bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Title: NeuroGPU, software for NEURON modeling in GPU-based hardware

Authors: Roy Ben-Shalom[1,2], Nikhil S. Artherya[3], Christopher Cross[1?], Hersh Sanghevi[3], Alon Korngreen[4,5], Kevin J. Bender[1,2].

Affiliations:

1. Weill Institute for Neurosciences, University of California, San Francisco, San Francisco CA, USA
2. Department of Neurology, University of California, San Francisco, San Francisco, CA, USA
3. Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley CA, USA
4. The Leslie and Susan Gonda Multidisciplinary Brain Research Center, Bar-Ilan University, Ramat-Gan Israel
5. The Mina and Everard Goodman Faculty of Life Sciences, Bar-Ilan University, Ramat-Gan Israel

Lead contact: roy.benshalom@ucsf.edu

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

18 **Abstract:**

19 Generating biologically detailed models of neurons is an important goal for modern
20 neuroscience. Unfortunately, constraining parameters within biologically detailed models can
21 be difficult, leading to poor model predictions, especially if such models are extended beyond
22 the specific problems for which they were designed. This major obstacle can be partially
23 overcome by numerical optimization and detailed exploration of parameter space. These
24 processes, which currently rely on central processing unit (CPU) computation, are
25 computationally demanding, often with exponential increases in computing time and cost for
26 marginal improvements in model behavior. As a result, models are often compromised in scale
27 given available CPU-based resources. Here, we present a simulation environment, NeuroGPU,
28 that takes advantage of the inherent parallelized structure of graphics processing unit (GPU) to
29 accelerate neuronal simulation. NeuroGPU can simulate most of biologically detailed models
30 from commonly used databases 1-2 orders of magnitude faster than traditional single core CPU
31 processors, even when implemented on relatively inexpensive GPU systems. Thus, NeuroGPU
32 offers the ability to apply compartmental, biologically detailed, modeling approaches with
33 supercomputer-level speed at substantially reduced cost.

34

35

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

36 **Introduction:**

37 Detailed numerical models, faithfully capturing neuronal complexity, are invaluable for
38 simulating the behavior of realistic neural networks (Einevoll et al., 2019). Generating models
39 that accurately recapitulate neuronal activity often requires one to tune individual model
40 parameters. This process can be aided by iterative rounds of parameter exploration and
41 optimization that aim to minimize the differences between empirical data targets and their
42 associated models. These processes can be computationally demanding. Indeed, each linear
43 improvement in model accuracy requires an exponential increase in computational resources
44 (Nocedal and Wright, 2006; Gurkiewicz and Korngreen, 2007) Thus, model optimization is often
45 done on supercomputers that parallelize these computations across central processing unit
46 (CPU) clusters. Unfortunately, due to the cost of constructing and running such supercomputing
47 clusters, such efforts are typically restricted to large consortia, such as the Blue Brain Project
48 (BBP) (Markram et al., 2015) and the Allen Institute (Gouwens et al., 2018). For more restricted
49 budgets, simulations must typically be compromised in scale or complexity to produce results
50 within budget and within a reasonable time frame.

51 In the past 10 years, graphical processing units (GPUs) have emerged as an alternative to
52 CPU-based clusters that may offer comparable levels of performance at substantially reduced
53 cost. GPUs utilize streaming multiprocessors with multiple simple cores that allow for
54 distributed, parallelized computing. With software optimized for distributed computing, GPU-
55 based applications can often outperform CPU-based applications in processing speed and cost
56 (Payne et al., 2010). Today, GPUs are widely used in scientific fields like molecular dynamics
57 (Go et al., 2012; Salomon-Ferrer et al., 2013) and climate modeling (Prein et al., 2015), and are
58 the computational engine for most modern artificial intelligence applications (Schmidhuber,
59 2015). In neuroscience, GPUs are currently being used to accelerate complex imaging dataset
60 processing (Eklund et al., 2013), spiking neural network analysis (Fidjeland and Shanahan,
61 2010), and clustering of activity from *in vivo* extracellular electrophysiological experiments
62 (Pachitariu et al., 2016). Despite these advances until recently that two frameworks for
63 simulating biophysical neuronal networks (Akar et al., 2019; Kumbhar et al., 2019) were
64 published, relatively little effort has been made to leverage GPU-based architecture for
65 biophysically detailed neuronal simulation.

66 Here, we describe NeuroGPU, a computational platform optimized to exploit GPU architecture
67 to dramatically accelerate simulation of neuronal compartmental models. To do so, we

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

68 developed new approaches to parallelize compartmental models, utilizing the GPU-based
69 programming language CUDA to optimize memory handling on GPUs. This resulted in
70 simulation speedups of up to 200-fold on a single GPU and up to 800-fold using a set of 4
71 GPUs. Building on our previous efforts (Ben-Shalom et al., 2013), we developed an intuitive
72 user interface that can import most compartmental models in ModelDB or the BBP portal.
73 Further, we provide methods to explore model parameter space and to optimize models with
74 evolutionary algorithms (DEAP). NeuroGPU therefore provides an open-source platform useful
75 for neuronal simulation with increased speed and reduced cost.

76

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

77 **Results:**

78 **General overview**

79 Our primary goal for NeuroGPU was to improve compartmental modeling speed with relatively
80 low-cost hardware and provide an interface to port NEURON models from public databases
81 such as ModelDB and the BBP portal. Toward that end, we utilized the same basic structure as
82 NEURON, including the use of hoc and mod files that define all aspects of the compartmental
83 model. To increase simulation speed, we focused primarily on parallelizing the most
84 computationally intensive aspect of NEURON simulations in GPU architecture. NEURON
85 calculates the voltages of each segment of the model by solving a system of differential
86 equations that describes current flow in each compartment. Within NEURON, this differential
87 equation system is represented within a tri-diagonal matrix (Hines, 1984). Typically, matrix
88 elements for neighboring compartments are solved in serial, as current flow in one compartment
89 is interdependent on flow in neighboring compartments. We and others have previously
90 developed methods to solve this tri-diagonal matrix in parallel across GPUs, despite the
91 interdependence of current flow across compartments (Hines et al., 2008, n.d.; Ben-Shalom et
92 al., 2013). At that time, the method was implemented only for Hodgkin-Huxley models (Ben-
93 Shalom et al., 2013). Here, we extended this method to support a wider range of models,
94 including most models available in ModelDB and the Blue Brain Project (BBP) repository. This is
95 implemented in Python, with an iPython Graphical User Interface (GUI).

96 **Porting NEURON simulations to NeuroGPU**

97 NeuroGPU simulations begin by importing NEURON's mod and hoc files via a GUI developed in
98 iPython notebook (Figure 1). The user must input a file containing model stimulation, which
99 includes temporal aspects of the model and command currents delivered at a prescribed
100 location. Furthermore, all free parameters, such as channel properties, must be described (Fig.
101 1B). These import components are translated into CUDA code, termed kernels, that can run on
102 the GPU via the python script "extractmodel.py" (Fig. 1C). This script first takes runModel.hoc
103 and loads it into NEURON, not to run simulations, but rather to query NEURON for model
104 properties needed for subsequent porting to NeuroGPU, including compartment names and the
105 tri-diagonal matrix (F-Matrix). Then, the script iterates over the .mod files in the directory, parses
106 them and creates relevant kernels for each mechanism described. Mechanism kernels are
107 written to the AllModels.cu in similar structure as described previously (Hines and Carnevale,

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

108 2000; Carnevale and Hines, 2006), iterating over all compartments defined in the model. A new
109 hoc file is created to register mechanism values, which are stored in AllParams.csv and inserted
110 in each compartment. Finally, the script writes code translated to CUDA in NeuroGPU.cu and
111 packages the application to run on either Windows or Unix. After compiling the code, an
112 executable is created that reads the AllParams.csv and the stimulation and runs the model on
113 the GPU.

114 **NeuroGPU implementation**

115 We used CUDA to implement NEURON-based modeling using GPUs. CUDA is an extension of
116 the C programming language that enables computation on the GPU (Nvidia, 2018). CUDA
117 kernels which are procedures running on the GPU can be invoked from either the GPU or CPU.
118 To invoke a kernel from the CPU, one must specify the number of parallel threads used.
119 Threads, which allow for parallelization on the GPU, are organized into blocks, with each thread
120 occupying a specific address within that block (idx.x, idx.y). GPUs are structured to operate well
121 when computing 32 parallel threads, a computing structure termed a warp (Nvidia, 2018).
122 Therefore, we structured NeuroGPU to utilize 32 threads in the x dimension, corresponding to
123 individual morphological segments within the model. For a given model with more than 32
124 segments, individual threads are responsible for calculating every 32nd segment. For example,
125 thread #1 would calculate segments 1, 33, 65, … 31N+1.

