




SCALING AND BENCHMARKING THE GENETIC ALGORITHM IN CONSTRUCTING BIOPHYSICAL NEURONAL MODELS

A PREPRINT

Alexander Ladd ¹, Kristofer Bouchard ^{2,3}, Jan Balewski ⁴, and Roy Ben-Shalom ⁵

¹Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA, 94704, USA

²Biological Systems & Engineering Division & Scientific Data Division,
Lawrence Berkeley National Laboratory, Berkeley, CA 94704, USA

³Helen Wills Neuroscience Institute & Redwood Center for Theoretical Neuroscience,
University of California Berkeley, Berkeley, CA, 94704, USA

⁴Data Science Engagement, Lawrence Berkeley National Laboratory, Berkeley, CA, 94704, USA

⁵MIND Institute, University of California Davis, Sacramento, CA 95616, USA

November 22, 2021

ABSTRACT

A common algorithmic approach in generating biophysical neuron models are genetic algorithms. The procedure in which the genetic algorithm constructs realistic neuronal models is heavily reliant on large scale neuron simulation and electrophysiological feature extraction. The goal of our work was to minimize the amount of time it takes to initialize, simulate, and evaluate one generation. To accomplish this goal, we have developed a genetic algorithm that uses high performance computing (HPC) nodes to maximize the computational efficiency and minimize the run time of the algorithm. In this paper, we run several benchmarking experiments to determine efficient utilization of resources and scaling. We show that our GPU/CPU based design enables simulating multiple neuronal responses to multiple stimuli with more efficient resource utilization with nominal cost for run time. This analysis demonstrates an opportunity for researchers to build more accurate biophysical neuronal models, thus enabling more us to ask more detailed questions about neuron firing properties.

Keywords biophysical neuron model · compartmental neuron model · high performance computing · genetic algorithms · nonconvex optimization · parallelism · concurrency · strong scaling · weak scaling · electrophysiology

1 Introduction

Since the onset of electrophysiological science, great progress has been made in understanding the computational properties of single neuron units. State of the art research has progressed in enabling finer resolution recordings of neurons. In parallel, with Moore’s law, we have seen exponential growth in the computing capacity capable on a single chip. In tandem, these advancements have brought the field of computational neuroscience much closer to realistically modeling biological neurons on computers *in silico*. Still, research on *in silico* construction of detailed biophysical neuronal models stands to benefit from the aforementioned advances in high performance computing. Biophysical models such as the Hodgkin and Huxley model [Hodgkin and Huxley, 1952] and the multi-compartmental model

[Rall, 1959] aim to simulate electrical proprieties of neuron micro-circuitry. However, a major obstacle in models that faithfully simulate realistic firing patterns is constraining the density, and consequently the conductance, of the various membrane ion channels that play a major role in determining neuron firing patterns [Druckmann et al., 2007]. Examples of such channel parameters are C_m as membrane capacitance, \bar{g}_k as potassium conductance, V as membrane potential, E_k and E_{na} as potassium and sodium reverse potential respectively, and \bar{g}_{na} as maximal sodium conductance density. Simulation software aims use such conductances to solve the differential system of equations using the F-matrix, or tri-diagonal matrix in order to find the voltages of the dendritic tree. The computational complexity of these models scales quickly with the number of axonal, somatic, and dendritic channels, as well as the number of compartments. There exist trade-offs between the amount of detail, computation time, computational resources, and the questions that need to be answered by such models [Almog and Korngreen, 2016]. There is an opportunity for progress in defining the most efficient methods for building models, which will subsequently provide a natural means for systematically exploring exactly such trade-offs [Eliasmith and Trujillo, 2014]. By efficiently using computational resources and minimizing computation time, researchers can construct more complex models that can answer more specific questions about neuron functionality, thus providing a motivation for benchmarking the run time required to generate such models.

With increasing model complexity comes the need for more efficient optimization methods. The challenge with this optimization is that the search space of possible channel parameters is non-convex and entails a wide range of solutions that result in non-linearities. There are several types approaches including brute force search, Monte Carlo optimization algorithms such as genetic algorithms and simulated annealing, or mathematical heuristic based algorithms [Van Geit et al., 2007]. We chose to use a prevalent method for the non-convex optimization posed by this problem in the genetic algorithm [Vanier and Bower, 1999] [Keren et al., 2005] [Masoli et al., 2017]. The genetic algorithm employs an objective or cost function as a linear combination of different penalty functions between a simulated spike and a target spike. In this implementation, our objective function is constructed of score functions comparing electrophysiological firing properties of simulated and experimental target voltages. A small perturbation in the conductance of a single channel parameter can have a significant impact on the voltage trace produced by the simulated neuron, and consequently the objective function. Further complicating the problem, [Jelescu et al., 2016] describes multiple biological plausible local minima corresponding to dissimilar sets of parameters and states that nonlinear fit outputs suffer from heavy bias and poor precision. Increasing the number of stimuli in which the cost function is optimized over and constructing better objective functions can remedy some of the problems posed, but quickly incurs computational cost. For simplicity we consider constructing a "better" score function outside the scope of this work and focus on scaling the computational bandwidth of the genetic algorithm instead. In this work, we present a novel GPU/CPU-based simulation and evaluation loop that aims specifically to benefit to computational design of the genetic algorithm by leveraging speed increases using concurrency and parallelism. We show the motivation for accelerating this algorithm through scaling the parameter search algorithm on motivating example model. Furthermore, we aim to quantify the acceleration in run time from this programmatic design using well established scaling experimental design.

Currently, there exists a wide variety of quality software tools for computational neuroscience. With recent advances that introduce more accessible super-computing and exascale systems for super computing, there has also been progress in the adaptation of computational neuroscience tools for HPC platforms. These software tools aim for generalizability and scalability so they can be used for different purposes, on different platforms and at different scales. Moreover, most computational modeling pipelines require more than one of these software modules. In this paper, we focus on benchmarking two classes of software modules – neuron simulators and electrophysiological spike train feature extractors. While it is important to experiment with performance benchmarks that are specific to individual modules and provide fine-grained analysis of power usage, it is also important to develop benchmarks that asses the application of combinations of modules. This kind of performance analysis is crucial in building and using workflows and frameworks for distributed computing, and embedding them in a collaborative setting. It can be extremely productive to have computational/theoretical frameworks that permeate research directions and guide the scaling-up of data acquisition and analysis [Bouchard et al., 2016]. This paper aims to draw from previous work in benchmarking for computer science

[Hoeftler and Belli, 2015] [Wu et al., 2019] to experiment using established performance benchmarks to software for neuron simulation and biophysical modeling and employ statistically sound analysis and reporting techniques. [Coleman et al., 2019]. We focus on run time comparisons between different combinations of computational resources and software tools in order to demonstrate the state of the art, challenges, and opportunities, relating to efficiently utilizing such resources for complex biophysical neuronal modeling. Currently, researchers aiming to construct neuronal models that answer questions about brain circuitry stand to gain a great deal from measuring and accelerating computational tools and algorithms.

