, each simulation was run on a single node with Haswell or Skylake CPU cores. For multi-GPU simulations, we used cluster nodes with NVLINK (Li et al., 2019) between the GPUs to enable memory peer-access. Fitting models to experimental data was done on the Cori GPU cluster from the National Energy Research Scientific Computing Cent (NERSC) at Lawrence Berkeley National Labs. Cori GPU nodes includes 8 NVIDIA Tesla V100 and 20 Skylake CPU cores with total of 384 GB memory.

2.2. Software

Simulations were performed in NEURON 7.6–8.0 and CUDA 10.1. All scripts were written in Python 3.7. All software is available at https://github.com/roybens/NeuroGPU.

2.3. Importing NEURON models

To ease installation, we separated processes for porting models (Translation Fig. S2C) and NeuroGPU execution (Execution Fig. S2D). After initial import, NeuroGPU models can run on a GPU machine independently from NEURON. The python script extractmodel.py exports NEURON models to NeuroGPU. This script reads all simulation details from runModel.hoc, which is populated using the GUI (Fig. S1). NEURON models are described using either hoc or python scripts. The scripts include a morphology that can either be called as a separate file or constructed within the script (Fig S1B). The user must input a file containing model stimulation, which includes temporal aspects of the model and command currents delivered at a prescribed location. Furthermore, all free parameters, such as channel properties, must be described. These import components are translated into CUDA code, termed kernels, that can run on the GPU via the python script “extractmodel.py” (Fig S1C). This script first takes runModel.hoc and loads it into NEURON, not to run simulations, but rather to query NEURON for model properties needed for subsequent porting to NeuroGPU, including compartment names and the tri-diagonal matrix (F-Matrix or Hines Matrix) which holds the differential system for the voltages of the dendritic tree. Then, the script iterates over the.mod files in the directory, parses them and creates relevant kernels for

each mechanism described. CUDA kernels containing model mechanisms are generated by adding relevant CUDA keywords to C code generated when NEURON compiles mechanisms. Mechanism kernels are written to the AllModels. cu in similar structure as described previously (Hines and Carnevale, 2000; Carnevale and Hines, 2006), iterating over all compartments defined in the model. A new hoc file is created to register mechanism values, which are stored in AllParams.csv and inserted in each compartment. After model parameter maps are determined, they are cataloged as part of ParamMappings.txt for reference for future reiterations of the same model, eliminating the need to reload NEURON. Finally, the script writes code translated to CUDA in NeuroGPU.cu and packages the application to run on either Windows or Unix. After compiling the code, an executable is created that reads the AllParams. csv and the stimulation and runs the model on the GPU.

2.4. Translating mechanisms to CUDA and memory assignment

Mechanisms in NEURON are described by NMODL (.mod) files (Hines and Carnevale, 2000), which update the mechanism states every simulation time step. This is done using three different procedures within NEURON that initialize mechanisms (nrn_init), update currents that mechanisms affect (nrn_cur), and then update mechanism states (nrn_state) (Carnevale and Hines, 2006). In NeuroGPU, CUDA kernels are written for each of these procedures using.mod and.c files that are generated by NEURON when running nrnivmodl. When a NEURON model is exported to NeuroGPU, all.mod files and their corresponding.c files are parsed using custom code to extract variables and procedures defined in each specific.mod file. These are then translated to CUDA kernels written in AllModels.cu. Future iterations of NeuroGPU may make use of the nmodl Python library for parsing as this software evolves.

Several mechanisms are regulated by intracellular calcium. To support this, we created an array that holds internal calcium concentration within each compartment and calculates the reveral potential for calcium at each time step. We determined that calcium was the only ion whose intracellular concentration varied substantially during ongoing activity in ways that affect simulations, due to their effects on calcium-activated potassium channels. We note that most models lack detailed models of sodium and potassium pumps/transporters and generally do not model changes in their concentrations. We found that this was a reasonable approximation for NeuroGPU, as the presence or absence of Na/K ion concentration calculations did not affect simulation Vm.

CUDA is an extension of the C programming language that enables computation on the GPU (Nvidia, 2018). CUDA kernels are procedures running on the GPU that can be invoked from either the GPU or CPU. To invoke a kernel from the CPU, one must specify the number of parallel threads used. Threads, which allow for parallelization on the GPU, are organized into blocks, with each thread occupying a specific address within that block (idx.x) (Fig S2C). GPUs are structured to operate well when computing 32 parallel threads, a computing structure termed a warp (Nvidia, 2018). Therefore, we structured NeuroGPU to utilize 32 threads in the x dimension, corresponding to individual morphological segments within the model. For a given model with more than 32 segments, individual threads are

responsible for calculating every 32nd segment. For example, thread #1 would calculate segments 1, 33, 65, … 32N + 1.

2.5. Extracting simulation properties from NEURON

NeuroGPU utilizes NEURON for simulation pre-processing, including mechanism translation which includes mathematical descriptions of various ion channels, calcium diffusion characteristics, and other elements of neuronal function, (Hines and Carnevale, 2000), a map for mechanism distribution across compartments (ParamMappings.txt), and exporting the tri-diagonal matrix using fmatrix(). These are stored in BasicConstSegP.csv. NEURON extracts all parameters for cable equations and mechanism values within each compartment to AllParams.csv (Fig. S1). External stimulation delivery location, intensity, and time-course are written in stim.csv. Resting membrane potential and number of time steps in the simulation are written in sim.csv.

2.6. Solving the tridiagonal matrix

Matrix solutions were performed using the branch-based parallelism approach as described previously (Ben-Shalom et al., 2013), with morphology analysis guiding iterative matrix computations. This analysis is done in extractmodel.py and the data structures to solve the tri-diagonal in parallel is stored in BasicConstSegP.csv.

2.7. Benchmarking

All benchmarking against a single CPU core running NEURON was done with NEURON 7.6, running in a single thread. The morphology was adjusted to have one segment per compartment in all platforms. Simulation runtimes were compared without hard drive read/write file steps, as these aspects depend more on hard drive properties than CPU/GPU comparisons. Benchmarking against the parallel version of NEURON was done using NEURON 8.0. In both cases, runtime compares time required to complete psolve() procedures. For NEURON-MPI, we used CoreNeuron settings with 32 MPI processors without GPUs (Kumbhar et al., 2019). For CoreNeuron, we used one MPI thread and one GPU (Kumbhar et al., 2019). Note, benchmarking of cases with 2^14 model instances is estimated based on linear extrapolation from the 2^13 case, as the time required to load model instances in the 2^14 case far exceeded the time allowed by policy queues at NSERC.

2.8. Multi-compartmental models

NeuroGPU performance was tested with 4 different models:

1. A passive model, utilizing passive channels described in NEURON distribution pas.mod file. These channels were distributed on both simple and complex morphologies (see Fig. S3A, D) (Mainen and Sejnowski, 1996). The simple morphology was based on the simple morphology described in Mainen and Sejnowski, with compartments reduced to 32, as this is the minimum number of compartments required for NeuroGPU-based simulations.
2. The Mainen and Sejnowski (1996) model, with channels distributed on the same complex and simple morphologies. Channels are distributed as in (Mainen and Sejnowski, 1996) (Fig. S4)

3. A pyramidal cell model from the Blue Brain Project portal (Ramaswamy et al., 2015) (Fig. 2). BBP PC refers to the model named L5_TTPC1_cADpyr232_1.

4. A chandelier cell model, termed BBP CC, referring to L5_ChC_dNAC222_1. For this model, the Kdshu2007.mod files were altered to run on NeuroGPU. Specifically, global variables were removed from the neuron block and instead placed in the assigned block (Carnevale and Hines, 2006) (Fig. 2).

2.9. Optimization algorithms

Two different genetic algorithm versions were used in this study. For data related to Fig. 4, the *eaMuPlusLambda* algorithm from the DEAP package. *eaMuPlusLambda* stands for evolutionary algorithm where the next generation population is comprised from the existing population (Mu) and the offspring (Lambda). We modified the varOR procedure to call NeuroGPU (Rainville et al., 2012).”Optimization was performed on the BBP PC model. For each iteration, the algorithm began with a new population of parameters with values randomly chosen with the range specified in Supplemental Table 3. The model was modified to accept new values from the optimization algorithm (similar changes were necessary for parameter space exploration for Fig. 3). Target data were generated using the original parameters values described in Supplemental Table 3. Optimization was targeted to reduce error between target data and test data using both the interspike interval (ISI) and the root mean square (RMS) of the voltage as the error function. Error was reduced to a single variable by weighting these two variables as: 10*ISI + RMS.