126 Complex neuronal models, including many described in the BBP (Hay et al., 2013; Ramaswamy
127 et al., 2015), are memory intensive. GPUs have several forms of memory that have tradeoffs in
128 terms of their size and relative speed that make them ideal for certain aspects of model
129 processing and impractical for others. GLOBAL memory is the largest physical memory space
130 available on the GPU but is relatively slow. Here, we use GLOBAL memory to store the largest
131 data structures associated with a given model, in part because they simply cannot be held by
132 other memory structures. SHARED memory is far faster, shared among the whole GPU block,
133 but limited to 48 kilobytes. This makes it ideal for storing the tridiagonal matrix, as this matrix is
134 the most accessed data structure within NeuroGPU. CONSTANT memory, which is a 64
135 kilobyte block of fast, read-only memory, is used to hold constant data structures, including the
136 order in which the tri-diagonal matrix is solved in parallel (Ben-Shalom et al., 2013). Lastly,
137 REGISTER\LOCAL memory is the fastest memory available on the GPU but is limited to
138 maximum of 63 registers per thread and a total of 16 kilobytes of memory shared across the
139 entire block. It is used to store local variables necessary for the course of the simulation.

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

140 To determine how best to utilize GPU parallel processing, we examined two ways in which to
141 simulate compartmental models on the GPU. In both cases, the GPU is responsible for updating
142 ionic currents from established mechanisms, solving the tridiagonal matrix, and updating model
143 states and voltages at each time step. In the first configuration, termed SingleKernel, we
144 computed all the time steps of each simulation in one kernel on the GPU, largely because this
145 would limit the amount of time performing the relatively slow step of transferring memory
146 between the GPU and CPU. In this case, the transfer is done only once and during the
147 simulation the GPU communicates with the CPU only to transfer voltages reported at the
148 recording electrode site. Alternatively, we also created a SplitKernel condition, in which the
149 simulation is split into many small kernels that are invoked every single time step. Data are then
150 registered back to the CPU and the next time step is run in serial. This approach may be
151 advantageous if memory transfer between the GPU and CPU is not the rate-limiting step.
152 Furthermore, in this case the GPU can also optimize computing timing by queueing certain
153 steps for execution while other memory is being transferred. Both the SingleKernel and
154 SplitKernal configuration were assessed in all cases reported below.

155 **Benchmarking**

156 To determine how NeuroGPU performs relative to NEURON, we benchmarked it for relative
157 speed and accuracy across different conditions: CUDA implementation, hardware configurations
158 and across a range of models. We first compared NeuroGPU performance with a single GPU to
159 NEURON implemented on a single CPU core.

160 We began with a simple model of a soma and single dendritic branch that has 64 segments in
161 total (Figure 1A), each containing a single external mechanism pas.mod that describes passive
162 current flow. This model was stimulated with a simple current step (Fig. 1B). Voltage
163 discrepancies that never exceeded 0.4 μV were observed between NeuroGPU and NEURON
164 when simulation voltage changed rapidly. These discrepancies were due to small differences in
165 timing that likely arise from how numbers are rounded in GPUs vs CPUs (Whitehead, 2011).

166 To benchmark relative speed, we evaluated computing time for multiple instances of the same
167 model. NEURON computation speed scales linearly with the number of simulations, and, for low
168 numbers of models (< 8), outperforms NeuroGPU. By contrast, models implemented on GPUs
169 scale linearly only after saturating all streaming multiprocessors. With NeuroGPU, processing
170 times are quite similar for any simulation incorporating fewer than 128 models, and begin to

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

171 outpace NEURON simulations when >32 simulations are run simultaneously. Relative gains in
172 processing time were noted when 32 to 16,384 models were run simultaneously. These gains
173 were dependent on hardware. For example, implementing NeuroGPU on an NVIDIA TitanXP
174 GPU resulted in 25.2-fold improvements in processing speed, while the same models run on an
175 NVIDIA Tesla V100 were 95.8-fold faster (both implemented in the "SingleKernel"
176 configuration). It is worth noting that TitanXP hardware is relatively low cost (<$1099) and very
177 similar card (NVIDIA GTX-1660) can currently be purchased for less than $300, suggesting that
178 significant improvements in processing speed can be obtained even with modestly priced
179 hardware.

180 More complex neuronal morphology could affect NeuroGPU processing speed. Therefore, we
181 implemented the same passive mechanism on the more complex structure of a neocortical
182 pyramidal neuron. While voltage discrepancies were similarly small in this instance ($< 4x10^{-6}$
183 mV), the relative speedup was lower than with simpler morphology (TitanXP: 15.2x; Tesla V100:
184 58.1x). Thus, while morphology does affect relative speed, NeuroGPU still outperforms CPU-
185 based modeling.

186 In addition to complex morphology, compartmental models typically contain an array of
187 mechanisms that simulate voltage-gated channels or ligand-gated receptors. To assess
188 NeuroGPU performance with such models, we began with a pyramidal model neuron first
189 described by Mainen and Sejnowski (1996). This model has 7 different mechanisms, including
190 voltage-gated sodium, potassium, and calcium channels, and a calcium-dependent potassium
191 channel. As with the passive model described above, we implemented these mechanisms in
192 both simple and complex morphologies (e.g., soma and primary dendrite alone, or complete
193 pyramidal cell morphology). In models with simple morphology, NeuroGPU was 30.3x (TitanXP)
194 or 153.1x (Tesla V100) faster than NEURON, with minimal voltage error (< 4 μV). In pyramidal
195 cell morphology models, NeuroGPU was 45.3x (TitanXP) or 114.2x (Tesla V100) faster than
196 NEURON. In this instance, we observed a relatively large voltage discrepancy of 6.6 mV. This
197 discrepancy occurred during the last AP within a burst and was due largely to a shift in the
198 timing of this AP (Fig. 4G). Indeed, we were able to reduce this error ~6x by interpolating the
199 data and shifting the timing of this AP by ¼ of a timestep.

200 While the Mainen and Sejnowski model can generate physiologically-realistic spiking activity,
201 these APs occur over a relatively narrow range of stimulus intensities. Outside this range the
202 model is either subthreshold or enters depolarization block. As a result, we found this model to

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

203 be impractical for benchmarking NeuroGPU across a range of stimuli. Therefore, we tested
204 NeuroGPU on more recently developed models from the Blue Brain Project portal. Here, we
205 used two models: one of a layer 5 pyramidal neuron (BBP_PC, see Methods for specific model)
206 and one of a layer 5 chandelier interneuron (BBP_CC). Models were interrogated with a range
207 of stimulus intensities to determine relative differences between NeuroGPU and NEURON (Fig.
208 5). Similar to Mainen and Sejnowski, voltage differences were small (maximum differences:
209 <0.2 mV) and were most commonly observed when voltage was changing markedly between
210 time steps (Fig. 5C, G).

211 As with other models (Fig. 3, 4), implementing NeuroGPU on faster GPUs decreased
212 processing time (Fig. 5D, H). Interestingly, CUDA has been recently updated to allow for
213 memory sharing across GPUs, which could be leveraged to decrease processing time further.
214 To test this, we connected up to 4 Tesla V100 GPUs together and measured speedup on both
215 BBP models displayed in Figure 5. As expected, adding more GPUs increased the overall
216 processing capacity, and we noted shifts in the number of models that could be handled
217 simultaneously before reaching maximum GPU utilization (Fig. 6). Furthermore, speedup was
218 almost 2 orders of magnitude faster relative to NEURON.

219 **Profiling**

220 To better understand why NeuroGPU accelerated some models more than others, we used the
221 NVIDIA profiler to monitor GPU utilization. Further, we tested two different memory handling
222 configurations—SingleKernel and SplitKernel—to determine how best to utilize GPU parallel
223 processing. In both cases, the GPU is responsible for updating ionic currents from given
224 mechanisms, solving the tridiagonal matrix, and updating model states and voltages at each
225 time step.

226 We found that configuring NeuroGPU in SingleKernel mode produced the fastest runtimes in all
227 models tested (Table 1), and had higher GPU utilization levels. This indicates that, for most
228 models, memory transfer between GPU and CPU is rate-limiting, and models run most
229 efficiently when the majority of calculations are isolated on the GPU. Nevertheless, the highest
230 utilization values were ~10% in the SingleKernel configuration (3.8% in SplitKernel), suggesting
231 that additional memory optimizations could be leveraged in future iterations of NeuroGPU.