Adapting well known performance benchmarks to this algorithm helps understand how the algorithm can scale using different permutations of computational resources and software modules. While [Criado et al., 2020], [Kulkarni et al., 2021] and [Knight and Nowotny, 2018] [Van Albada et al., 2018] all provide distinct and relevant examples of benchmarking simulation modules and computational platforms for simulation, such as neuromorphic hardware. However, there is a gap in research benchmarking the performance of such simulators applied to an optimization problem. [Benjamin et al., 2018] focuses on benchmarking the predictive power of machine learning methods for fitting neural responses, but does not focus on run time or scalability of methods with that objective. This work aims to address this gap by providing several examples of evaluating the run time performance of the genetic algorithm as a method to construct biophysical neuron models. Thus, the principal contributions of this paper are as follows:

1. We present an optimized implementation of the genetic algorithm that aims to accelerate the time it takes to fit a biophysical neuronal model by leveraging parallelism on high performance GPU nodes.
2. We benchmark the run time of this algorithm using well established performance benchmarks *weak scaling* and *strong scaling*.
3. We show the applicability of such a practice across different software modules for evaluation-simulation loops by:
 - 3a. Running experiments on CPU only nodes.
 - 3b. Considering implementations with different electrophysiological feature extraction libraries
 - 3c. Considering implementations with different GPU-based neuron simulation modules.

In the following sections of this paper we will first give a brief overview of the methodology and implementation of the genetic algorithm and how simulation and feature extraction drive the algorithm toward increasingly realistic neuronal modeling. Next, we will specify the hardware and software of the machines we used. Then we will give a description of the experimental design run on CORI to test the scaling of each variation of this algorithm. The experimental design allows for the comparison of different algorithms for using GPU and CPU as well as different software modules in the simulate-evaluate loop. Subsequently, we will demonstrate the results of such experiments and discuss the implications, limitations and future steps for our analyses. We will show a motivating example cell and how scaling the genetic algorithm for this cell result in a more realistic model.

2 Methods

First we will describe the use case of the genetic algorithm to fit neuronal models. Next, we will describe approaches to implementing parallelism in the genetic algorithm and show that each approach embarrassingly parallel [Herlihy and Shavit, 2012]. Finally, we will conclude by specifying the how the variations of the genetic algorithm, that are ran in our experiments, are implemented.

2.1 Genetic Algorithm

Evolutionary algorithms are a class of optimization methods that rely on instantiating a population and subsequently implementing natural selection through biologically inspired operators such as mutation, crossover, and fitness based selection [Mitchell, 1998]. Genetic algorithms, are a subset of evolutionary algorithms that encode solutions to an optimization problem into vector representations in a real valued search space and represent the quality of these representations by evaluating a fitness function that takes these vectors as an input. Below, is the pseudocode for the genetic algorithm we implement, known as the (μ, λ) genetic algorithm [Beyer, 2007] [Beyer and Schwefel, 2002] and referred to in the text as GA. In this implementation, the following parameters are provided, namely μ and λ which are the number of individuals to select for the next generation and the number of children to produce for the next generation respectively. The parameter cX rate determines the probability that an offspring is produced by crossover and the parameter $mutRate$ is the probability that and offspring is produced via mutation. Mutation is a perturbation of a single or more parameters and crossover is a combination between a pair of parameter sets.

The function *variation* in the pseudocode applies mutation, reproduction, or crossover exclusively to each individual, or pair in the case of crossover, to produce λ new offspring. The top μ individuals among the current population and offspring are kept in the population and thus produce the next set of offspring. The optimization problem posed in this paper defines an individual i in generation j as $\mathbf{x}_i^{(j)} \in \mathbb{R}^{15}$ and the entire population at generation j is defined as $\mathbf{X}^{(j)} \in \mathbb{R}^{15 \times P}$, where 15 is the number of channel parameters

for each simulation and P is the size of the population. Finally, in the *evaluate* function, the fitness for each individual $i \in N$, where N , the total number of individuals in a population, also referred to as "offspring", is determined by an objective function where \mathbf{x}_i is the input for simulation at S stimuli, shown in Figure 2A and evaluated against experimental data, shown in Figure 2B and evaluated using F score functions shown in Figure 2C. This procedure

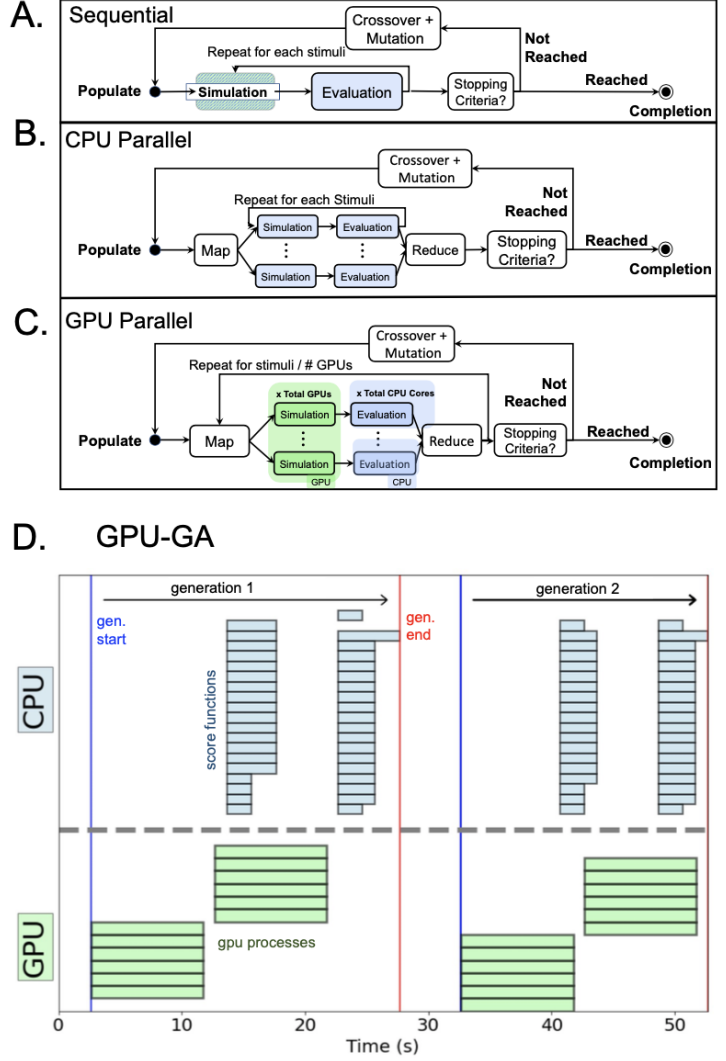


Figure 1: **Algorithmic design:** **A:** Sequential execution of tasks for genetic algorithm. The simulation can take place on either the CPU or GPU. **B:** The CPU algorithm(CPU-GA) simply maps the simulation/evaluation of a model to a single core. **C:** The GPU adapted algorithm (GPU-GA). Each stimuli is mapped to run in parallel across each GPU, then scored in parallel, mapping one score function to one core, across all CPU cores available on the GPU node. **D** Shows two generations that illustrates in GPU-GA with NeuroGPU, stimuli can be launched on GPU while the CPU evaluation step is still completing.