For data related to Fig. 5, the BluePyOpt (Van Geit et al., 2016) implementation of Multiple Objective Optimization (MOO) was used. Experimental target data for these experiments were from whole-cell current-clamp recordings from layer 5b thick tufted pyramidal cells in acute slices from wild-type mouse prefrontal cortex (postnatal day 62) (Spratt et al., 2019). Optimization was targeted to minimize the root mean square voltage error at each time point between empirical data and model output as well as the following objectives, as defined in the electrophysiology feature extraction library (eFEL) from the BBP: voltage_base, AP_amplitude, voltage_after_stim, ISI_values, spike_half_width, and afterhyperpolarization_depth. Electrophysiological data were fitted in models with morphology from L5_TTPC1_cADpyr232 Fig. 5(A) or a reconstructed prefrontal cortex L5 thick tufted pyramidal neuron deposited at NeuroMorpho.Org (Ascoli et al., 2007; Yin et al., 2018). The model parameters that were varied for the S1 model and PFC model are described in Supplemental Tables 4 and 5, respectively.

2.10. Support

A series of tutorial notebooks that walk users through various approaches are available on Github, with corresponding video walkthroughs available at: https://www.youtube.com/playlist?list=PL-Amxh_lBdw99alE5L1yfnfwPuK2wLeel These tutorials describe: 1) the structure of the documentation, 2) standards for input data and file structure to run NeuroGPU, 3) porting of models from NEURON to NeuroGPU, 4) Parameter space exploration, and 5) Using DEAP optimization. Future developments will be documented in the NeuroGPU Github portal.

3. Results

Our goal for NeuroGPU was to develop user-friendly software for fitting compartmental models to experimental data, with improved speed, using relatively low-cost hardware. Furthermore, we sought to make NeuroGPU cross-compatible with NEURON, to allow one to import models available on public databases, including ModelDB and the BBP portal. Toward that end, we utilized the same basic structure as NEURON, including the use of *hoc* and *mod* files that define all aspects of the compartmental model (Fig. S1). To increase simulation speed, we focused primarily on parallelizing the most computationally intensive aspects of NEURON simulations in GPU architecture. NEURON calculates the voltages of each segment of the model by solving a system of differential equations that describes current flow in each compartment. Within NEURON, this system of differential equations is represented within a tri-diagonal matrix (Hines, 1984). Typically, matrix elements for neighboring compartments are solved in serial, as current flow in one compartment is interdependent on flow in neighboring compartments. We and others have previously developed methods to solve this tri-diagonal matrix in parallel across GPUs, despite the interdependence of current flow across compartments (Fig. S2) (Hines, 1984; Hines et al., 2008; Ben-Shalom et al., 2013). At that time, the method was implemented only for classic Hodgkin-Huxley models with 3 parameters (gNa, gK, gLeak) (Ben-Shalom et al., 2013). Here, we extended this method to support a wider range of models, including most models available in ModelDB and the Blue Brain Project (BBP) repository. We implemented this in Python and created an iPython Graphical User Interface (GUI) for easy use.

3.1. NeuroGPU Implementation

Traditionally, neuronal simulations diffusion matrixes are computed serially, since calculating the voltage at each section depends on voltage in neighboring sections. To leverage the Single Instruction Multiple Data (SIMD) architecture of GPUs, one needs to instead solve the diffusion matrix in parallel. This challenge has slowed adoption of GPUs for neuronal simulation (Kumbhar et al., 2019); however, in 2013, we described an algorithm that uses a parallel lower upper (LU) decomposition to solve the diffusion matrix, based on the work of Stone (Stone, 1973; Ben-Shalom et al., 2013). Our implementation relied on the fact that separate neuronal branches can be calculated in parallel before calculating regions where they merge at branch points, thus parallelizing computations. Each neuron can therefore be calculated using 32 parallel threads, with each thread calculating the voltage for n/32 compartments where n is the total number of segments in the model. A thread first calculates the contribution of all the mechanisms and then updates the mechanisms states, similar to implementation in NEURON (Carnevale and Hines, 2006). Given this structure, NeuroGPU outperforms NEURON only when the GPU is fully utilized when multiple model instances are simulated in parallel (Fig. 2). In such cases, several blocks of 32 threads each simulate a different model instance. To comply with SIMD requirements, these multiple models must have the same morphology to allow threads in different blocks to execute the same instructions. This approach is advantageous for techniques like parameter exploration, where multiple model instances with different conductance parameters built on the same morphology are compared. In this design, memory is optimized, since all the models share constant memory that holds mechanism

details, morphological details, and the indices to solve the diffusion matrix in parallel. Individual models hold copies of the matrix diagonal, which changes every time step, and the parameters and states of the mechanisms that vary between model instances. Importantly, many of the optimizations made here can be leveraged to address other questions that are appropriate for GPU-based computing, or could be optimized better in future iterations. Current optimizations and future applications are discussed in Supplemental Table 1.

3.2. NeuroGPU Performance

To evaluate how NeuroGPU performs relative to NEURON, we benchmarked it for speed and accuracy across a range of models and hardware configurations. We first compared NeuroGPU performance with a single GPU to NEURON implemented on a single CPU core. To benchmark speed, we evaluated computing time for multiple instances of the same model. NeuroGPU was primarily evaluated on recently developed models from the Blue Brain Project (BBP) portal (Hay et al., 2011; Markram et al., 2015; Ramaswamy et al., 2015), but was also benchmarked on models with reduced morphology or reduced numbers of voltage-gated channels or ligand-gated receptors to determine how each of these aspects affects performance (Figs. S3–4). We used BBP models to benchmark different versions of NEURON (NEURON, MPI-Neuron, CoreNeuron). We focused on two specific models: a layer 5 pyramidal neuron (Fig. 2, top row: BBP PC, see Methods for specific model) and a layer 5 chandelier interneuron (Fig. 2, bottom row: BBP CC). Initially, models were interrogated with a range of stimulus intensities to determine relative differences between NeuroGPU and NEURON (Fig. 2).

We first confirmed the quality of the simulations. Overall, NeuroGPU was able to replicate NEURON simulations with high fidelity; however, small voltage differences were observed between the two platforms in all models tested. These were most commonly observed when voltage was changing rapidly between time steps (Fig. 2B–C, G–H), and were due to small differences in timing that likely arise from different approaches to number rounding in GPUs vs CPU architecture (Whitehead, 2011).

NEURON computation time scales linearly with the number of simulations, and, for relatively small numbers of model instances (< 8), outperforms NeuroGPU (Fig. 2, D,E,I,J). By contrast, models implemented on GPUs (NeuroGPU and CoreNEURON) scale linearly only after saturating all of the GPUs streaming multiprocessors (Nvidia, 2018) (Fig. 2D, E, I, J). Similarly, runtime scales linearly with number of mechanisms (e.g., distinct conductances), provided such mechanisms can be stored in the GPU's constant memory (see NeuroGPU Implementation in the Results section). For example, in the Mainen model, 37 mechanisms could be simulated on a V100 GPU. This limitation is not an issue for most models, which typically use fewer mechanisms, and may be less of a bottleneck in future hardware with a larger constant memory allocation. Similarly, MPI-NEURON simulations start to scale linearly after all MPI processors (32) are occupied. NeuroGPU is 4.3x slower than MPI-NEURON when simulating only one neuron, but 5x faster when simulating 2^14 neurons Fig. 2D. For NeuroGPU and CoreNeuron, processing times are practicaly constant for any simulation incorporating fewer than 128 model instances, and begin to outpace NEURON simulations when > 8 simulations are run simultaneously. When computing

large numbers of model instances, NeuroGPU outpaces both parallel versions of NEURON: NeuroGPU is 1.5–3.3x faster than CoreNEURON for the PC model and 1.9–4.1x faster for the CC model. This difference is likely due to differences in how memory transfers are implemented in the two programs. In NeuronGPU, memory transfers require little overhead, due to its use of CUDA, whereas CoreNeuron has significant overhead for similar transfers (Supplemental Table 2) (Anon, 2019).

Relative gains in processing time were noted when 8–16,384 model instances were run simultaneously. These gains were dependent on hardware. For example, implementing NeuroGPU on an NVIDIA TitanXP and Tesla V100 GPU resulted in 1.8–3.8x faster runtime when using the Tesla V100 GPUs (Fig. S3F). It is worth noting that TitanXP hardware is relatively low cost (< $1099) and a very similar card (NVIDIA GTX-1080) can currently be purchased for less than $500. As such, significant improvements in processing speed can be obtained even with modestly priced hardware. Additional returns can be gained from GPU tethering, as CUDA has been recently updated to allow for memory sharing across GPUs. To evaluate this, we connected up to 4 Tesla V100 GPUs together and measured speedup on both BBP models displayed in Fig. 2. As expected, adding more GPUs increased the overall processing capacity, and we noted shifts in the number of model instances that could be handled simultaneously before reaching maximum GPU utilization (Fig. 2E and J). Furthermore, speedup was almost 3 orders of magnitude faster relative to NEURON.