232

233

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

234 **Benefits of using NeuroGPU for parameter space exploration and genetic optimization**

235 Neuronal simulations are often tested over a range of parameter values to both explore the
236 range of output generated and to optimize models to best fit empirical data (Druckmann et al.,
237 2007; Van Geit et al., 2008; Keren et al., 2009; Gouwens et al., 2018). These simulations
238 essentially run the same model repeatedly with small differences in underlying parameters,
239 making them ideal for parallelization with NeuroGPU. Indeed, relative speedups would be
240 identical to situations considered above (Fig. 3-6) and depend simply on the number of
241 parameter sets used. Based on this, we developed a GUI that streamlines parameter space
242 exploration in NeuroGPU.

243 To provide an example of parameter space exploration, we examined neuronal output in the
244 BBP_PC model when co-varying the density of the axonal fast inactivating sodium channel and
245 axonal slow-inactivating potassium channel over a range of 0 to 10 and 0 to 20 S/cm^2,
246 respectively. Total spike output and select single traces are shown in Figure 7. As expected,
247 increasing sodium conductance allowed models to generate more APs until sodium
248 conductance was so high that models entered depolarization block. Similarly, reducing
249 potassium conductance produced comparable results. Interestingly, certain combinations of
250 sodium and potassium conductance concentrations produced bursting phenotypes
251 characterized by high-frequency APs riding atop long-duration depolarizations. These
252 presumably reflect parameter ranges that then interact with other ion channels in the model
253 (e.g., Ca_V3 channels) that promote such burst dynamics.

254 To implement genetic optimization within NeuroGPU, we integrated the DEAP (Distributed
255 Evolutionary Algorithms in Python) package (Gagn, 2012). Genetic algorithm success lies in the
256 balance between exploration of the whole parameter space and the exploitation of specific
257 areas that seem promising. For this, large sample populations are ideal, as this allows for
258 effective and broad parameter space exploration. NeuroGPU is more efficient when many
259 instances are running in parallel, allowing for more effective application of genetic algorithms.

260 Genetic optimization was tested here by fitting model-generated voltages to a single voltage
261 epoch containing APs that was generated by the default values present in the BBP_PC model.
262 We then determined how close different optimization sets could come to identifying these
263 original parameter values. Optimization began with different population sizes comprised of 100
264 to 10,000 individual parameter sets with random initial values (Fig. 8A). These populations were

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

265 run in four independent trials, each for 50 generations, and the difference between the naïve
266 model and ground-truth model was compressed to a single score value (see Methods). For
267 these scores, lower values indicate less difference between the two cases.

268 Scores improved for each of these populations, but the variance across trials and the overall
269 score were markedly affected by the population size, with score decreasing in a near-linear
270 fashion with each doubling of population size (Fig 8C). These score improvements were
271 paralleled by a decrease in total processing time. For example, optimization with 10,000
272 individual parameter sets ran 7.7x faster on NeuroGPU than NEURON (Fig. 8D; 10 vs 77 hours,
273 respectively). While these are significant improvements in simulation speed, they are relatively
274 modest compared to those observed in other conditions (Fig. 5), likely because current versions
275 of NeuroGPU require NEURON to load the simulation and generate parameter values. This step
276 is currently done using the CPU. Whether it is possible to parallelize this step will be explored in
277 future versions of NeuroGPU.

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

278 **Discussion:**

279 In this work, we implemented a simulation environment to run single neuron compartmental
280 models on GPUs. Based on our previous efforts (Ben-Shalom et al., 2013), we designed a user-
281 friendly environment that enables one to port multi-compartmental models for implementation
282 with CUDA. NeuroGPU was developed to be interoperable with NEURON (Cannon et al., 2007),
283 thereby allowing anyone with expertise in the NEURON environment access to GPU-based
284 acceleration. Towards this goal, we developed a platform to easily port NEURON models from
285 either ModelDB or the BBP portal (Ramaswamy et al., 2015; McDougal et al., 2017) using a
286 iPython notebook-based graphical user interface (GUI). We further developed GUIs for creating
287 stimulation protocols, parameter exploration, and genetic optimization. By taking advantage of
288 parallel processing inherent to GPUs, we were able to accelerate simulations dramatically, in
289 some cases by almost two orders of magnitude.

290 NeuroGPU accelerates compartmental modeling largely through parallelization of matrix
291 calculations. Solving the tridiagonal matrix is the most computationally demanding aspect of
292 compartmental model simulations (Hines, 1984; Hines et al., 2008; Ben-Shalom et al., 2013).
293 Therefore, we took advantage of fast, on-GPU memory and controlled the timing of calculations
294 and memory transfers to optimize the use of computational resources (Volkov and Demmel,
295 2008; Ben-Shalom et al., 2013; Nvidia, 2018). Resulting speedups depended primarily on
296 neuronal morphology, and in general we found the NeuroGPU performed best when processing
297 anatomically complex cases. Even in these cases, overall GPU utilization was limited by
298 execution dependencies, where one aspect of GPU processing could not proceed until another
299 aspect either transferred or processed its own memory. In the future, these dependencies may
300 be further reduced through either dynamic parallelization (Zhang et al., 2015) or by increasing
301 instruction level parallelism (ILP) (Volkov and Demmel, 2008). Nevertheless, the current version
302 of NeuroGPU can still accelerate single neuron compartmental simulations by several orders of
303 magnitude.

304 NeuroGPU addresses a major gap in currently implemented GPU-based simulation
305 environments. In addition to NeuroGPU, two other neuronal simulations environments for multi-
306 compartmental models have been implemented using GPUs, CoreNeuron (Hines et al., n.d.)
307 and Arbor (Akar et al., 2019). Both of these environments are designed primarily to accelerate
308 large scale network simulations. NeuroGPU, by contrast, is focused more on exploring the
309 parameter space of single models and optimizing such models to best fit empirical data. As
310 such, NeuroGPU has expanded GUIs for parameter exploration, which allows for quick

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

311 assessment of how changes in ion channel density across compartments affects neuronal
312 excitability (Fig. 7). This approach may be particularly useful to generate testable hypotheses
313 regarding channel distribution with pharmacological manipulations (Keren et al., 2009; Almog
314 and Korngreen, 2014; Mäki-Marttunen et al., 2018), modulation of ion channels (Byczkowicz et
315 al., n.d.), or in disease states where ion channel density is thought to be affected (Migliore and
316 Migliore, 2012; Miceli et al., 2013; Ben-Shalom et al., 2017; Spratt et al., 2019). Furthermore,
317 one could also generate a range of cells with variable channel densities and confirm that their
318 activity is physiologically realistic (e.g., Fig. 7, all cases before generating depolarization block).
319 These conditions could then be used as building blocks for variable activity within neuronal
320 networks (Prinz et al., 2003, 2004; Alonso and Marder, 2019).

321 In addition to parameter exploration, NeuroGPU is designed for extensive model optimization
322 using DEAP. Fitting computational models to empirical data is computationally taxing, and fits
323 typically improve two-fold with each doubling of computational resources. Here, we found that
324 NeuroGPU can accelerate DEAP processing times 8x (Fig. 8). Of note, these speedups
325 compare single GPUs and CPUs. Leveraging multiple GPUs should accelerate this process
326 further.