results in a scalar score for each individual. Considering \mathbf{y}_{s*} as the target voltage response to stimuli (5B) s the fitness score for an individual \mathbf{x}_i can be defined as $\sum_{s \in S} \sum_{f \in F} f(\text{simulate}(\mathbf{x}_i, s), \mathbf{y}_{s*})$ where the stimuli and the individual parameter vector $\mathbf{x}_i^{(j)}$ are inputs into the function *simulate*. Using figure 2A for reference, an example objective could be $F(t, v) = ISI(t, v) + AHP(t, v) + \dots + AP_Amplitude(t, v)$ where t is the target voltage trace, v is the simulated voltage trace, and *ISI* is inter-spike interval, *AHP* is after hyper-polarization peak, and *AP_Amplitude* is the action potential amplitude. See Figure 2C for an illustration of how scoring functions are computed on a single trace. In conclusion, the problem space for this step can be defined as having cardinality $N \times S \times F$, that is, the population size \times the number of stimuli presented \times the number of score functions used. Next, we will describe more specifically what a population is and how simulations are run on this population.

The genetic algorithm population consists of individual multi-compartment neuron models. In the context of this work, we use a pyramidal neuron from mouse V5 as a complex model, with 14 different ionic channel parameters in axon, soma, and dendrite [Lein et al., 2007]. This cell morphology and parameterization is can be found ModelDB as, L5_TTPC1_cADpyr232_1 or model 267067 [Hines et al., 2004] Spratt et al. [2021]. This cell is based on the Layer 5 thick-tufted pyramidal cell from the Blue Brain Project [Markram et al., 2015]. The following model parameters were used: low voltage calcium (*LVACA*) Avery and Johnston [1996], high voltage calcium (*HVACA*) Reuveni et al. [1993], hyper-polarization activated current (I_h) [Kole et al., 2006], persistent potassium current (*KPst*), transient potassium current (*KTst*) [Korngreen and Sakmann, 2000], persistent sodium channel (*Nap*) [Magistretti and Alonso, 1999], Fast inactivating sodium channel (*NaT*) [Colbert and Pan, 2002], calcium activate potassium *SK* [Köhler et al., 1996], fast non-inactivating potassium channel, (*Kv3.1*) [Rudy and McBain, 2001]; M-type potassium current (I_m) [Adams et al., 1982]. Figure S1 shows how these parameters are distributed across axonal, somatic sections, as well as the upper and lower optimization bounds for each conductance.

Algorithm 1 Genetic Algorithm

```

1: procedure OPTIMIZE( $\mu, \lambda, \text{cxRate}, \text{mutRate}, n\text{Generations}$ )
2:   population  $\leftarrow$  INITIALIZE()
3:   for generation  $\leftarrow 1, n\text{Generations}$  do
4:     offspring  $\leftarrow$  VARIATION(population,  $\lambda, \text{cxRate}, \text{mutRate}$ )
5:     scores = EVALUATE(offspring, population)
6:     population  $\leftarrow$  SELECT(population, offspring, scores,  $\mu$ )            $\triangleright$  keep  $\mu$  individuals with highest scores
   end
7:   return  $\text{argmax}(\text{EVALUATE}(\text{population}))$             $\triangleright$  this is the individual with the highest known fitness

```

Algorithm 2 Evaluation

```

1: procedure EVALUATE(offspring)
2:   for all stim  $\in Stims$  do
3:     responses  $\leftarrow$  SIMULATE(offspring, stim)
4:     for all scoreFunction  $\in scoreFunctions$  do
5:       scores  $+=$  scoreFunction(responses, target)
   end
6:   return scores

```

2.2 Implementations

In **Algorithm 2** there are three opportunities to implement parallelism. First, one could run all of the score functions at once for a given stimulus. Second, one could run of of the simulations for each stimuli at one. Third, one could run the simulate-evaluate loop in parallel across the entire population. IN this example, *scores* and *responses* are lists containing the firing traces and scores for each individual respectively. The problem could be written as a triple for loop, but is written as a double loop in order to simplify the fact that *scores* is a list with a cumulative score (over all stimuli and score functions) corresponding to each individual. Because each *response* and *score* do not require any

information about other simulations, other score functions, or even other individuals in the population, the problem is *embarrassingly parallel*. Our CPU and GPU algorithms take advantage of this design in different ways.

(1) For comparison, the sequential algorithm is demonstrated in the top panel of Figure 1. (2) The CPU-GA implementation of this algorithm is parallelized over the entire offspring and allocates one CPU core per offspring. This is represented in the bottom panel of Figure 1. In this implementation, Algorithm 2 demonstrates the exact order of execution performed by the algorithm. The only difference between CPU-GA and the pseudo-code is that the procedure is run in parallel across the entire offspring and aggregated into a list once all scores have been calculated in a process known as map-reduce [Chu et al., 2007]. (3) The GPU-GA version of this algorithm uses multiple layers of parallelism, as demonstrated in Figure 1. First, like CPU-GA, the entire population is divided using map-reduce schema, however, in this version the entire population is split across nodes. If the population size is 1000 and there are 2 nodes, each node will be responsible for running Algorithm 2 for 500 individual cells. In the next layer of parallelism, the simulations for each stimuli are computed in parallel across each available GPU. Following the example, if there are 8 GPUs per node, then 8 stimuli will be ran for each of the 500 cells using eight GPUs. If there are more stimuli than GPUs, then a loop will be required. Finally, once the responses have been obtained, each score function is computed in parallel across each CPU core. CORI GPU nodes have 80 CPU cores, see more in experimental design. Following the example, if there are 80 cores and 10 score functions for 8 stimuli then there are 80 total score functions and each core will compute the one score function on one stimuli for all 500 individuals. The order of execution in this example is represented in Figure 1C. A perk of this execution flow that will be covered more in depth in the results section is that if there are more stimuli than GPUs, we can launch the next set of simulations while the CPU cores handle score function evaluation. This instance is demonstrated in Figure 1D.

Hardware: Each **Cori GPU** node has two sockets of 20-core Intel Xeon Gold 6148 ('Skylake') CPUs with 384 GB DDR4 RAM memory and a clock rate of 2.40 GHz. Each of these nodes also has 8 NVIDIA Tesla V100 ('Volta') GPUs, connected with NVLink interconnect, each with with 16 GB HBM2 memory. We used Cray's Programming Environment version 2.6.2. Allocated nodes were chosen by the batch system (slurm 20.11.8) and were allocated exclusively to eliminate on-node interference. The system uses InfiniBand host network adapters (HCA) and network interface cards (NICs).