3.3. Exploring neuronal model parameter space

Parameter values (e.g. ion channels distributions) are correlated in a non-linear manner. This may lead to situations where vastly different parameter combinations nevertheless produce similar voltage outputs, at least for a limited set of stimuli (Prinz et al., 2004). The diversity of these parameter sets can be limited by constraining the range over which a parameter can vary before initiating model optimization, thus leading to more biologically realistic sets of parameters. NeuroGPU may be ideal for parameter exploration within such ranges, as these types of simulations require one to repeatedly model the same morphology with small differences in constituent parameter values, a process that lends itself well to parallelization within GPUs. Indeed, we predict that relative speedups would be identical to situations considered above (Fig. 2) and depend simply on the number of parameter sets used. To provide an example of parameter space exploration, we examined neuronal output (i.e., number of action potentials) in the BBP PC model when co-varying the density of the axonal fast inactivating sodium channel and axonal slow-inactivating potassium channel over a range of 0–10 and 0–20 S/cm^2, respectively. Single traces from with different sodium and potassium conductances are shown in Fig. 3A and total spike output as function of these two channel densities is shown in Fig. 3B. As expected, increasing sodium conductance allowed models to generate more APs until sodium conductance was so high that models entered depolarization block. Similarly, reducing potassium conductance produced comparable results. Interestingly, certain combinations of sodium and potassium conductance concentrations produced bursting phenotypes characterized by high-frequency APs riding atop long-duration depolarizations. These presumably reflect parameter ranges that then interact with other ion channels in the model (e.g., Ca_V3 of HCN channels) that promote such burst dynamics.

3.4. Fitting models to surrogate and empirical data

With the ability to rapidly sample parameter space, NeuroGPU may be ideally suited to accelerate model fitting to data, where a key constraint is the time needed to exhaustively sample possible solutions. To test this, we implemented two different genetic optimization algorithms within NeuroGPU. Initially we integrated the DEAP (Distributed Evolutionary Algorithms in Python) package (Gagn, 2012) (Fig. 4). Genetic algorithm success lies in the balance between exploration of the whole parameter space and the exploitation of specific areas that seem promising. For this, large sample populations are ideal, as this allows for effective and broad parameter space exploration. NeuroGPU is more efficient when many instances are running in parallel, allowing for more effective application of genetic algorithms.

Genetic optimization was tested here by fitting model-generated voltages to a single voltage epoch containing APs that was generated by the default values present in the BBP PC model. We focused first on such surrogate data, as the ground truth values for all parameters are already known. As such, we can easily compare how well NeuroGPU performs in arriving at similar values. Optimization began with different population sizes comprised of 100–10,000 individual parameter sets with random initial values (Fig. 4B). These populations were run in four independent trials, each for 50 generations, and the difference between the naïve model and ground-truth model was compressed to a single score value (see Methods). For these scores, lower values indicate less difference between the two cases. Scores improved for each of these populations, but the variance across trials and the overall score were markedly affected by the population size, with score decreasing in a near-linear fashion with each doubling of population size (Fig. 4C). These score improvements were paralleled by a decrease in total processing time. For example, optimization with 10,000 individual parameter sets ran 7.7× faster on NeuroGPU than NEURON (Fig. 4D; 10 vs 77 h, respectively). While these are significant improvements in simulation speed, they are relatively modest compared to those observed in other conditions (Fig. 2), likely since the version of NeuroGPU used here required NEURON to load the simulation and generate parameter values. In the next section we present how eliminating calls to NEURON greatly increases speedup.

Given these promising results fitting surrogate data, we next asked whether similar performance could be noted for empirical electrophysiological data. Therefore, we fitted the BBP PC model to whole-cell current-clamp recordings of action potential activity from neurons in acute mouse frontal cortex slices (Fig. 5A black traces). Models were implemented on two different pyramidal cell morphologies from somatosensory (original BBP PC morphology; Fig. 5A) and prefrontal cortex [from NeuroMorpho.Org; Fig. 5B (Ascoli et al., 2007; Yin et al., 2018)] to determine whether morphology differentially affects optimization. Here we used the BluePyOpt package, which is an extension DEAP that is specified for neuronal optimizations (Van Geit et al., 2016). We fitted the model to eight voltage responses from varying stimulations to verify that the model is robust across stimulation conditions (Fig. 5A, B). The optimization algorithm found values for the model's parameters that resulted with good fits to both morphologies. Thus, models can be fitted to empirical data and new morphologies by combining NeuroGPU with BluePyOpt.

Unlike single trace stimulations (Fig. 4), optimization to 8 separate traces required more generations to obtain reasonable fits, which unfortunately increased processing time dramatically. To reduce runtimes, we revised NeuroGPU by adding two final features. First, we eliminated NEURON calls to the CPU entirely by generating a procedure that identifies the free parameters of the model and modifies NeuroGPU's input, 'AllParams.csv', to new values without using NEURON. Second, we used CPU multithreading scoring, which reduced the overall computation time of each generation. With these improvements, runtimes were reduced for each iteration of the optimization algorithm. Simulation of all 8 traces with 10,000 putative solutions required 60 s of processing time. This was followed by a 103 s period required for scoring for each genetic algorithm generation. This latter aspect has been accelerated dramatically. On 8 CPUs, the same process requires 2891 s per generation. These speedups (48.2 fold in simulation time) are close to the speedups of single simulations (Fig. 2D). With these improvements, we were able to obtain models that fit well to experimental data using an 8 GPU node running NeuroGPU in just 4 h, a task that traditionally may require several days of computation using a large cluster (Hay et al., 2011; Hill et al., 2011; Almog and Korngreen, 2014; Nandi et al., 2020; Schneider-Mizell et al., 2020). As such, NeuroGPU allow one the ability to develop more complex models and fit them to empirical data in reasonable time frames (Fig. 1).

4. Discussion

Detailed models of neurons are critical to our understanding of neuronal functioning. However, the computational resources required of current software implementations of complex neuronal models are prohibitive, typically requiring supercomputers. At best, this limits the accuracy of results; at worst, it limits access to all but a select set of scientists. To address this gap, based on our previous efforts (Ben-Shalom et al., 2013), we designed a user-friendly environment that enables one to port multi-compartmental models for implementation with CUDA to run simulations on GPUs. By taking advantage of parallel processing inherent to GPUs, we were able to accelerate simulations dramatically—up to 2–3 orders of magnitude with multiple GPUs. NeuroGPU was developed to be interoperable with NEURON, thereby allowing anyone with expertise in the NEURON environment access to GPU-based acceleration. Towards this goal, we developed a platform to easily port NEURON models from either ModelDB or the BBP portal (Ramaswamy et al., 2015; McDougal et al., 2017) using a iPython notebook-based graphical user interface (GUI). We further developed GUIs for creating stimulation protocols, parameter exploration, and genetic optimization.

Leveraging on our parallel algorithm for solving the tri-diagonal matrix representing the dendritic tree (Ben-Shalom et al., 2013), we developed an NMODL importer to CUDA. This allowed us to translate any single neuron multi-compartmental model developed in NEURON to NeuroGPU. We implemented NEURON's framework in CUDA and translated NMODL mechanisms to kernels for GPU execution. NeuroGPU has two independent processes:: First, models are translated into GPU-executable code (Fig S1C). Second, simulations of many instances of this model are executed on GPUs. Importantly, these two processes can be done on different machines, helping facilitate NeuroGPU installation on

GPU cluters, as only CUDA and pythons are required for execution on such clusters (and not NEURON or other software).

The power of GPUs to speedup application comes from the single instruction multiple data (SIMD) architecture where one instruction can modify many different memory instances in parallel. In 2013 we showed for the first time that single neurons can be simulated efficiently in parallel utilizing SIMD architecture (Ben-Shalom et al., 2013). We suggested a novel algorithm to solve the tri-diagonal (Hines) matrix (Hines, 1984), which is the core of neuronal simulations and usually is the main calculation bottleneck. Since that time, other solutions for solving Hines matrices have been suggested, such as using Exact Domain Decomposition method (EDD) (Vooturi et al., 2018) and splitting the dendritic tree and solving each subtree asynchronously (Magalhães et al., 2019). Also, a tailored solver for the Hines tridiagonal matrices has been developed that produced major speedups (Valero-Lara et al., 2018). However, it is not clear how these algorithms are to be used by the neuroscience community. Conversely, NeuroGPU is a full CUDA implementation that builds on our tri-diagonal solver to accelerate existing NEURON models. As discussed in Ben-Shalom et al., 2013, we use the Stone algorithm (Stone, 1973) for parallelizing the tri-diagonal solver. While this might not be the optimal solver, NeuroGPU still outperforms other GPU based neuronal simulations. Implementing other solvers using our CUDA implementation might lead to better performance, and will be tested in future versions of NeruoGPU.

NeuroGPU addresses a major gap in currently implemented GPU-based simulation environments. Two other neuronal simulation environments for multi-compartmental models have been implemented using GPUs, CoreNeuron (Kumbhar et al., 2019) and Arbor (Akar et al., 2019). These environments are designed primarily to accelerate large scale network simulations. NeuroGPU, by contrast, is focused on greatly accelerating the simulation of single neurons with complex, multi-compartment morphologies, critical for exploring the parameter space of single models and optimizing such models to best fit empirical data.