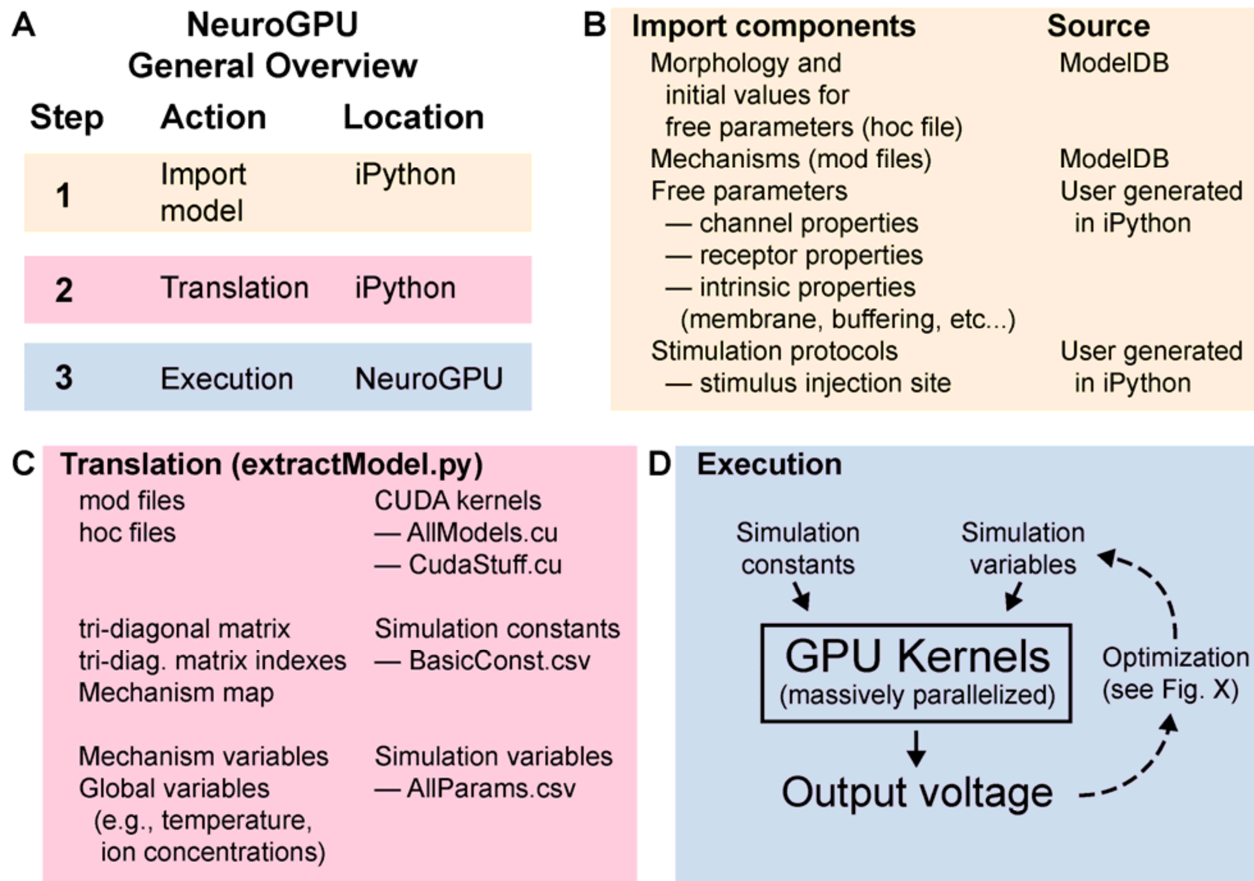
327 Future iterations of NeuroGPU may expand on the strengths and address limitations in using
328 GPUs for compartmental modeling. Ion channels are modeled typically with Markov-based
329 kinetics, or a simpler Markov approximation based on Hodgkin-Huxley type equations.
330 NeuroGPU currently supports Hodgkin-Huxley-based mechanisms only, as we found that
331 implementation of full Markov-based mechanisms on GPUs requires too much shared memory
332 and reduces performance drastically (Ben-Shalom et al., 2012). As with total GPU utilization,
333 improvements in memory handling may improve these cases. Furthermore, GPUs work best
334 when the same instructions are occurring simultaneously on multiple memory addresses. This
335 makes them ideal for iterating through models with identical morphologies and different channel
336 distributions, but less ideal for network models containing a diversity of neuron types. As an
337 intermediate, one could address this limitation by modeling networks containing discrete sets of
338 neurons. For example, a network could contain several compartmental morphology models that
339 each support multiple instances with different channel parameters, similar to the Ring model
340 applied by Arbor (Akar et al., 2019; Kumbhar et al., 2019).

341 In its current state of development, NeuroGPU may help democratize compartmental modeling.
342 While NeuroGPU can support simulations in large clusters using UNIX-based mutli-GPU
343 architectures, it also is ideal for individual laboratories running simulations on Windows-based

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

workstations. Indeed, a workstation with total costs <$3000, when kitted with appropriate GPUs, can out-perform large CPU-based clusters. This could help broaden the use and utility of computational modeling by bringing supercomputer-level processing power to a large range of academic settings.

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



351

352 **Figure 1:** ***NeuroGPU overview and flowchart***

353 **A:** Overview of the general workflow in NeuroGPU: The user ports a model via the iPython GUI
354 and customizes the simulation (panel B). NeuroGPU translates the model to CUDA code
355 that can run on the GPU and compiles executable code.

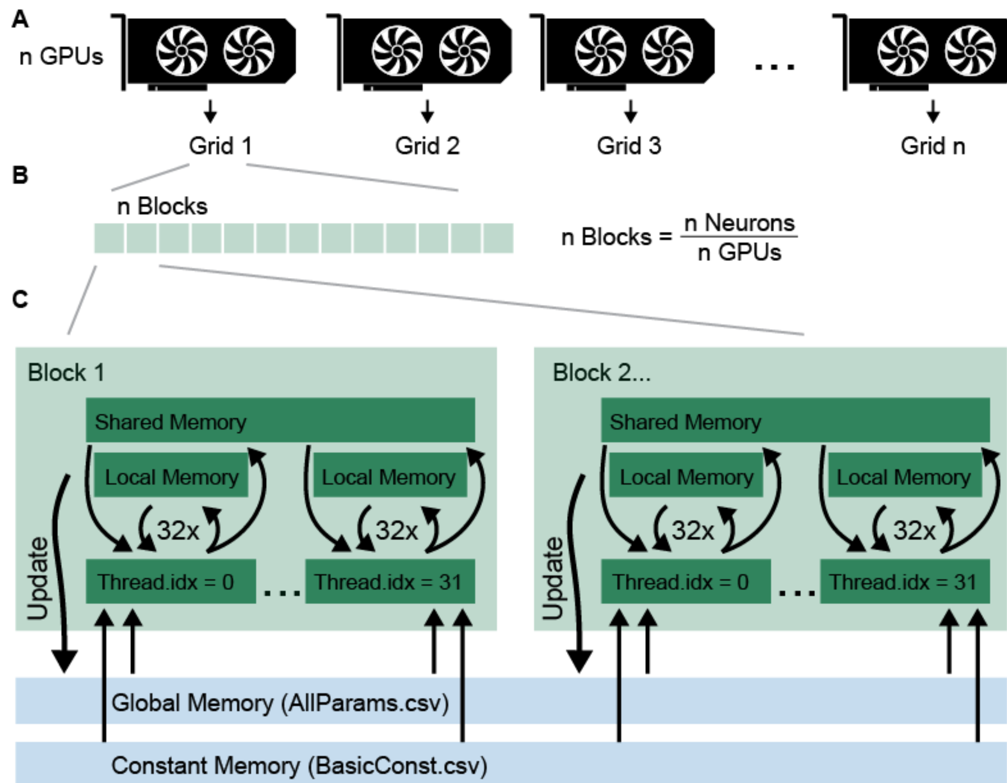
356 **B:** Sources for model components: The morphology and model's properties are described in
357 the hoc file. Additional mechanisms such as ion-channels are described in .mod files. The
358 stimulation protocols can be either imported or can be generated with our provided GUI

359 **C:** Import to NeuroGPU is done by the extractModel.py script. It translates mod files to GPU
360 kernels (see methods), which are written to AllModels.cu, and updates the course of the
361 simulation at CudaStuff.cu. extractModel.py writes to the BasicConst.csv the tri-diagonal
362 matrix and mechanism map, which indicates the mechanisms for each compartment. Finally,
363 extractModel.py writes all the mechanism parameters to AllParams.csv.

364 **D:** After extractModel.py terminates, it creates NeuroGPU.exe. When NeuroGPU is invoked it
365 reads the input files and runs the simulations for the different instances of the model and
366 writes their voltages output to a file. When NeuroGPU is used for optimization, new
367 instances of the models are created each iteration, and only AllParams.csv is updated via a
368 python script.