Each **Cori CPU** node has two sockets, each socket is populated with a 2.3 GHz 16-core Haswell Intel Xeon Processor E5-2698 v3. Each core supports 2 hyper-threads, and has two 256-bit-wide vector units 36.8 Gflops/core (theoretical peak), 1.2 TFlops/node (theoretical peak) and 2.81 PFlops total (theoretical peak) Each node has 128 GB DDR4 2133 MHz memory (four 16 GB DIMMs per socket) and 298.5 TB total aggregate memory. The interconnect is Cray Aries with Dragonfly topology with >45 TB/s global peak bisection bandwidth. We used Cray's Programming Environment version 2.6.2. Allocated nodes were chosen by the batch system (slurm 20.11.8) and were allocated exclusively to

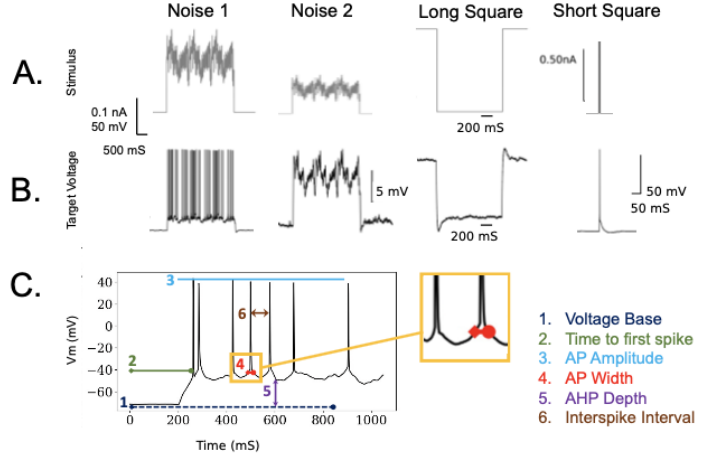


Figure 2: **Stimuli and score functions used in algorithm:** **2A** represents various stimuli used in the fitting procedure of the genetic algorithm. **2B** represents some of the corresponding target voltages that are recorded from patch clamp experiments as a result of the stimuli in 2A. **2C** Demonstrates how electrophysiological score functions are computed on a single trace. These score functions are used to compare target and simulated firing traces.

eliminate on-node interference. For all experiments, we used Cori *SCRATCH* which is a Lustre file system designed for high performance temporary storage of large files. All experiments were ran on x86_64 computing architecture, SUSE Linux Enterprise 15 and kernel 4.12.14-150.75-default.

Software: We used GCC compiler version 8.3.0, CUDA version 11.1.1, OpenMPI version 4.0.3, Python 3.8.6. As in previous work [Ben-Shalom et al., 2020], the genetic algorithm was implemented using the DEAP 1.3 [Fortin et al., 2012] and Bluepyopt 1.9.126 [Van Geit et al., 2016] python libraries. Score functions are implemented using Blue Brain Project’s Electrophys Feature Extract Library 3.2.4 (eFEL) [Van Geit et al., 2016]. The CPU based neuron simulations are run using NEURON 7.6.7 *.mod* files and the NEURON python interface. ther software versions and requirements are added as a supplementary information. For the installation of CoreNeuron we used the PGI compiler, version 20.11-0 and we used Cori’s cray-python version 3.7.3.2 to avoid compilation issues with anaconda python used in other experiments.

3 Experiments

3.0.1 Experimental Design

Experiments were designed to record the run time of the different algorithmic designs and measure the scaling efficiency of the algorithm. In these experiments, we refer to run time, t , as wall time or elapsed real time. t is the time in seconds it takes for a program to execute from start to finish. The target measurement is the run time to complete one simulation-evaluation step and all experiments are ran for at least 50 generations of the genetic algorithm in order gain enough samples. Following [Hoeffler and Belli, 2015] suggestion that speed-up should be replaced by run time lower bounds, we provide run time lower bounds as ideal scaling measures. Furthermore, in many cases, it was not realistic to benchmark the sequential case described in the methods section as it would be too slow to show any meaningful results. Below we describe each experiment and provide a table of the specifications for each run.

(1) Compute Fixed Problem Scales

In this experimental design the population size N , starts at 500 and increases by 500 in each trial until the population size reaches 5000. The computational power remains fixed at one node. Each trial uses 20 score functions and 8 stimuli. Because the only experimental value that changes is population size, a table is not shown for this experiment. Here, we expect to see run time increase proportionally to the increase in population size. These experiments still advantage single-node parallelism, but they demonstrate how the computation time increases as the population size increases.

(2) Compute Scales Problem Fixed

In this experimental design, specified by Table 1, the population size N is fixed at 3000, while the computational power scales exponentially. These experiments are defined as *strong scaling*. Here, the expectation is that run time decreases exponentially by a factor of 2, which corresponds to the compute scaling rate. These experiments demonstrate how effective multi-node parallelism is at accelerating the time it takes to simulate and evaluate a population.

Table 1: Compute Scales and Problem Fixed

Nodes	Total Cpus	Total Gpus	Offspring	Stimuli	Score Functions
1	42	8	3000	8	20
4	168	32	3000	8	20
8	336	64	3000	8	20
16	672	128	3000	8	20

(3) Problem Scales Compute Scales In this experimental design, specified by Table 2, the initial trial sets $N = 250$ for one node. The subsequent trials increase computing power and population size proportionally (at 250 individuals per node). These experiments are defined as *weak scaling*. Here, the expectation is that run time remains constant. These experiments demonstrate how multi-node parallelism can accommodate the scaling of population size in the genetic algorithm.

The set of experiments above only changes the problem size using population size, N . In order to make straightforward comparisons, we run further experiments interchanging parts of the algorithm or scaling the problem size using different scaling factors. Figure 3A demonstrates an experiment following the specified design where CoreNeuron has been substituted for NeuroGPU in the GPU

Table 2: Compute Scales and Problem Scales

Nodes	Total Cpus	Total Gpus	Offspring	Stimuli	Score Functions
1	42	8	250	8	20
2	84	16	500	8	20
4	168	32	1000	8	20
8	336	64	2000	8	20
16	672	128	4000	8	20

genetic algorithm [Kumbhar et al., 2019]. Further experiments were ran to compare the performance of GA with various electrophysiological feature extraction libraries. In these experiments we use the previously specified experimental design but change the number of score functions from 20 to 8. These experiments, shown in row 2 of Figure S7 are examples of an experiment that interchanges the score function module eFEL for the score function module from Allen IPFX ¹ in the GPU-based genetic algorithm. In these experiments, we modify the experimental design by using 8 score functions from the IPFX module and comparing against 8 eFEL score functions.

To further explore the axes of scaling GPU-GA problem space, we ran scaling experiments where score functions are set to 20, population size is set to 500 but the number of stimuli used in GA increases from 1 to 18 stimuli. This experiment is shown in Figure 2A. Furthermore, we ran an experiment on GPU-GA where stimuli are set to 8, population size is set to 500 but the number of score functions used in GA increases from 1 to 180. This experiment is shown in Figure 2B.