Leveraging the acceleration of single-neuron simulations, NeuroGPU has expanded GUIs for parameter exploration, which allows for rapid assessment of how changes in ion channel density across compartments affects neuronal excitability (Fig. 3). This approach may be particularly useful to generate testable hypotheses regarding channel distribution with pharmacological manipulations (Keren et al., 2009; Almog and Korngreen, 2014; Mäki-Marttunen et al., 2018), modulation of ion channels (Byczkowicz et al., 2019), or in disease states where ion channel density is thought to be affected (Migliore and Migliore, 2012; Miceli et al., 2013; Ben-Shalom et al., 2017; Spratt et al., 2019). Furthermore, one could generate a range of cells with variable channel densities and confirm that their activity is physiologically realistic (e.g., Fig. 3, all cases before generating depolarization block). These conditions could then be used as building blocks for variable activity within neuronal networks (Prinz et al., 2003, 2004; Alonso and Marder, 2019). In addition to parameter exploration, NeuroGPU is designed for extensive model optimization (Gagn, 2012; Van Geit et al., 2016). Fitting complex models to empirical data is computationally expensive, often requiring days of compute time, even on large supercomputing systems (Hay et al., 2011; Almog and Korngreen, 2014). Here, we show that NeuroGPU accelerated model fitting runtime 7.7 fold (Fig. 4). While appreciable, these accelerations can be improved further

by bypassing model export via NEURON and instead using the “mapping parameters” function. In this configuration, runtime was accelerated 48.2 fold. Of note, this runtime does not match that found for simulating single neuron instances (Fig. 2), since score calculation using the CPU imposes a bottleneck. Using more sophisticated evolutionary algorithms that can minimize CPU latency or implementing the score function calculation on the GPU may result in further improvements, scaling towards GPU acceleration of 800 fold observed with multiple GPUs (e.g., Fig. 2). Nevertheless, the current instantiation of NeuroGPU offers individual labs the opportunity to implement optimization algorithms with their own hardware. Furthermore, it opens the door to extremely high-speed model fitting, as NeuroGPU can be run easily on GPU supercomputing systems, which inherently have exponentially more computational resources compared than similarly kitted CPU systems. Finally, NeuroGPU is ideal for generating neuronal datasets with different configurations for exploration of, e.g., firing rate ‘phenotypes (e.g., Prinz et al., 2004), or for deep learning training (Ben-Shalom et al., 2019; Gonçalves et al., 2020).

Recent advances in genetic characterization and novel analysis methods have resulted in characterization of diverse neuronal types with respect to their morphologies, projections, and protein expression (Gouwens et al., 2019). However, these advancements lack detailed biophysical models that can describe and simulate these neurons. In the BBP portal there are ~200 models that each is comprised of an m-type, which describes the morphology, and one of 11 e-types, which is a set of conductances that describes the electrical properties of the cell. Currently, these e-types describe somewhat generalized activity patterns [e.g., neurons that fire at high frequency without accommodation, or neurons that have stuttering firing patterns (Markram et al., 2015)]. With NeuroGPU we can easily expand this repertoire of e-types, and even replace generalizable e-types with models that recapitulate the activity of data obtained from single neurons. This would improve network simulations, as single neurons would display biologically accurate diversity within and across neuronal subtypes (e.g., Fig. 5B).

NeuroGPU accelerates compartmental modeling through parallelization of matrix calculations. Solving the tridiagonal matrix is the most computationally demanding aspect of multi-compartmental model simulations (Hines, 1984; Hines et al., 2008; Ben-Shalom et al., 2013). Therefore, we took advantage of fast, on-GPU memory and controlled the timing of calculations and memory transfers to optimize the use of computational resources (Volkov and Demmel, 2008; Ben-Shalom et al., 2013; Nvidia, 2018). Resulting speedups depended primarily on neuronal morphology and the implemented conductance complexity (Fig. 2D&I, Fig. S3 S4). Overall, GPU utilization was limited by execution dependencies (profiling data not shown), where one aspect of GPU processing could not proceed until another aspect either transferred or processed its own memory. In the future, these dependencies may be further reduced through either dynamic parallelization (Zhang et al., 2015) or by increasing instruction level parallelism (ILP) (Volkov and Demmel, 2008). Nevertheless, the current version of NeuroGPU accelerates single neuron compartmental simulations by several orders of magnitude.

4.1. Future goals for NeuroGPU

Future iterations of NeuroGPU may expand on the strengths and address limitations in using GPUs for compartmental modeling. Ion channels are modeled typically with Markov-based kinetics, or a simpler Markov approximation based on Hodgkin-Huxley type equations. NeuroGPU currently supports Hodgkin-Huxley-based mechanisms only, as we previously found that implementation of full Markov-based mechanisms on GPUs requires too much shared memory and reduces performance drastically (Ben-Shalom et al., 2012). Furthermore, NeuroGPU is currently limited in the mechanisms that can be translated easily, including passive conductances and voltage-gated ion channels, and does not at present support point-processes, including synapses. Nevertheless, the current iteration of NeuroGPU can support most neuron models on ModelDB (70% of such models utilized Hodgkin-Huxley mechanisms), and the majority of models from the BBP (Ramaswamy et al., 2015) and the Allen Brain institute (Gouwens et al., 2018).

As with total GPU utilization, improvements in memory handling may improve these cases. Furthermore, GPUs work best when the same instructions are occurring simultaneously on multiple memory addresses. This makes them ideal for iterating through models with identical morphologies and different channel distributions, but less ideal for network models containing a diversity of neuron types. As an intermediate, one could address this limitation by modeling networks containing discrete sets of neurons. For example, a network could contain several compartmental morphology models that each support multiple instances with different channel parameters, similar to the Ring model applied by Arbor (Akar et al., 2019; Kumbhar et al., 2019).

NeuroGPU will help democratize multi-compartmental modeling. While NeuroGPU can support simulations in large, tightly integrated systems using UNIX-based mutli-GPU architectures, it also is ideal for individual laboratories running simulations on Windows-based workstations with GPUs, which are becoming increasingly common. Indeed, a workstation with total costs < $3000, when kitted with appropriate GPUs, can out-perform loosely inter connected CPU-based clusters (i.e., with commodity inter-node connection hardware). This could help broaden the use and utility of multi-compartmental modeling by bringing supercomputer-level processing power to a large range of research settings.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

We are grateful to Dr. Gilad Liberman who helped conceptualize this project. To the support and advice of NVIDIA developers – Dr. Jonathan Lefman, Dr. Jonathan Bentz, Dr. Xuemeng Zhang and Angela Chen in optimizing the CUDA code. To Maxwell Chen and Mathew Derango who helped with code development. To NVIDIA Corporation for donating the GPUs used in this study. To all the members of the Bender Lab for critically assessing this work. This research was supported by NIH Grants F32 NS095580 (RBS), MH112729 (KJB), and DA035913 (KJB).

References

Akar NA, Cumming B, Karakasis V, Küsters A, Klijn W, Peyser A, Yates S, 2019. Arbor - A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures. In: Proceedings - 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2019, pp 274–282.

Allen NM, Mannion M, Conroy J, Lynch SA, Shahwan A, Lynch B, King MD, 2014. The variable phenotypes of KCNQ-related epilepsy. Epilepsia 55, e99–e105. [PubMed: 25052858]

Almog M, Korngreen A, 2014. A quantitative description of dendritic conductances and its application to dendritic excitation in layer 5 pyramidal neurons. J. Neurosci. 34, 182–196. [PubMed: 24381280]

Almog M, Korngreen A, 2016. Is realistic neuronal modeling realistic? J. Neurophysiol. 2 jn.00360.2016.

Alonso LM, Marder E, 2019. Visualization of currents in neural models with similar behavior and different conductance densities. Elife 8.

Anon, 2019. CUDA C BEST PRACTICES GUIDE Design Guide.

Ascoli GA, Donohue DE, Halavi M, 2007. NeuroMorpho.Org: a central resource for neuronal morphologies. J. Neurosci. 27, 9247–9251. [PubMed: 17728438]

Ben-Shalom R, Aviv A, Razon B, Korngreen A, 2012. Optimizing ion channel models using a parallel genetic algorithm on graphical processors. J. Neurosci. Methods 206, 183–194. [PubMed: 22407006]

Ben-Shalom R, Liberman G, Korngreen A, 2013. Accelerating compartmental modeling on a graphical processing unit. Front. Neuroinform. 7, 4. [PubMed: 23508232]

Ben-Shalom R, Keeshen CM, Berrios KN, An JY, Sanders SJ, Bender KJ, 2017. Opposing effects on NaV1.2 function underlie differences between SCN2A variants observed in individuals with autism spectrum disorder or infantile seizures. Biol. Psychiatry 82, 224–232. [PubMed: 28256214]

Ben-Shalom R, Balewski J, Siththaranjan A, Baratham V, Kyoung H, Kim KG, Bender KJ, Bouchard KE, 2019. Inferring neuronal ionic conductances from membrane potentials using CNNs. bioRxiv:727974.