369

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



370
371
372 **Figure 2:** ***NeuroGPU CUDA implementation***

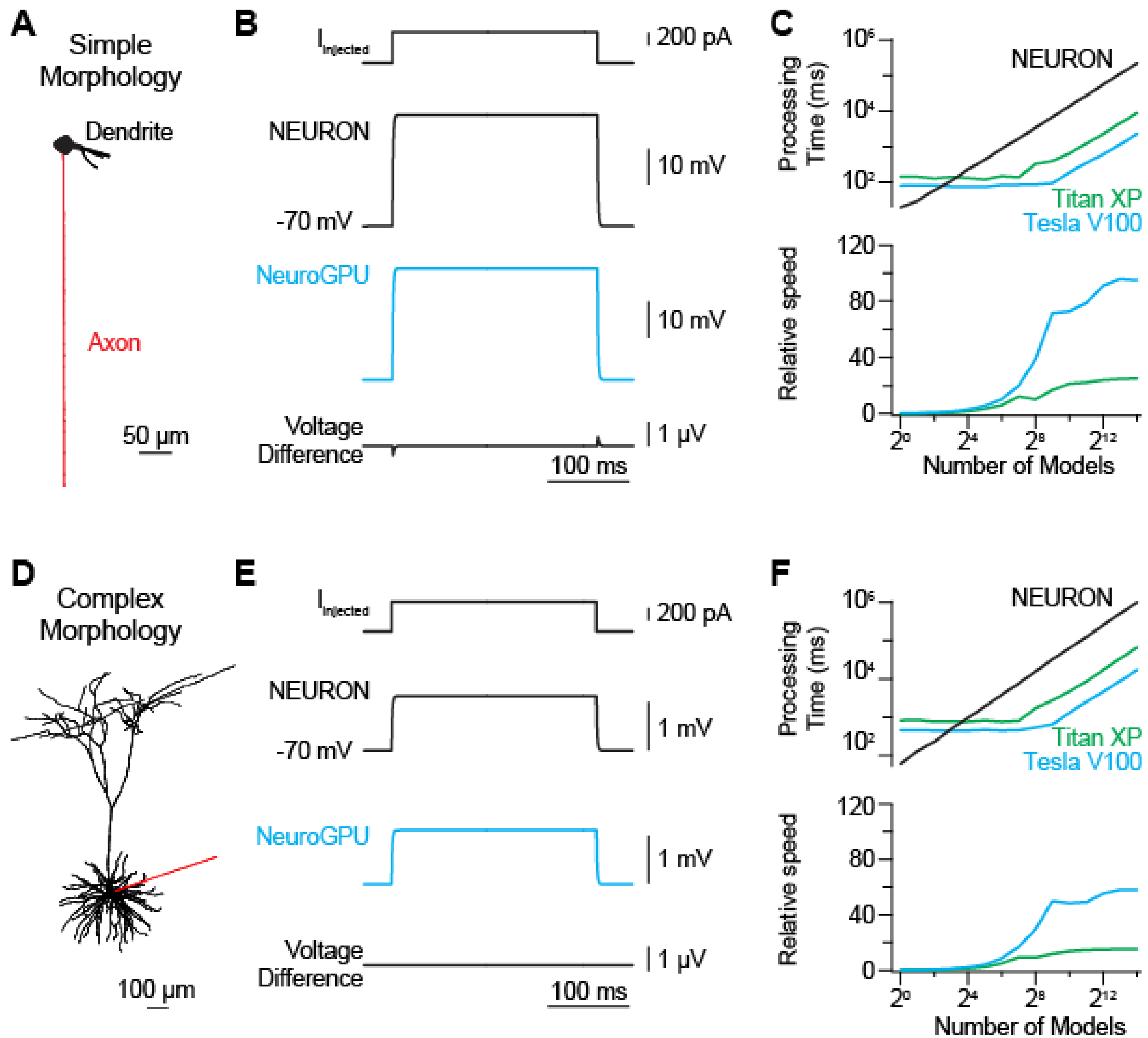
373 **A:** NeuroGPU can be run on multiple GPUs; each GPU will run a separate grid of
374 block/neurons (Nvidia, 2018).

375 **B:** Grids are distributed in blocks, with each block representing an instance of a model. The
376 number of blocks in a grid is set by the number of model instances that will be simulated on
377 an individual GPU.

378 **C:** A block is the basic simulation unit upon which 32 threads each update the memory in an
379 ILP manner (see Methods). Global memory, which can be accessed by all blocks, stores
380 mechanism parameters for every compartment. Constant memory, which is limited in size,
381 stores the simulation constants such as the tri-diagonal matrix and the mechanism map.

382

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



383

384 **Figure 3: *Passive model simulations***

385 **A:** Simple morphology with artificial axon. This model contains passive channels (pas.mod) in
386 all compartments.

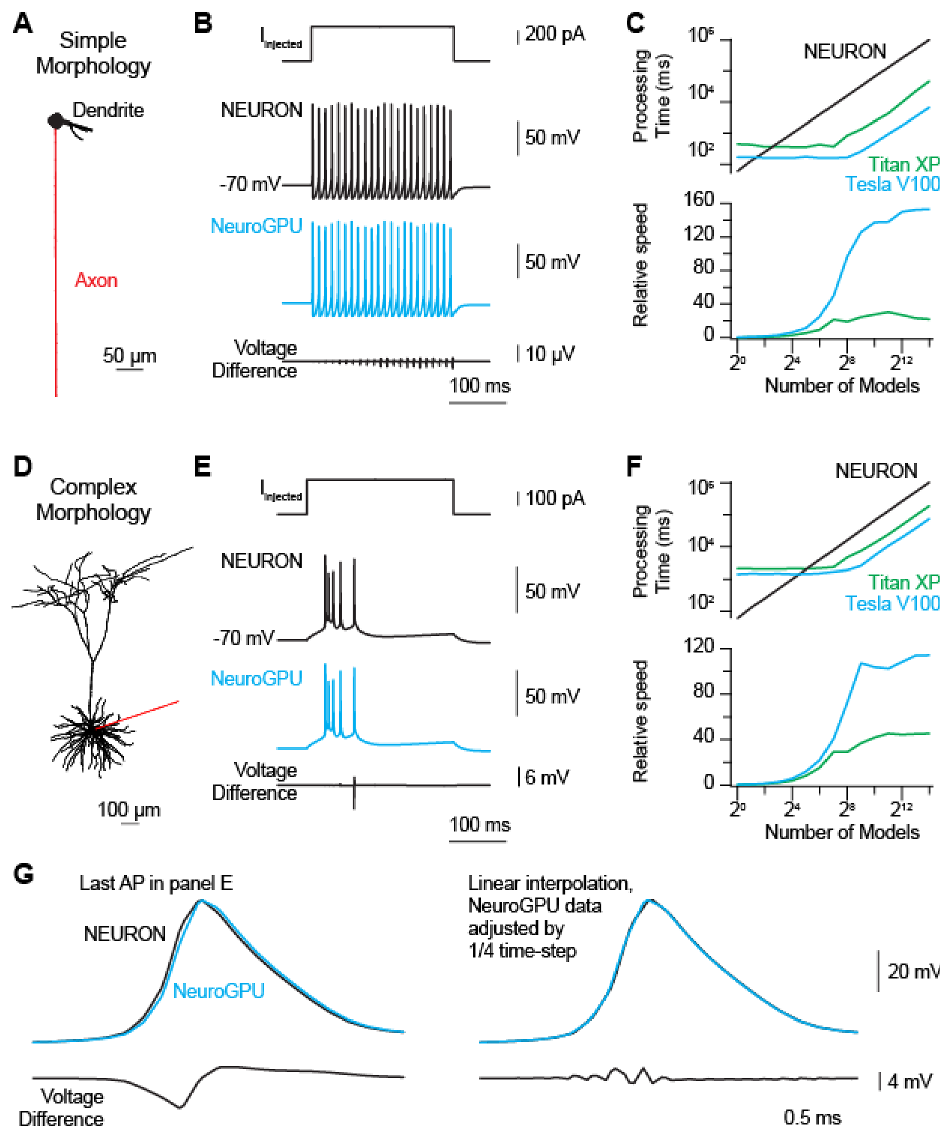
387 **B:** Top: injected current at the soma Middle: NEURON voltage response as recorded at the
388 soma. Blue: NeuroGPU response as recorded at the soma. Bottom: difference in voltage
389 between NEURON and NeuroGPU.

390 **C:** Top: Runtimes for the model using the different architectures: black – NEURON, green –
391 NeuroGPU on TitanXP, blue – NeuroGPU on TeslaV100. X-axis in log2 scale, Y-axis in
392 log10 scale. Bottom: Speedup compared to NEURON.

393 **D-F:** Same as A-C, but for complex morphology from (Mainen and Sejnowski, 1996).

394

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



395

396 **Figure 4:** ***Mainen and Sejnowski model neuron simulations***

397 **A:** Simple morphology with artificial axon and active and passive components distributed as in
398 (Mainen and Sejnowski, 1996)

399 **B:** Top: injected current at the soma. Middle: NEURON voltage response as recorded at the
400 soma. Cyan: NeuroGPU response as recorded at the soma. Bottom: difference in voltage
401 between NEURON and NeuroGPU.