4 Results

The benchmarks described in the methods section serve as general measures to determine scaling performance in the defined problem space. Each benchmark serves a specific purpose in determining algorithmic performance. The first benchmark, described as *Compute Fixed Problem Scales* measures the difficulty of the scaling application. Keeping computational resources constant demonstrates how the run time scales as the problem size increases if only one node was used. The results from this benchmark are shown in 3A and S7A where we chose to increase the population size by 500 in each trial. The run time of the algorithm is expected to increase directly proportionally to the increase of the population size. If 500 offspring take 14 seconds to simulate, then 1000 should take 28 seconds. These experiments demonstrate that the GPU-GA experiments ran much faster than the CPU-GA experiments. The CPU-GA is a more accessible solution to users working with a single desktop computer or CPU only nodes. At a small population size (< 1000) CPU-GA took 10x the amount of time it took GPU-GA to complete a simulation-evaluation step, but at larger scales (> 2000) CPU-GA took 100x longer than GPU-GA. These results suggest that CPU-GA may be a reasonable choice for fitting simple electrophysiological neuron models, but that researchers should use caution in more computationally complex optimization problems that require scaling. Next, it is clear in both Figure 3A and Figure S7A that the speed increase of the genetic algorithm can be closely predicted and modeled as a proportional to the increase in population size. Understanding this property is important because it shows that speed increases on single nodes or single desktop computes will translate favorably towards an algorithm being run at a large scale. Overall, these experiments confirm that regardless of hardware or software configuration, the optimization problem grows directly proportional to population size.

The second benchmark, described as *Compute Scales Problem Fixed* determines the strong scaling of the application. By keeping the problem size constant and increasing computational resources, we can see how much performance can be gained through exploiting parallelism on a single problem. The expectation in this experiment is that as the compute increases by a factor N , the problem should decrease by a factor of N . We chose to scale N exponentially, which

¹© 2020 Allen Institute for Brain Science. Intrinsic Physiology Feature Extractor (IPFX). Available from: github.com/AllenInstitute/ipfx

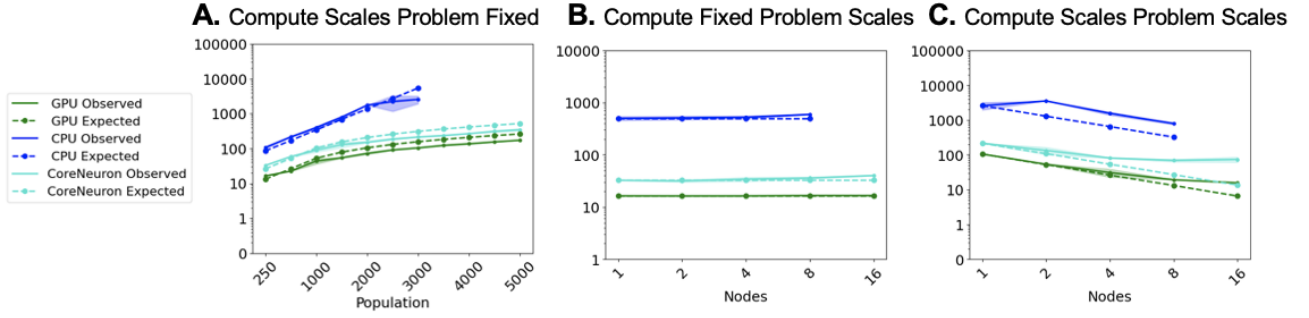


Figure 3: **Simulation-evaluation scaling CPU v. GPU: 3A, 3B, 3C:** Represent scaling experiment results of the time it takes to run one simulation-evaluation step. Figures correspond to experimental design sections. Fig. 3B and 3C correspond to Table 1 and Table 2 respectively. Experiments compare the GPU-based algorithm using CoreNeuron as a simulator and NeuroGPU as a simulator, as well as the CPU-based genetic algorithm using NEURON as a simulator. **Note:** In some figures the y axis is logarithmically scaled to show exponential scaling and in some figures it is not.

is represented in Figure 3B and Figure S7B. First, for all GPU-based algorithms it is clear that at after 4 nodes, or 2 nodes in the case of CoreNeuron GPU-GA, that run time acceleration per node starts to decrease and no longer match ideal scaling. Notably this demonstrates a limit to which parallelism in GPU-GA can efficiently leverage available resources. In 3B, CoreNeuron GPU-GA incurs overhead when too many nodes are used. The reason for this being that because all simulations are run at once and then all simulations are evaluated at once, there is overhead cost associated with a CPU bottleneck in the evaluation step. Figure 4 shows GPU-GA CPU run time decreases correspondingly with the inclusion of more resources. Further investigation of this experiment in Figure 4 demonstrates a trade-off between theoretical efficiency and empirical efficiency. At around 250 individuals per node, an equal amount of time is spent on GPU computation as CPU computation. Figure 4 B, there is a trade-off between time spent on the GPU and CPU time and population size. Choosing 250 individuals per node results in a balance between CPU time and GPU time and thus most efficient utilization of resources in achieving concurrent CPU/GPU calculations. Based on this trade off, we chose to use 250 individuals per node for the final experiment, *Compute Scales Problem Scales*.

The third benchmark, described as *Compute Scales Problem Scales* determines the weak scaling of the application. Maintaining a constant run time on this benchmark exemplifies how the most efficiently scale the implemented algorithm for then application of developing single neuron models. As demonstrated in the 3C and Figure S7C, by scaling at 250 individuals per node on the GPU and CPU, the run time of algorithm remains constant while running up to 10 nodes on GPU. Subsequently, the run time starts to marginally increase with each trial. On the CPU, the increase in run time is marginal as well. Figure S7C demonstrates that eFEL score functions and Allen IPFX provide the same constant scaling we expect to see with the performance being nearly identical, with the IPFX library a few seconds faster than eFEL. In 3C, the CPU-GA and GPU-GA both scale at a constant rate without incurring much overhead. CPU-GA incurs 10 seconds overhead and NeuroGPU-GA and CoreNeuron-GA incurs nominal overhead for up to 16 nodes. Further experiments, shown in Figure S3, demonstrate that overhead is incurred when NeuroGPU-GA is run on larger allocations of GPU nodes (64-128 Nodes) using the Summit computing cluster. Ultimately, this set of experiments demonstrates that regardless of hardware and software configuration, that optimization using the genetic algorithm for fitting electrophysiological neuronal models is a prime target for scaling on HPC clusters and particularly GPU enabled clusters. In the discussion section, further consideration is taken towards the explaining implications of successful large scale optimization runs and the software/hardware that powers such runs.