Byczkowicz N, Eshra A, Montanaro J, Trevisiol A, Hirrlinger J, Kole MHP, Shigemoto R, Hallermann S, 2019. HCN channel-mediated neuromodulation can control action potential velocity and fidelity in central axons. Elife 8.

Canavier CC, Landry RS, 2006. An increase in AMPA and a decrease in SK conductance increase burst firing by different mechanisms in a model of a dopamine neuron in vivo. J. Neurophysiol. 96, 2549–2563. [PubMed: 16885519]

Canavier CC, 1999. Sodium Dynamics Underlying Burst Firing and Putative Mechanisms for the Regulation of the Firing Pattern in Midbrain Dopamine Neurons: A Computational Approach.

Carnevale NT, Hines ML, 2006. The NEURON Book. Cambridge University Press,.

Chou TS, Kashyap HJ, Xing J, Listopad S, Rounds EL, Beyeler M, Dutt N, Krichmar JL, 2018. CARLsim 4: an open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters. In: Proceedings of the International Joint Conference on Neural Networks. Institute of Electrical and Electronics Engineers Inc.

Clarkson RL, Liptak AT, Gee SM, Sohal VS, Bender KJ, 2017. D3 receptors regulate excitability in a unique class of prefrontal pyramidal cells. J. Neurosci. 37, 5846–5860. [PubMed: 28522735]

Cohen CCH, Popovic MA, Klooster J, Weil MT, Möbius W, Nave KA, Kole MHP, 2020. Saltatory conduction along myelinated axons involves a periaxonal nanocircuit. Cell 180, 311–322.e15. [PubMed: 31883793]

Colquhoun D, Hawkes AG, 1995. A Q-Matrix Cookbook. Single-Channel Recording. Springer US, Boston, MA, pp. 589–633.

De Schutter E, Bower JM, 1994. An active membrane model of the cerebellar Purkinje cell I. Simulation of current clamps in slice. J. Neurophysiol. 71, 375–400. [PubMed: 7512629]

Deitcher Y, Eyal G, Kanari L, Verhoog MB, Atenekeng Kahou GA, Mansvelder HD, De Kock CPJ, Segev I, 2017. Comprehensive morpho-electrotonic analysis shows 2 distinct classes of L2 and L3 pyramidal neurons in human temporal cortex. Cereb. Cortex 27, 5398–5414. [PubMed: 28968789]

Dembrow NC, Chitwood RA, Johnston D, 2010. Projection-specific neuromodulation of medial prefrontal cortex neurons. J. Neurosci. 30, 16922–16937. [PubMed: 21159963]

Destexhe A, Rudolph M, Paré D, 2003. The high-conductance state of neocortical neurons in vivo. Nat Rev Neurosci 4, 739–751. [PubMed: 12951566]

Dlesmann M, Gewaltig MO, Aertsen A, 1999. Stable propagation of synchronous spiking in cortical neural networks. Nature 402, 529–533. [PubMed: 10591212]

Einevoll GT, Destexhe A, Diesmann M, Grün S, Jirsa V, de Kamps M, Migliore M, Ness TV, Plesser HE, Schürmann F, 2019. The Scientific Case for Brain Simulations. Neuron 102, 735–744. [PubMed: 31121126]

Eklund A, Dufort P, Forsberg D, LaConte SM, 2013. Medical image processing on the GPU - past, present and future. Med. Image Anal. 17, 1073–1094. [PubMed: 23906631]

Fletcher LN, Williams SR, 2019. Neocortical topology governs the dendritic integrative capacity of layer 5 pyramidal neurons. Neuron 101, 76–90.e4. [PubMed: 30472076]

Gagn C, 2012. DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. 13, 2171–2175.

Gee S, Ellwood I, Patel T, Luongo F, Deisseroth K, Sohal VS, 2012. Synaptic activity unmasks dopamine D2 receptor modulation of a specific class of layer V pyramidal neurons in prefrontal cortex. J. Neurosci. 32, 4959–4971. [PubMed: 22492051]

Go AW, Williamson MJ, Xu D, Poole D, Grand S.Le, Walker RC, Götz AW, Williamson MJ, Xu D, Poole D, Le Grand S, Walker RC, 2012. Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. generalized born. J. Chem. Theory Comput. 8, 1542–1555. [PubMed: 22582031]

Gonçalves PJ, Lueckmann JM, Deistler M, Nonnenmacher M, Öcal K, Bassetto G, Chintaluri C, Podlaski WF, Haddad SA, Vogels TP, Greenberg DS, Macke JH, 2020. Training deep neural density estimators to identify mechanistic models of neural dynamics. Elife 9, 1–46.

Gouwens NW, Berg J, Feng D, Sorensen SA, Zeng H, Hawrylycz MJ, Koch C, Arkhipov A, 2018. Systematic generation of biophysically detailed models for diverse cortical neuron types. Nat. Commun. 9.

Gouwens NW, et al. , 2019. Classification of electrophysiological and morphological neuron types in the mouse visual cortex. Nat Neurosci 22, 1182–1195. [PubMed: 31209381]

Gurkiewicz M, Korngreen A, 2007. A numerical approach to ion channel modelling using whole-cell voltage-clamp recordings and a genetic algorithm. PLoS Comput. Biol. 3, e169. [PubMed: 17784781]

Hallermann S, de Kock CPJ, Stuart GJ, Kole MHP, 2012. State and location dependence of action potential metabolic cost in cortical pyramidal neurons. Nat. Neurosci. 15, 1007–1014. [PubMed: 22660478]

Hausser M, Spruston N, Stuart GJ, 2000. Diversity and dynamics of dendritic signaling. Science (80-) 290, 739–744.

Hay E, Hill S, Schürmann F, Markram H, Segev I, 2011. Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. PLoS Comput. Biol. 7, e1002107. [PubMed: 21829333]

Hay E, Schurmann F, Markram H, Segev I, Schürmann F, Markram H, Segev I, 2013. Preserving axosomatic spiking features despite diverse dendritic morphology. J. Neurophysiol. 109, 2972–2981. [PubMed: 23536715]

Hill S, Markram H, Segev I, Druckmann S, Berger TK, Schu F, 2011. Effective stimuli for constructing reliable neuron models. PLoS Comput. Biol. 7.

Hille B, 1984. Ionic Channels of Excitable Membranes, third. Sinauer Associates, Sunderland, Mass.

Hines M, 1984. Efficient computation of branched nerve equations. Int. J. Biomed. Comput. 15, 69–76. [PubMed: 6698635]

Hines ML, Carnevale NT, 2000. Expanding NEURON's repertoire of mechanisms with NMODL. Neural Comput. 12, 995–1007. [PubMed: 10905805]

Hines ML, Eichner H, Schürmann F, 2008. Neuron splitting in compute-bound parallel network simulations enables runtime scaling with twice as many processors. J. Comput. Neurosci. 25, 203–210. [PubMed: 18214662]

Hodgkin AL, Huxley AF, 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. Bull. Math. Biol. 117, 25–71 discussion 5–23.

Hu W, Tian C, Li T, Yang M, Hou H, Shu Y, 2009. Distinct contributions of Na(v) 1.6 and Na(v)1.2 in action potential initiation and backpropagation. Nat. Neurosci. 12, 996–1002. [PubMed: 19633666]

Kanari L, Ramaswamy S, Shi Y, Morand S, Meystre J, Perin R, Abdellah M, Wang Y, Hess K, Markram H, 2019. Objective morphological classification of neocortical pyramidal cells. Cereb. Cortex 29, 1719–1735. [PubMed: 30715238]

Keren N, Bar-Yehuda D, Korngreen A, 2009. Experimentally guided modelling of dendritic excitability in rat neocortical pyramidal neurones. J. Physiol. 587, 1413–1437. [PubMed: 19171651]

Kole MHP, Letzkus JJ, Stuart GJ, 2007. Axon initial segment Kv1 channels control axonal action potential waveform and synaptic efficacy. Neuron 55, 633–647. [PubMed: 17698015]

Kole MHP, Ilschner SU, Kampa BM, Williams SR, Ruben PC, Stuart GJ, 2008. Action potential generation requires a high sodium channel density in the axon initial segment. Nat. Neurosci. 11, 178–186. [PubMed: 18204443]

Korngreen A, Sakmann B, 2000. Voltage-gated K+ channels in layer 5 neocortical pyramidal neurones from young rats: subtypes and gradients. J. Physiol. 525 (Pt 3), 621–639. [PubMed: 10856117]

Kumbhar P, Hines M, Fouriaux J, Ovcharenko A, King J, Delalondre F, Schürmann F, 2019. CoreNEURON: An Optimized Compute Engine for the NEURON Simulator.

Kuznetsova AY, Huertas MA, Kuznetsov AS, Paladini CA, Canavier CC, 2010. Regulation of firing frequency in a computational model of a midbrain dopaminergic neuron. J. Comput. Neurosci. 28, 389–403. [PubMed: 20217204]

Li A, Song SL, Chen J, Li J, Liu X, Tallent N, Barker K, 2019. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect.