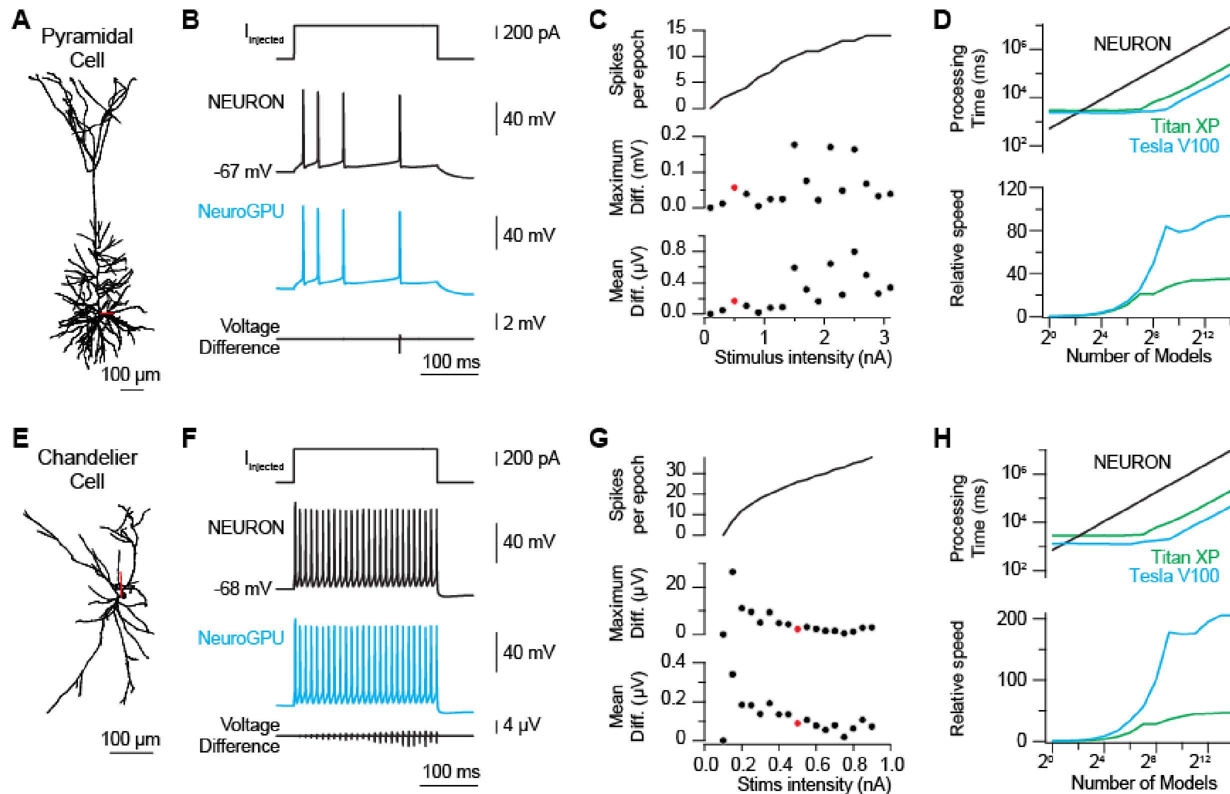
402 **C:** Top: Runtimes for the model using the different architectures: black – NEURON, green –
403 NeuroGPU on TitanXP, blue – NeuroGPU on TeslaV100. X-axis in log2 scale, Y-axis in
404 log10 scale. Bottom: Speedup compared to NEURON.

405 **D-F:** Same as A-C, but for neocortical layer 5 pyramidal cell morphology, as in (Mainen and
406 Sejnowski, 1996).

407 **G:** Last AP in panel E, with expanded timebase, highlighting differences in voltage during the
408 rising phase of the AP. Voltage differences are minimized by linearly interpolating the data
409 4-fold and advancing NeuroGPU simulation by ¼ time-step.

410

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



411

412 **Figure 5:** ***BBP portal model simulations***

413 **A:** Morphology of a BBP portal layer 5 neocortical pyramidal cell (Ramaswamy et al., 2015).
414 Dendrite in black, axon in red.

415 **B:** Top: injected current at the soma. Middle: NEURON voltage response as recorded at the
416 soma. Cyan: NeuroGPU response as recorded at the soma. Bottom: difference in voltage
417 between NEURON and NeuroGPU.

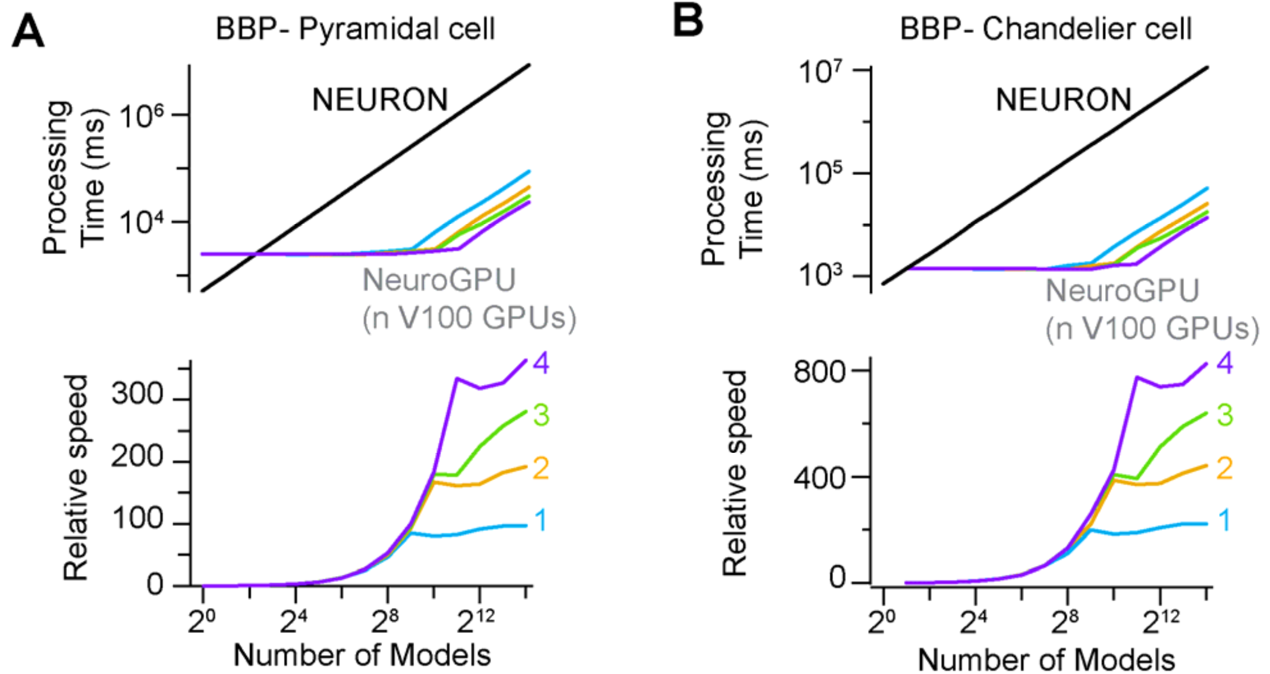
418 **C:** Top: APs generated per current injection intensity in the soma. Middle, bottom: Peak and
419 average voltage difference between the voltage response in NEURON and NeuroGPU. Red
420 circles denote examples in B.

421 **D:** Top: Runtimes for the model using the different architectures: black – NEURON, green –
422 NeuroGPU on TitanXP, blue – NeuroGPU on TeslaV100. X-axis in log2 scale, Y-axis in
423 log10 scale. Bottom: Speedup compared to NEURON.

424 **E-H:** Same as A-D, but for a model chandelier cell.