Initially, we present two efficient CPU and GPU based algorithms for the simulation and evaluation of single neuronal models, CPU-GA and GPU-GA. Unlike CPU-GA, GPU-GA is able to leverage parallelized kernel computation for fast simulation. GPU-GA also adds concurrency to the algorithm described in methods. Concurrency, defined as the

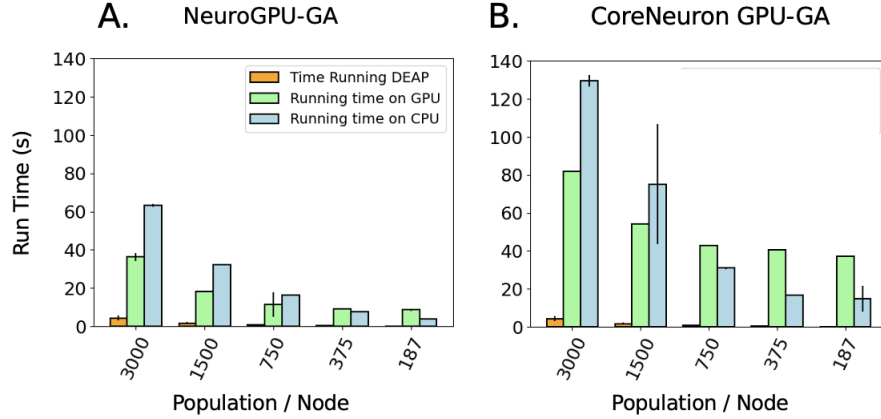


Figure 4: **CPU bottleneck shifts to GPU as population per node decreases:** The time the algorithm spends on simulation/evaluation/evolution as a function of population size. At large population sizes the CPU operation for score functions is the bottleneck. At smaller population sizes, the DEAP evolution is the bottleneck for NeuroGPU in **4A** and the GPU simulation is the bottleneck for CoreNeuron in **4B**.

capacity to run separate tasks at the same time, is different than the achieved levels of multi-node and single node parallelism. This algorithm also achieves concurrency because although it can simulate one stimuli per GPU in parallel (one on each GPU), simulating the remainder of stimuli can start as soon as the first set finishes, independently of the evaluation step. Thus, while the CPU is evaluating the quality of the simulations, the GPU can run more simulations. This is shown in Figure 1B as the CPU and GPU are running at the same time. The result of concurrency in GPU-GA is shown in 5A where the algorithm scales logarithmically with the number of stimuli used in the algorithm. This makes it favorable to include multiple types of diverse stimuli. Overall, the GPU-GA algorithm scales favorably and demonstrates efficacy in accelerating / scaling the genetic algorithm.

5 Discussion

The most basic comparison drawn in this paper is between CPU and GPU based simulation-evaluation loops. GPU based simulation is markedly faster than CPU based simulation, which is to be expected as (lit review here). (explicate quote). However, the evaluation step for both loops was implemented on the CPU so no direct comparisons can be drawn. It is notable that in the GPU based simulation-evaluation loop, the bottleneck preventing the efficient utilization of resources for larger population sizes is the CPU processing capacity for scoring electrophysiological features. To the best of the researchers knowledge, there are no available GPU based software toolkit for scoring features of simulated spike trains based on a target train. There are several GPU-based

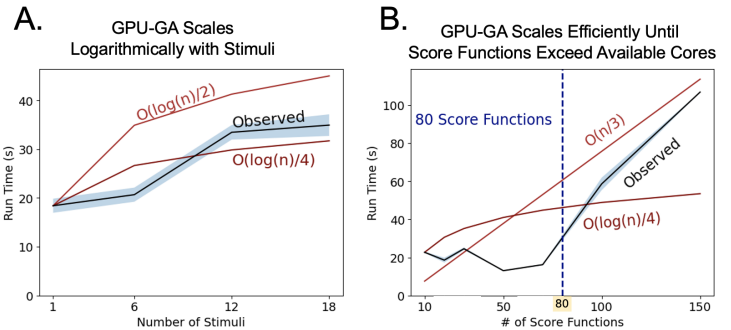


Figure 5: **Scaling stimuli and scoring functions:** **5A** Represents the observed run time as the number of stimuli used in the algorithm increases. We provide two lines for scaling reference $O(\log(n)/2)$ and $O(\log(n)/4)$. **5B** Represents the observed run time as the number of score functions used in the algorithm increases. We provide two lines for reference, $O(\log(n)/3)$ and $O(\log(n)/4)$.

applications that are used for real time analysis of Electroencephalography (EEG) waveform data, Magnetoencephalography (MEG) data, and Multi-electrode Arrays (MEA) signals [Sahoo et al., 2016] [Tadel et al., 2011] [Guzman et al., 2014], but none that exist for evaluating simulated neuron firing traces. As methods for recording and simulating spiking data from populations of neurons scale in capacity, it is also important to scale libraries for scoring electrophysiological traces. In the results section, we state that we chose a population size of 250 for scaling our algorithm. After profiling the GPU code at different population sizes, it was clear that at smaller population sizes, the GPU simulation is the bottleneck and at larger population sizes (> 250) the CPU is the bottleneck. The motivation for balancing the run time on the CPU and GPU is that if the problem size gets bigger, the GPU simulations can be ran concurrently with the CPU evaluation. The GPU simulation for the next set of stimuli to be simulated can run before the previous evaluation is completed since the two problems are independent. Thus, obtaining this balance results in the most time-efficient scaling constant.

This comparison between simulation and evaluation steps is further evidence to confirm that heavily vectorized computations scale more efficiently on the GPU. The prospective advantage of GPU accelerated electrophysiological feature extraction further implicates the need for scalable libraries. Because the practice of quantifying efficiency is relative, we aim to demonstrate the use of benchmarks that equitably generalize across applications. The benchmarks used in this analysis can serve as benchmarks for a range of similar applications that use software suites with similar outputs but different implementations. For example, in Figure 3, we demonstrate the comparative scaling of a GPU-based algorithm that implements the Allen institute IPFX feature extraction library and a GPU-based algorithm that implements the eFEL feature extraction library. The practice of using the Compute Fixed and Population Scales benchmark demonstrates that both libraries perform similarly in strong and weak scaling experiments and provides an example of drawing a principled comparison between the efficiency of two evaluation libraries. In this example, both libraries demonstrate a similar level of efficiency. These examples build toward a quantitative methodology for the comparison of other simulation and evaluation based tools, including ones that are not based on single cell patch-clamp recordings but from other recording/imaging modalities such as fMRI, MEA, and EEG. Some example experiments could be designed to compare: (1) research example of the comparisons we make and write it up tactfully. While the comparisons being made are specific, the broad motivation for conducting experiments to determine how to design scalable algorithms for electrophysiological model fitting.