London M, Häusser M, 2005. Dendritic computation. Annu. Rev. Neurosci. 28, 503–532. [PubMed: 16033324]

Magalhães BRC, Sterling T, Hines M, Schürmann F, 2019. Asynchronous branch-parallel simulation of detailed neuron models. Front. Neuroinform. 13.

Magee JC, Cook EP, 2000. Somatic EPSP amplitude is independent of synapse location in hippocampal pyramidal neurons. Nat. Neurosci. 3, 895–903. [PubMed: 10966620]

Mainen ZF, Sejnowski TJ, 1996. Influence of dendritic structure on firing pattern in model neocortical neurons. Nature 382, 363–366. [PubMed: 8684467]

Mainen ZF, Carnevale NT, Zador AM, Claiborne BJ, Brown TH, 1996. Electrotonic architecture of hippocampal CA1 pyramidal neurons based on three-dimensional reconstructions. J. Neurophysiol. 76, 1904–1923. [PubMed: 8890303]

Mäki-Marttunen T, Halnes G, Devor A, Metzner C, Dale AM, Andreassen OA, Einevoll GT, 2018. A stepwise neuron model fitting procedure designed for recordings with high spatial resolution: application to layer 5 pyramidal cells. J. Neurosci. Methods 293, 264–283. [PubMed: 28993204]

Markram H, et al. , 2015. Reconstruction and simulation of neocortical microcircuitry. Cell 163, 456–492. [PubMed: 26451489]

Maršálek P, Koch C, Maunsell J, 1997. On the relationship between synaptic input and spike output jitter in individual neurons. Proc Natl Acad Sci U S A 94, 735–740. [PubMed: 9012854]

McDougal R. a, Morse TM, Hines ML, Shepherd GM, 2015. Modelview for ModelDB: online presentation of model structure. Neuroinformatics 13, 459–470. [PubMed: 25896640]

McDougal RA, Morse TM, Carnevale T, Marenco L, Wang R, Migliore M, Miller PL, Shepherd GM, Hines ML, 2017. Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. J. Comput. Neurosci. 42, 1–10. [PubMed: 27629590]

Miceli F, Soldovieri MV, Ambrosino P, Barrese V, Migliore M, Cilio MR, Taglialatela M, 2013. Genotype–phenotype correlations in neonatal epilepsies caused by mutations in the voltage sensor of K_V 7.2 potassium channel subunits. Proc. Natl. Acad. Sci. USA 110, 4386–4391. [PubMed: 23440208]

Migliore M, Migliore R, 2012. Know your current Ih: interaction with a shunting current explains the puzzling effects of its pharmacological or pathological modulations Attali B, ed. PLoS One 7, e36867. [PubMed: 22606301]

Milstein AD, Bloss EB, Apostolides PF, Vaidya SP, Dilly GA, Zemelman BV, Magee JC, 2015. Inhibitory gating of input comparison in the CA1 microcircuit. Neuron 87, 1274–1289. [PubMed: 26402609]

Miyasho T, Takagi H, Suzuki H, Watanabe S, Inoue M, Kudo Y, Miyakawa H, 2001. Low-threshold potassium channels and a low-threshold calcium channel regulate Ca2+ spike firing in the dendrites of cerebellar Purkinje neurons: a modeling study. Brain Res. 891, 106–115. [PubMed: 11164813]

Nandi A, Chartrand T, Geit W. Van, Buchin A, Yao Z, Lee SY, Wei Y, Kalmbach B, Lee B, Lein E, Berg J, Sümbül U, Koch C, Tasic B, Anastassiou C.Nandi, A., Chartrand T, Van Geit, W., Buchin A, Yao Z, Lee SY, Wei Y, Kalmbach B, Lee B, Lein E, Berg J, Sümbül U, Koch C, Tasic B, Anastassiou C, 2020. Single-neuron models linking electrophysiology, morphology and transcriptomics across cortical cell types. bioRxiv:2020.04.09.030239.

Narayanan R, Johnston D, 2008. The h channel mediates location dependence and plasticity of intrinsic phase response in rat hippocampal neurons. J. Neurosci. 28, 5846–5860. [PubMed: 18509046]

Nocedal J, Wright S, 2006. Numer. Optim.

Nvidia C, 2018. Cuda c Programming Guide, Version 9.1. NVIDIA Corp.

Pachitariu M, Steinmetz N, Kadir S, Carandini M, D HK, 2016. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. bioRxiv: 061481.

Payne JL, Sinnott-Armstrong NA, Moore JH, 2010. Exploiting graphics processing units for computational biology and bioinformatics. Interdiscip. Sci. 2, 213–220. [PubMed: 20658333]

Poirazi P, Brannon T, Mel BW, 2003. Pyramidal neuron as two-layer neural network. Neuron 37, 989–999. [PubMed: 12670427]

Prein AF, Langhans W, Fosser G, Ferrone A, Ban N, Goergen K, Keller M, Tölle M, Gutjahr O, Feser F, Brisson E, Kollet S, Schmidli J, Van Lipzig NPM, Leung R, 2015. A review on regional convection-permitting climate modeling: demonstrations, prospects, and challenges. Rev. Geophys. 53, 323–361. [PubMed: 27478878]

Prinz AA, Billimoria CP, Marder E, 2003. Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. J. Neurophysiol. 90, 3998–4015. [PubMed: 12944532]

Prinz AA, Bucher D, Marder E, 2004. Similar network activity from disparate circuit parameters. Nat. Neurosci. 7, 1345–1352. [PubMed: 15558066]

Rainville F. De, Fortin F, Gardner M, Parizeau M, Gagné C, 2012. DEAP: a python framework for evolutionary algorithms. Companion Proc. Genet. Evol. Comput. Conf 85–92.

Rall W, 1962a. Theory of physiological properties of dendrites. Ann. N.Y. Acad. Sci. 96, 1071–1092. [PubMed: 14490041]

Rall W, 1962b. Electrophysiology of a dendritic neuron model. Biophys. J. 2, 145–167. [PubMed: 14490040]

Ramaswamy S, Courcol JD, Abdellah M, Adaszewski SR, Antille N, Arsever S, Atenekeng G, Bilgili A, Brukau Y, Chalimourda A, Chindemi G, Delalondre F, Dumusc R, Eilemann S, Gevaert ME, Gleeson P, Graham JW, Hernando JB, Kanari L, Katkov Y, Keller D, King JG, Ranjan R, Reimann MW, Rössert C, Shi Y, Shillcock JC, Telefont M, Van Geit W, Diaz JV, Walker R, Wang Y, Zaninetta SM, DeFelipe J, Hill SL, Muller J, Segev I, Schürmann F, Muller EB, Markram H, 2015. The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. Front. Neural Circuits 9, 44. [PubMed: 26500503]

Roth A, Häusser M, 2001. Compartmental models of rat cerebellar Purkinje cells based on simultaneous somatic and dendritic patch-clamp recordings. J. Physiol. 535, 445–472. [PubMed: 11533136]

Salomon-Ferrer R, Götz AW, Poole D, Le Grand S, Walker RC, Go AW, Poole D, Grand S.Le, Walker RC, 2013. Routine microsecond molecular dynamics simulations with AMBER on GPUs.

2. Explicit solvent particle mesh ewald. J. Chem. Theory Comput. 9, 3878–3888. [PubMed: 26592383]

Schmidhuber J, 2015. Deep Learning in neural networks: an overview. Neural Netw. 61, 85–117. [PubMed: 25462637]

Schneider-Mizell CM et al. , 2020. Chandelier cell anatomy and function reveal a variably distributed but common signal. bioRxiv:2020.03.31.018952.

Shu Y, Yu Y, Yang J, McCormick D a, 2007. Selective control of cortical axonal spikes by a slowly inactivating K+ current. Proc. Natl. Acad. Sci. USA 104, 11453–11458. [PubMed: 17581873]

Smith SL, Häusser M, 2010. Parallel processing of visual space by neighboring neurons in mouse visual cortex. Nat. Neurosci. 13, 1144–1149. [PubMed: 20711183]

Spratt PWE, Ben-Shalom R, Keeshen CM, Burke KJ, Clarkson RL, Sanders SJ, Bender KJ, 2019. The autism-associated gene Scn2a contributes to dendritic excitability and synaptic function in the prefrontal cortex. Neuron.

Spruston N, 2008. Pyramidal neurons: dendritic structure and synaptic integration. Nat. Rev. Neurosci. 9, 206–221. 〈https://www.nature.com/articles/nrn2286〉 . [PubMed: 18270515]

Stone HS, 1973. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. J. ACM 20, 27–38.

Valero-Lara P, Martínez-Pérez I, Sirvent R, Martorell X, Peña AJ, 2018. cuThomasBatch and cuThomasVBatch, CUDA Routines to compute batch of tridiagonal systems on NVIDIA GPUs. In: Concurrency Computation.

Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol J-D, Muller E, Schürmann F, Segev I, Markram H, 2016. BluePyOpt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. arXiv 10:1–18.