425

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



426

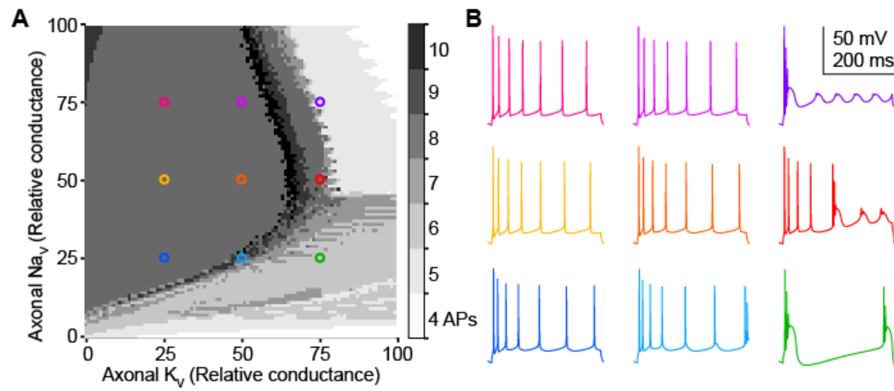
427 **Figure 6: *NeuroGPU simulation on multiple GPUs***

428 **A:** Top: Runtimes for pyramidal cell model using a different numbers of V100 GPUs (cyan – 1
429 orange – 2 green -3 purple – 4). X-axis is in log2 scale and Y-axis is in log10 scale. Bottom:
430 Speedup compared to NEURON.

431 **B:** Same as A, but for chandelier cell model.

432

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.



433

434 **Figure 7:** ***Parameter space exploration in the BBP pyramidal model***

435 **A:** Each point in the grid represents the number of APs in the relevant model. Points on the
436 axis represent the varied conductances of Na_v and K_v at the axon in the range of [0,10] and
437 [0,20] S/cm^2, respectively.

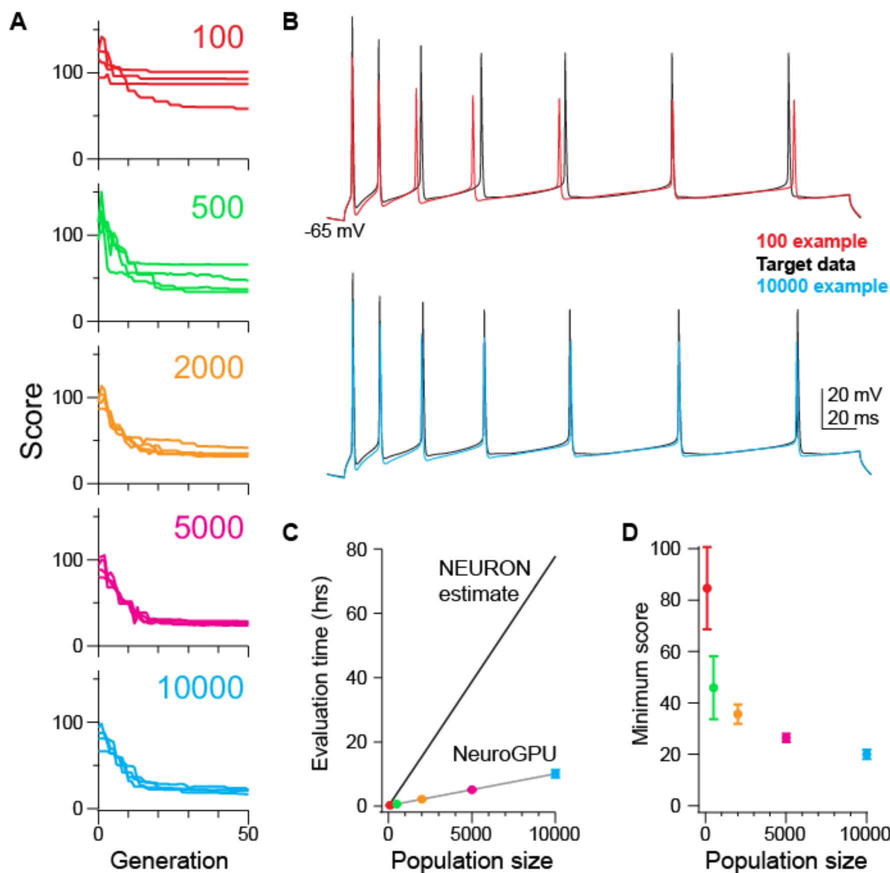
438 **B:** Example voltage responses for chosen models from A. Colors matched to the
439 corresponding model location in A.

440

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

441

442 .



443

444 **Figure 8: *Evolutionary optimization with NeuroGPU***

445 **A:** Optimizations examples using DEAP with different sizes of populations. Four Optimizations
446 with different random starting population over 50 generations. Y axis is the error from the
447 target voltage as described in the methods section. Lower values denote less error from
448 target data.

449 **B:** Voltage traces obtained from optimization (worst case from population of 100: red; best case
450 from population of 10,000: cyan) compared to ground truth (black).

451 **C:** Comparing runtimes for optimizations using NeuroGPU and NEURON (linearly extrapolated
452 from 5 generations). Circles are color coded for population size as in A.

453 **D:** Best score in each optimization in A. Circles and error bars are mean ± SEM.

454

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

455 **Table 1:**

Model	Morphology	SingleKernel		SplitKernel	
		Acceleration	Utilization	Acceleration	Utilization
Passive membrane	Soma & apical dendrite	95.8	0.30%	78.8	0.41%
	Pyramidal Cell	58.1	10%	55.2	3.62%
Mainen and Sejnowski	Soma & apical dendrite	153.1	10%	99.5	1%
	Pyramidal Cell	114.2	10%	111.8	3.80%
Blue Brain Project	Pyramidal Cell	93	10%	107	3.70%
	Chandelier Cell	205.6	10%	197.5	1.60%

Acceleration: fold increase in processing speed relative to single core CPU (MODEL)

456 Utilization: Percent of time the GPU is being used

457

458 **Table 2:**

Parameter Name	Base value	Lower Bound	Upper Bound
gNaTa_tbar_NaTa_t	3.137968	0.3137968	31.37968
gNaTs2_tbar_NaTs2_t	0.983955	0.0983955	9.83955
gK_Tstbar_K_Tst	0.089259	0.0089259	0.89259
gIhbar_Ih	0.00008	0.000008	0.0008
gImbar_Im	0.000143	0.0000143	0.00143
gSKv3_1bar_SKv3_1	0.303472	0.0303472	3.03472

459

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

460 **Methods:**

461 **Hardware —** NEURON and TitanXP-based simulations were run on a PC with Intel Core I7-
462 7700K 4.2GHz with 16GB of RAM. Tesla V100-based simulations were run using the NVIDIA
463 PSG cluster. Here, each simulation was run on a single node with Haswell or Skylake CPU
464 cores. For multi-GPU simulations, we used cluster nodes with NVLINK (Li et al., 2019) between
465 the GPUs to enable memory peer-access.

466 **Software —** Simulations were performed in NEURON 7.6 and CUDA 9.1. All scripts were
467 written in Python 3.7. All software is available at https://github.com/roybens/NeuroGPU.

468 **Importing NEURON models —** The python script extractmodel.py (Fig 1) exports NEURON
469 models to NeuroGPU. This script reads all simulation details from runModel.hoc, which is
470 populated using the GUI. NEURON models are described using either hoc or python scripts.
471 The scripts include a morphology that can either be called as a separate file or constructed
472 within the script.

473 **Translating mechanisms to CUDA —** Mechanisms in NEURON are described by NMODL
474 (.mod) files (Hines and Carnevale, 2000), that update the mechanism states every simulation
475 time step. This is done using three different procedures within NEURON that initialize
476 mechanisms (nrn_init), update currents that mechanisms affect (nrn_cur), and then update
477 mechanism states (nrn_state) (Carnevale and Hines, 2006). In NeuroGPU, CUDA kernels are
478 written for each of these procedures using .mod and .c files that are generated by NEURON
479 when running nrnivmodl. Kernels are saved and editable in AllModels.cu and AllModels.h.

480 **Extracting simulation properties from NEURON** — NeuroGPU utilizes NEURON for
481 simulation pre-processing, including generating the mechanism map for mechanism distribution
482 across compartments and exporting the tri-diagonal matrix using the fmatrix(). These are stored
483 in BasicConstSegP.csv. NEURON extracts all parameters for cable equations and mechanism
484 values within each compartment to AllParams.csv. External stimulation delivery location,
485 intensity, and timecourse are written in stim.csv. Resting membrane potential and number of
486 time steps in the simulation are written in sim.csv.

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

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

491 **Benchmarking** — All benchmarking was done compared to NEURON 7.6 running in a single
492 thread. The morphology was adjusted to have one segment per compartment in both NEURON
493 and NeuroGPU comparison. Simulation runtimes were compared without hard drive read/write
494 file steps, as these aspects depend more on hard drive properties than CPU/GPU comparisons.