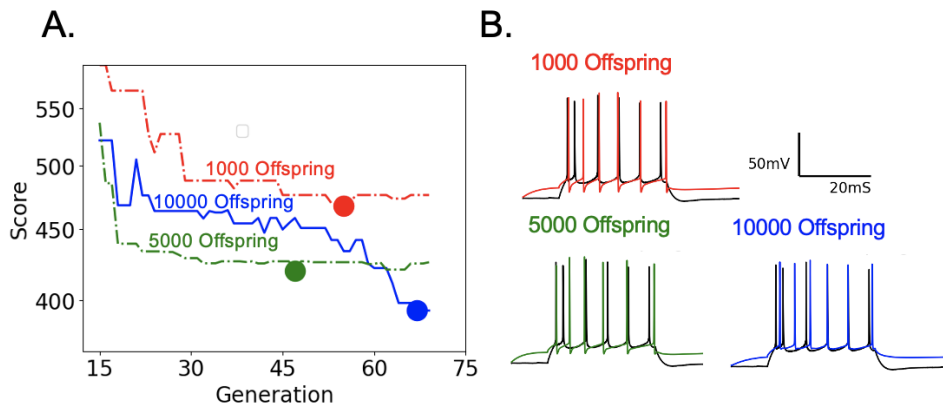


Figure 6: **GA score and model fit both improve with larger amount of offspring:** **6A** demonstrates the progression of the objective function being optimized by the genetic algorithm for varying population sizes. It shows that GAs with larger population sizes are more capable of reaching lower values of the objective function. The minima of the objective function are denoted by large circles and the lower the minima the more the best simulated response resembles the target response. **6B** illustrates the neuron model responses corresponding to varying population sizes. Vm denotes neuronal membrane voltage

To understand the impact of accelerating the genetic algorithm, we set up an experiment on a 14 parameter L5 pyramidal cell model from the Blue Brain Project. In this experiment, we ran 75 iterations of the genetic algorithm for three different trials with population sizes 1000, 5000, 10000, respectively. We fixed number of simuli to 20 and number of score functions to 8. The purpose of this experiment is to demonstrate the benefit of increasing population size on the resulting optimized model. Figure 6A demonstrates the loss of the objective function for the genetic algorithm. The 10000 individual curve achieves a more realistic model, resulting in the lowest achieved value for the objective function. Furthermore, the smaller populations converge upon local minima while the 10000 individual optimization records it's minimized loss value at generation 75 indicating the potential for further improvement. Figure 6A and B support the previous claims that the larger population size results in a more representative simulated neuronal model. In figure 6A, the GA with larger population size achieves the lowest objective function score. This score represents a penalty against simulated neurons where the electrophysiological features of voltage traces differ from those of the target trace. Furthermore, as the population sizes increase, models show improvement in the depolarization after the action potential is triggered, also known as *AHP Depth*. After the initial double spike the 10000 individual model demonstrates a closer correspondence to the inter-spike interval, or *ISI*. This effect is shown in figure 6B.

5.1 Conclusion

This work demonstrates the potential of efficiently parallelized simulation and evaluation software for electrophysiological modeling. Specifically, applications that leverage GPU utilization demonstrate the capacity to run larger fitting optimizations. In turn, these optimizations can result in a larger search of the parameter space, and consequently, a more accurate model. As the processor count continues to increase on hyperthreaded and multicore chips, computational methods that leverage parallelism can continue to leverage new innovations in high performance computing in order to generate more accurate *in silico* models. While this progression is beneficial, it is ever relevant to apply established benchmarks such as weak scaling and strong scaling in order for neuroscientists to get the most value out of new computing resources. In this paper, we first defined the problem space associated with a neuron parameter optimization problem. Then we demonstrate three established benchmark tests, scaling the problem size, the compute resources and both in proportion on CPU and GPU based applications. Next, we shared experiments to show the modularity of these methodologies for benchmarking the performance for different software tools and research questions. Finally, we demonstrated the importance of acceleration for this optimization problem using an example cell model.

Simulation-Evaluation Scaling IPFX v. eFEL

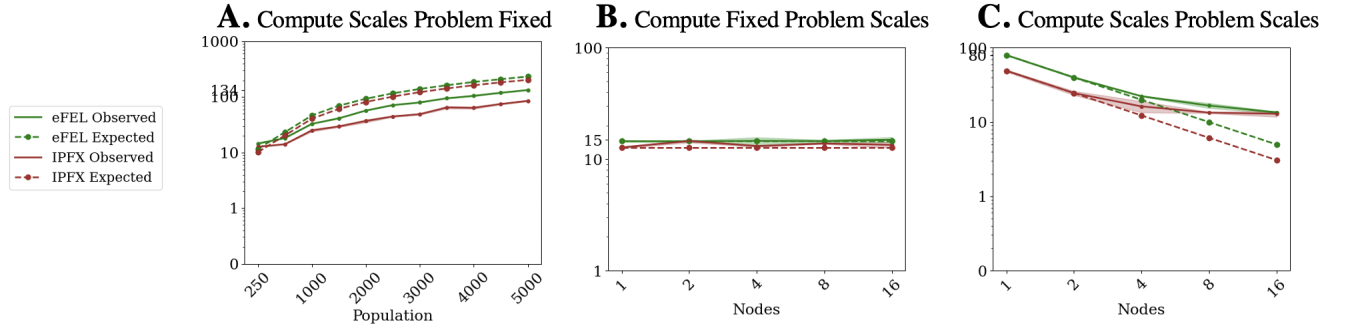


Figure 7: **7A, 7B, 7C** : Compares genetic algorithm simulate-evaluate run time between Blue Brain Project’s eFEL score function library and the Allen Institute’s IPFX score function library. All experiments in this figure use the NeuroGPU simulator. Unlike Figure 5, 8 score functions were used instead of 20. **Note:** In some figures the y axis is logarithmically scaled to show exponential scaling and in some figures it is not.

References

- Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.
- Wilfrid Rall. Branching dendritic trees and motoneuron membrane resistivity. *Experimental neurology*, 1(5):491–527, 1959.
- Shaul Druckmann, Yoav Banitt, Albert Gidon, Felix Schürmann, Henry Markram, and Idan Segev. A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. *Frontiers in Neuroscience*, 1:1, 2007. ISSN 1662-453X. doi:10.3389/neuro.01.1.1.001.2007. URL <https://www.frontiersin.org/article/10.3389/neuro.01.1.1.001.2007>.
- Mara Almog and Alon Korngreen. Is realistic neuronal modeling realistic? *Journal of neurophysiology*, 116(5): 2180–2209, 2016.
- Chris Eliasmith and Oliver Trujillo. The use and abuse of large-scale brain models. *Current Opinion in Neurobiology*, 25: 1–6, 2014. ISSN 0959-4388. doi:<https://doi.org/10.1016/j.conb.2013.09.009>. URL <https://www.sciencedirect.com/science/article/pii/S095943881300189X>. Theoretical and computational neuroscience.
- Werner Van Geit, Pablo Achard, and Erik De Schutter. Neurofitter: a parameter tuning package for a wide range of electrophysiological neuron models. *Frontiers in neuroinformatics*, 1:1, 2007.
- M. C. Vanier and J. M. Bower. A comparative survey of automated parameter-search methods for compartmental neural models. *J Comput Neurosci*, 7(2):149–171, 1999.
- Naomi Keren, Noam Peled, and Alon Korngreen. Constraining compartmental models using multiple voltage recordings and genetic algorithms. *Journal of neurophysiology*, 2005.
- Stefano Masoli, Martina F Rizza, Martina Sgritta, Werner Van Geit, Felix Schürmann, and Egidio D’Angelo. Single neuron optimization as a basis for accurate biophysical modeling: the case of cerebellar granule cells. *Frontiers in cellular neuroscience*, 11:71, 2017.
- Ileana O Jelescu, Jelle Veraart, Els Fieremans, and Dmitry S Novikov. Degeneracy in model parameter estimation for multi-compartmental diffusion in neuronal tissue. *NMR in Biomedicine*, 29(1):33–47, 2016.