Volkov V, Demmel JW, 2008. Benchmarking GPUs to tune dense linear algebra. In: 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 1–11.

Vooturi DT, Kothapalli K, Bhalla US, 2018. Parallelizing Hines matrix solver in neuron simulations on GPU. In: Proceedings of the - 24th IEEE Int Conf High Perform Comput HiPC 2017 2017-December, pp. 388–397.

Whitehead N, 2011. Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs.

Yavuz E, Turner J, Nowotny T, 2016. GeNN: A code generation framework for accelerated brain simulations. Sci. Rep. 6, 1–14. [PubMed: 28442746]

Yin L, Zheng R, Ke W, He Q, Zhang Y, Li J, Wang B, Mi Z, Long Y. sheng, Rasch MJ, Li T, Luan G, Shu Y, 2018. Autapses enhance bursting and coincidence detection in neocortical pyramidal cells. Nat. Commun. 9, 1–12. [PubMed: 29317637]

Zamponi GW, Lory P, Perez-Reyes E, 2010. Role of voltage-gated calcium channels in epilepsy. Pflug. Arch. Eur. J. Physiol. 460, 395–403. 〈https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/s00424-009-0772-x&casa_token=w42m9NgEtgcAAAAA:mRvrqQi8F18ax7hUuXOhRian5voI0wG33bvVOj2CgJV86srCg7ISBBYP4SmpR_tqw-NCMpEI2Vc8fmW1cKk〉 [Accessed June 30, 2020].

Zhang P, Holk E, Matty J, Misurda S, Zalewski M, Chu J, McMillan S, Lumsdaine A, 2015. Dynamic parallelism for simple and efficient GPU graph algorithms. In: Proceedings of the 5th Workshop on Irregular Applications Architectures and Algorithms - IA3 '15. ACM Press, New York, New York, USA, pp 1–4.

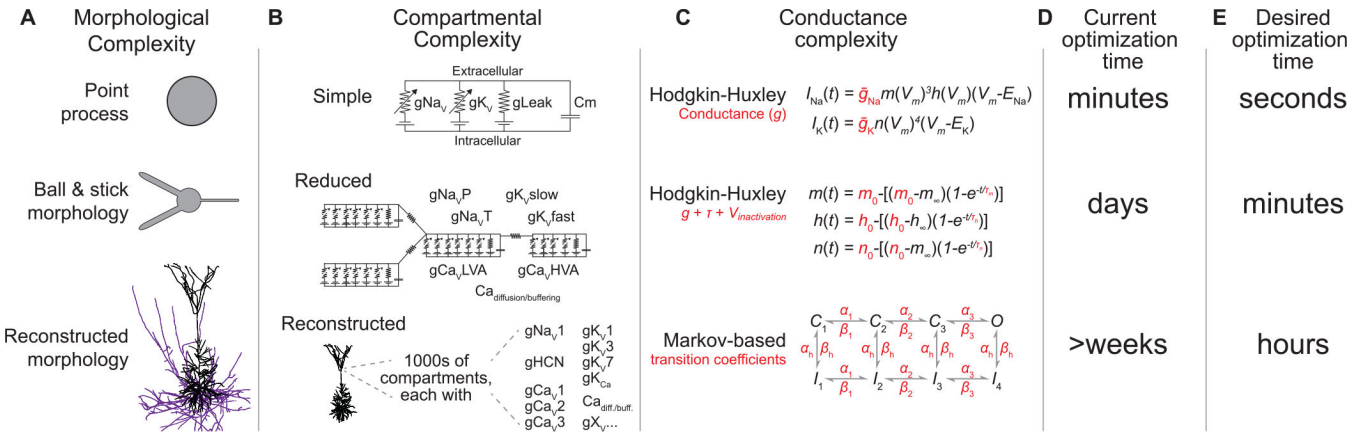

Fig. 1.

Optimizing biophysical neuronal models increase with their complexity. Biophysical neuronal model complexity depends on three factors (A–C) which determine computational costs required to simulate and fit models to empirical data (optimization) (D and E). A: Morphological complexity: The abstraction of the neuronal morphology ranges from a point neuron, to a simplified morphology, and ultimately toward different methods to reconstruct neuronal morphology with high resolution. B: Compartmental complexity: Compartmental models vary in complexity both in number of compartments and the content of each compartment. Top: Single compartment with basic, conductances required for spiking and a resting membrane potential. Middle: Multi-compartmental model where ion-channels are aggregated under several conductances e.g. all different voltage gated potassium channels are represented in a slow and fast inactivating channel (Korngreen and Sakmann, 2000). Bottom: High resolution morphology with detailed representation to all channels sub-types in each compartment. C: Single-channel complexity: The overall conductance in compartmental models is formalized either with Hodgkin-Huxley equations or with Markov-based models. When fitting a model to empirical data, several parameters can be varied. Top: only the maximal conductance in a Hodgkin-Huxley formulation. Middle: coefficients added to Hodgkin-Huxley equations (time constants, voltage dependence). Bottom: In Markov-based channels the transition coefficients can be varied. D: Optimizing a model (fitting model to data) depends on the complexity of the model and number of free parameters. When complexity and number of free parameters require compute times that exceed a few days, it becomes impractical to use high resolution models. E: Developing tools to accelerate simulation and optimization time will enable us to use more complicated and biophysical relevant models.

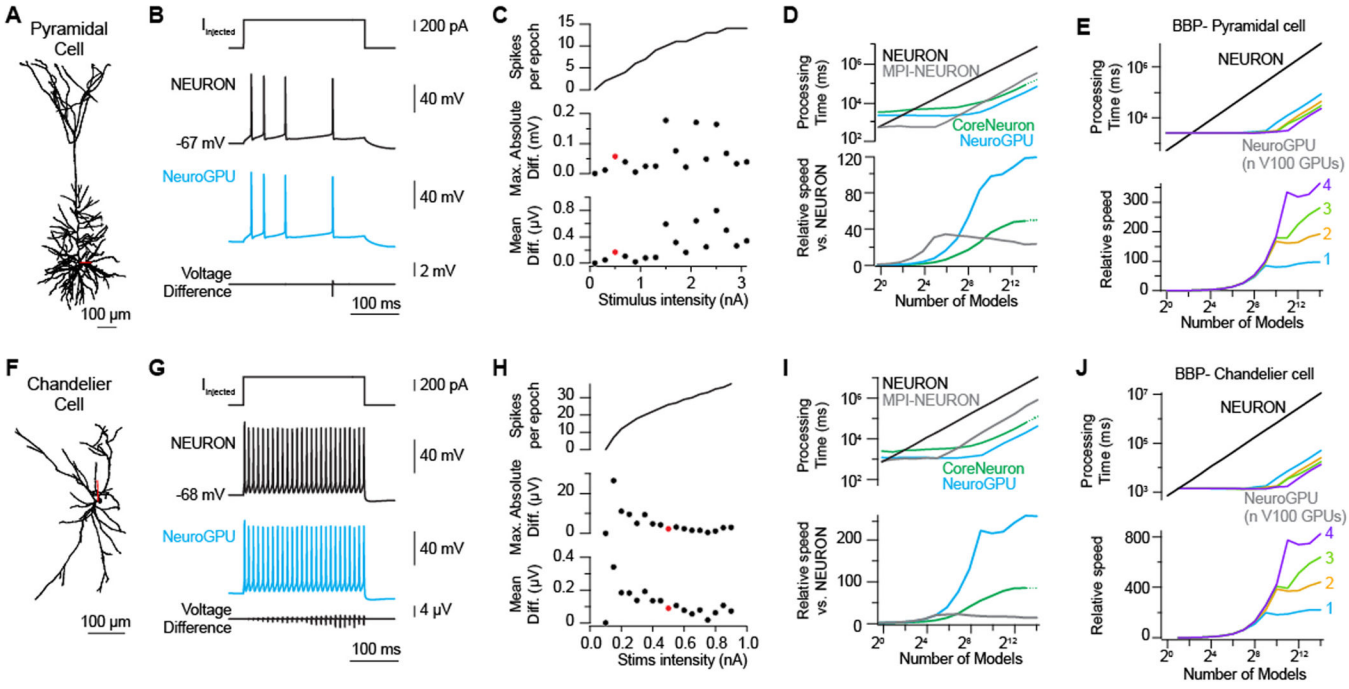

Fig. 2.