495 **Multi-compartmental models** — NeuroGPU performance was tested with 4 different models:
496 1) A passive model, utilizing passive channels described in NEURON distribution pas.mod file.
497 These channels were distributed on both simple and complex morphologies (see Fig. 3A, D)
498 (Mainen and Sejnowski, 1996). The simple morphology was based on the simple morphology
499 described in Mainen and Sejnowski, with compartments reduced to 32, as this is the minimum
500 number of compartments required for NeuroGPU-based simulations.

501 2) The Mainen and Sejnowski (1996) model, with channels distributed on the same complex
502 and simple morphologies (Fig 4). Channels are distributed as in (Mainen and Sejnowski, 1996)

503 3) A pyramidal cell model from the Blue Brain Project portal (Ramaswamy et al., 2015) (Fig 5).
504 BBP_PC refers to the model named L5_TTPC1_cADpyr232_1.

505 4) A chandelier cell model, termed BBP_CC, referring to L5_ChC_dNAC222_1. For this model,
506 the Kdshu2007.mod files were altered to run on NeuroGPU. Specifically, global variables were
507 removed from the neuron block and instead placed in the assigned block (Carnevale and Hines,
508 2006).

509 **Optimization algorithm** — The *eaMuPlusLambda* algorithm from the DEAP package was
510 implemented by modifying the varOR procedure to call NeuroGPU (Rainville et al., 2012).
511 Optimization was performed on the BBP_PC model. For each iteration, the algorithm began with
512 a new population of parameters with values randomly chosen with the range specified in Table
513 2. The model was modified to accept new values from the optimization algorithm (similar
514 changes were necessary to run the parameter space exploration for Figure 7). Target data were
515 generated using the original parameters values described in Table 2. Optimization was targeted
516 to reduce error between target data and test data using both the interspike interval (ISI) and the
517 root mean square (RMS) of the voltage as the error function. Error was reduced to a single
518 variable by weighting these two variables as: 10*ISI + RMS.

519

520

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

521 **Acknowledgments**

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

528

529

530

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

531 **References**

532 Akar NA, Cumming B, Karakasis V, Küsters A, Klijn W, Peyser A, Yates S (2019) Arbor - A
533 Morphologically-Detailed Neural Network Simulation Library for Contemporary High-
534 Performance Computing Architectures. In: Proceedings - 27th Euromicro International
535 Conference on Parallel, Distributed and Network-Based Processing, PDP 2019, pp 274–
536 282.

537 Almog M, Korngreen A (2014) A quantitative description of dendritic conductances and its
538 application to dendritic excitation in layer 5 pyramidal neurons. J Neurosci 34:182–196.

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

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

543 Ben-Shalom R, Keeshen CM, Berrios KN, An JY, Sanders SJ, Bender KJ (2017) Opposing
544 Effects on NaV1.2 Function Underlie Differences Between SCN2A Variants Observed in
545 Individuals With Autism Spectrum Disorder or Infantile Seizures. Biol Psychiatry 82:224–
546 232.

547 Ben-Shalom R, Liberman G, Korngreen A (2013) Accelerating compartmental modeling on a
548 graphical processing unit. Front Neuroinform 7:4.

549 Byczkowicz N, Eshra A, Montanaro J, Trevisiol A, Hirrlinger J, P Kole MH, Shigemoto R (n.d.)
550 HCN channel-mediated neuromodulation can control action 1 potential velocity and fidelity
551 in central axons.

552 Cannon RC, Gewaltig M-O, Gleeson P, Bhalla US, Cornelis H, Hines ML, Howell FW, Muller E,
553 Stiles JR, Wils S, De Schutter E (2007) Interoperability of Neuroscience Modeling
554 Software: Current Status and Future Directions. Neuroinformatics 5:127–138.

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

556 Druckmann S, Banitt Y, Gidon A, Schürmann F, Markram H, Segev I (2007) A novel multiple
557 objective optimization framework for constraining conductance-based neuron models by
558 experimental data. Front Neurosci 1:7.

559 Einevoll GT, Destexhe A, Diesmann M, Grün S, Jirsa V, de Kamps M, Migliore M, Ness T V.,
560 Plesser HE, Schürmann F (2019) The Scientific Case for Brain Simulations. Neuron
561 102:735–744.

562 Eklund A, Dufort P, Forsberg D, LaConte SM (2013) Medical image processing on the GPU -
563 past, present and future. Med Image Anal 17:1073–1094.

564 Fidjeland AK, Shanahan MP (2010) Accelerated Simulation of Spiking Neural Networks Using
565 GPUs. Ijcnn'10:1–8.

566 Gagn C (2012) DEAP□: Evolutionary Algorithms Made Easy. J Mach Learn Res 13:2171–2175.

567 Go AW, Williamson MJ, Xu D, Poole D, Grand S Le, Walker RC, Götz AW, Williamson MJ, Xu
568 D, Poole D, Le Grand S, Walker RC (2012) Routine microsecond molecular dynamics
569 simulations with AMBER on GPUs. 1. generalized born. J Chem Theory Comput 8:1542–
570 1555.

571 Gouwens NW, Berg J, Feng D, Sorensen SA, Zeng H, Hawrylycz MJ, Koch C, Arkhipov A

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

572 (2018) Systematic generation of biophysically detailed models for diverse cortical neuron
573 types. Nat Commun 9.

574 Gurkiewicz M, Korngreen A (2007) A Numerical Approach to Ion Channel Modelling Using
575 Whole-Cell Voltage-Clamp Recordings and a Genetic Algorithm. PLoS Comput Biol
576 3:e169.

577 Hay E, Schurmann F, Markram H, Segev I, Schürmann F, Markram H, Segev I (2013)
578 Preserving axosomatic spiking features despite diverse dendritic morphology. J
579 Neurophysiol 109:2972–2981.

580 Hines M (1984) Efficient computation of branched nerve equations. Int J Biomed Comput
581 15:69–76.

582 Hines ML, Carnevale NT (2000) Expanding NEURON's Repertoire of Mechanisms with
583 NMODL. Neural Comput 12:995–1007.

584 Hines ML, Eichner H, Schürmann F (2008) Neuron splitting in compute-bound parallel network
585 simulations enables runtime scaling with twice as many processors. J Comput Neurosci
586 25:203–210.

587 Hines P, Fouriaux M, Jan NC (n.d.) An Optimized Compute Engine for the NEURON Simulator.

588 Keren N, Bar-Yehuda D, Korngreen A (2009) Experimentally guided modelling of dendritic
589 excitability in rat neocortical pyramidal neurones. J Physiol 587:1413–1437.

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

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

594 Mainen ZF, Sejnowski TJ (1996) Influence of dendritic structure on firing pattern in model
595 neocortical neurons. Nature 382:363–366.

596 Mäki-Marttunen T, Halnes G, Devor A, Metzner C, Dale AM, Andreassen OA, Einevoll GT
597 (2018) A stepwise neuron model fitting procedure designed for recordings with high spatial
598 resolution: Application to layer 5 pyramidal cells. J Neurosci Methods 293:264–283.

599 Markram H et al. (2015) Reconstruction and Simulation of Neocortical Microcircuitry. Cell
600 163:456–492.

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

604 Miceli F, Soldovieri MV, Ambrosino P, Barrese V, Migliore M, Cilio MR, Taglialatela M (2013)
605 Genotype–phenotype correlations in neonatal epilepsies caused by mutations in the
606 voltage sensor of K_v 7.2 potassium channel subunits. Proc Natl Acad Sci 110:4386–4391.

607 Migliore M, Migliore R (2012) Know Your Current Ih: Interaction with a Shunting Current
608 Explains the Puzzling Effects of Its Pharmacological or Pathological Modulations Attali B,
609 ed. PLoS One 7:e36867.

610 Nocedal J, Wright S (2006) Numerical optimization.

611 Nvidia C (2018) Cuda c programming guide, version 9.1. NVIDIA Corp.

bioRxiv preprint doi: https://doi.org/10.1101/727560; this version posted August 6, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

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.

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.

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.

Prinz AA, Bucher D, Marder E (2004) Similar network activity from disparate circuit parameters. Nat Neurosci 7:1345–1352.

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.

Ramaswamy S et al. (2015) The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. Front Neural Circuits 9:44.

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.

Schmidhuber J (2015) Deep Learning in neural networks: An overview. Neural Networks 61:85–117.

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. epub ahead of print

Van Geit W, De Schutter E, Achard P (2008) Automated neuron model optimization techniques: a review. Biol Cybern 99:241–251.

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, pp 1–11. IEEE.

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

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, pp 1–4. New York, New York, USA: ACM Press.