NeuroGPU reduces simulation run-time of complex neurons by orders of magnitude without compromising accuracy.NeuroGPU reduces simulation run-time of complex neurons by orders of magnitude without compromising accuracy. A: Morphology of a BBP portal layer 5 neocortical pyramidal cell (Ramaswamy et al., 2015). Dendrite in black, axon in red. B: Top: injected current at the soma. Middle: NEURON voltage response as recorded at the soma. Cyan: NeuroGPU response as recorded at the soma. Bottom: difference in voltage between NEURON and NeuroGPU. C: Top: APs generated per current injection intensity in the soma. Middle, bottom: Peak and average voltage difference between the voltage response in NEURON and NeuroGPU. Red circles denote examples in B. D: Top: Comparing PC model runtimes for the different simulators: black – NEURON, grey MPI-Neuron (32 processors) green – CoreNeuron, blue – NeuroGPU. X-axis in log2 scale, Y-axis in log10 scale. Bottom: Speedup compared to NEURON. Note that the run time for 2^{14} neurons using CoreNeuron is extrapolated (see methods). E: Top: Runtime for the model on 1–4 GPUs (Tesla V100) on the same node. Bottom: Speedup compared to NEURON. F–J: Same as A-D, but for the chandelier cell model.

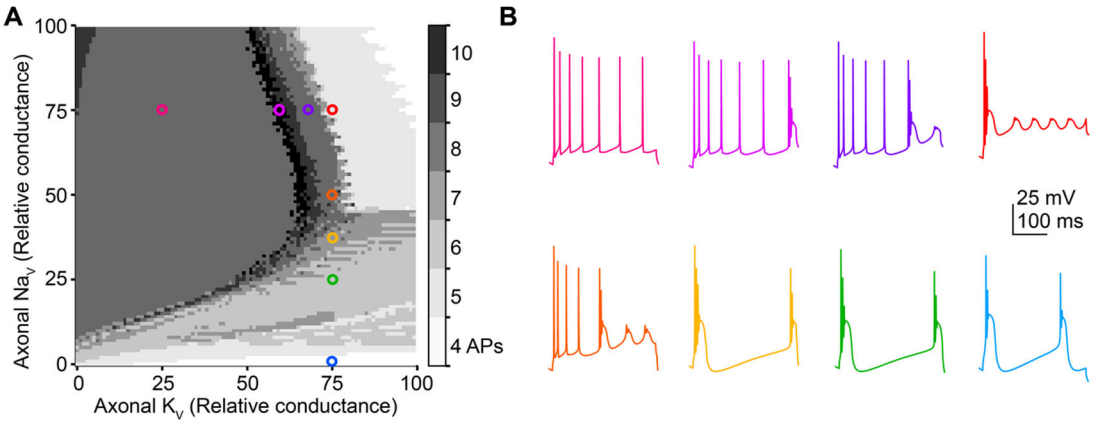

Fig. 3.

NeuroGPU enables rapid exploration of parameter space in complex pyramidal neuron model. A: Each point in the grid represents the number of APs in the relevant model. Points on the axis represent the varied conductances of Na_v and K_v at the axon in the range of [0,10] and [0,20] S/cm^2, respectively. B: Example voltage responses for chosen models from A. Colors match to the corresponding model location.

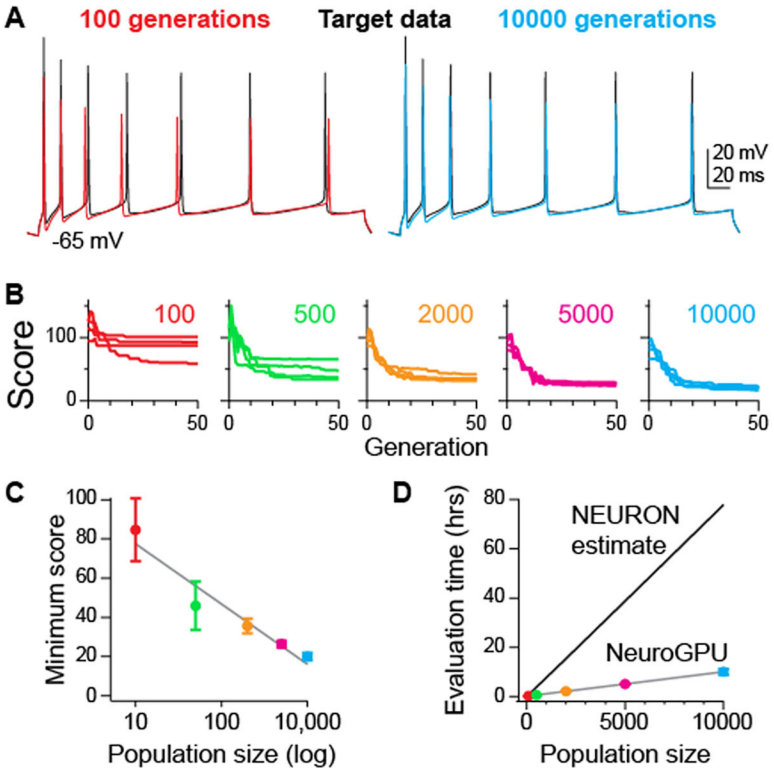

Fig. 4.

NeuroGPU accelerates evolutionary optimization for fitting models to neuronal data. A: Voltage traces obtained from optimization (worst case from population of 100: red; best case from population of 10,000: cyan) compared to ground truth (black). B: Optimizations examples using DEAP with different sizes of populations. Four Optimizations with different random starting population over 50 generations. Y axis is the error from the target voltage as described in the methods section. Lower values denote less error from target data. C: Comparing runtimes for optimizations using NeuroGPU and NEURON (linearly extrapolated from 5 generations). Circles are color coded for population size as in A, and represent mean ± SEM. D: Best score in each optimization in A. Circles and error bars as in C.

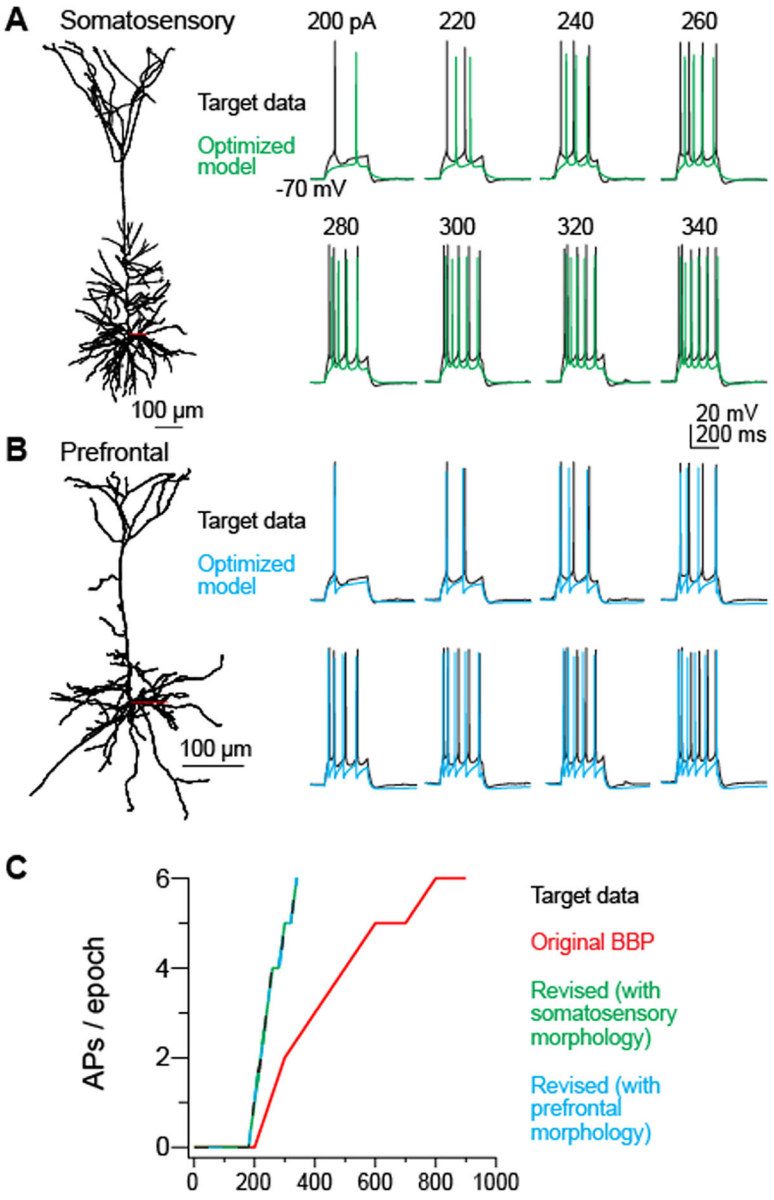

Fig. 5.

NeuroGPU fits BBP PC model to empirical data. A: Left: morphology of L5 thick-tufted pyramidal neuron from somatosensory cortex (Ramaswamy et al., 2015). Right: NeuroGPU fits (green traces) the L5 PC model to empirical data recorded from an L5 prefrontal cortex pyramidal neuron of a mouse (Black Traces) with different stimulus intensity 200–340pA (Spratt et al., 2019). B: Left: Prefrontal-cortex layer 5 pyramidal neuron morphology (Ascoli et al., 2007; Yin et al., 2018). The morphology of BBP PC model was modified to that of the prefrontal neuron and fitted to the same data as in A. C: Number of APs per 300 ms of step stimulation for different models of layer 5 pyramidal neuron. Note that both revised models overlap target data.