- Kristofer E Bouchard, James B Aimone, Miyoung Chun, Thomas Dean, Michael Denker, Markus Diesmann, David D Donofrio, Loren M Frank, Narayanan Kasthuri, Chirstof Koch, et al. High-performance computing in neuroscience for data-driven discovery, integration, and dissemination. *Neuron*, 92(3):628–631, 2016.
- Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–12, 2015.
- Xingfu Wu, Valerie Taylor, Justin M Wozniak, Rick Stevens, Thomas Brettin, and Fangfang Xia. Performance, energy, and scalability analysis and improvement of parallel cancer deep learning candle benchmarks. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–11, 2019.
- Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *SIGOPS Oper. Syst. Rev.*, 53(1):14–25, July 2019. ISSN 0163-5980. doi:10.1145/3352020.3352024. URL <https://doi.org/10.1145/3352020.3352024>.
- Joel Criado, Marta Garcia-Gasulla, Pramod Kumbhar, Omar Awile, Ioannis Magkanaris, and Filippo Mantovani. Coreneuron: Performance and energy efficiency evaluation on intel and arm cpus. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 540–548, 2020. doi:10.1109/CLUSTER49012.2020.00077.
- Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell, and Catherine D. Schuman. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing*, 447:145–160, 2021. ISSN 0925-2312. doi:<https://doi.org/10.1016/j.neucom.2021.03.028>. URL <https://www.sciencedirect.com/science/article/pii/S0925231221003969>.
- James C. Knight and Thomas Nowotny. Gpus outperform current hpc and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Frontiers in Neuroscience*, 12:941, 2018. ISSN 1662-453X. doi:10.3389/fnins.2018.00941. URL <https://www.frontiersin.org/article/10.3389/fnins.2018.00941>.
- Sacha J Van Albada, Andrew G Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B Stokes, David R Lester, Markus Diesmann, and Steve B Furber. Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Frontiers in neuroscience*, 12:291, 2018.
- Ari S Benjamin, Hugo L Fernandes, Tucker Tomlinson, Pavan Ramkumar, Chris VerSteeg, Raed H Chowdhury, Lee E Miller, and Konrad P Kording. Modern machine learning as a benchmark for fitting neural responses. *Frontiers in computational neuroscience*, 12:56, 2018.
- Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012. ISBN 9780123973375.
- Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
- H. Beyer. Evolution strategies. *Scholarpedia*, 2(8):1965, 2007. doi:10.4249/scholarpedia.1965. revision #193589.
- Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies—a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.
- Ed S. Lein, Michael J. Hawrylycz, and Allan R. Jones. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124):168–176, Jan 2007. ISSN 1476-4687. doi:10.1038/nature05453. URL <https://doi.org/10.1038/nature05453>.
- Michael L Hines, Thomas Morse, Michele Migliore, Nicholas T Carnevale, and Gordon M Shepherd. Modeldb: a database to support computational neuroscience. *Journal of computational neuroscience*, 17(1):7–11, 2004.

- Perry WE Spratt, Ryan PD Alexander, Roy Ben-Shalom, Atehsa Sahagun, Henry Kyoung, Caroline M Keeshen, Stephan J Sanders, and Kevin J Bender. Paradoxical hyperexcitability from nav1. 2 sodium channel loss in neocortical pyramidal cells. *Cell Reports*, 36(5):109483, 2021.
- Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.
- Robert B Avery and Daniel Johnston. Multiple channel types contribute to the low-voltage-activated calcium current in hippocampal ca3 pyramidal neurons. *Journal of Neuroscience*, 16(18):5567–5582, 1996.
- I Reuveni, A Friedman, Y Amitai, and Michael J Gutnick. Stepwise repolarization from ca2+ plateaus in neocortical pyramidal cells: evidence for nonhomogeneous distribution of hva ca2+ channels in dendrites. *Journal of Neuroscience*, 13(11):4609–4621, 1993.
- Maarten HP Kole, Stefan Hallermann, and Greg J Stuart. Single ih channels in pyramidal neuron dendrites: properties, distribution, and impact on action potential output. *Journal of Neuroscience*, 26(6):1677–1687, 2006.
- Alon Korngreen and Bert Sakmann. Voltage-gated k+ channels in layer 5 neocortical pyramidal neurones from young rats: subtypes and gradients. *The Journal of physiology*, 525(3):621–639, 2000.
- Jacopo Magistretti and Angel Alonso. Biophysical properties and slow voltage-dependent inactivation of a sustained sodium current in entorhinal cortex layer-ii principal neurons: a whole-cell and single-channel study. *The Journal of general physiology*, 114(4):491–509, 1999.
- Costa M Colbert and Enhui Pan. Ion channel properties underlying axonal action potential initiation in pyramidal neurons. *Nature neuroscience*, 5(6):533–538, 2002.
- M Köhler, B Hirschberg, CT Bond, John Mark Kinzie, NV Marrion, James Maylie, and JP Adelman. Small-conductance, calcium-activated potassium channels from mammalian brain. *Science*, 273(5282):1709–1714, 1996.
- Bernardo Rudy and Chris J McBain. Kv3 channels: voltage-gated k+ channels designed for high-frequency repetitive firing. *Trends in neurosciences*, 24(9):517–526, 2001.
- PR Adams, DA Brown, and A Constanti. M-currents and other potassium currents in bullfrog sympathetic neurones. *The Journal of Physiology*, 330(1):537–572, 1982.
- Cheng Chu, Sang Kyun Kim, Yian Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.
- Roy Ben-Shalom, Nikhil Shridhar Athreya, Christopher Cross, Hersh Sanghevi, Kyung Geun Kim, Alexander Ladd, Alon Korngreen, Kristofer E Bouchard, and Kevin J Bender. Neurogpu, software for neuron modeling in gpu-based hardware. *bioRxiv*, page 727560, 2020.
- Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.
- Werner Van Geit, Michael Gevaert, Giuseppe Chindemi, Christian Rössert, Jean-Denis Courcol, Eilif B. Muller, Felix Schürmann, Idan Segev, and Henry Markram. Bluepyopt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Frontiers in Neuroinformatics*, 10:17, 2016. ISSN 1662-5196. doi:10.3389/fninf.2016.00017. URL <https://www.frontiersin.org/article/10.3389/fninf.2016.00017>.
- Pramod Kumbhar, Michael Hines, Jeremy Fouriaux, Aleksandr Ovcharenko, James King, Fabien Delalondre, and Felix Schürmann. Coreneuron: an optimized compute engine for the neuron simulator. *Frontiers in neuroinformatics*, 13: 63, 2019.
- Satya S. Sahoo, Annan Wei, Joshua Valdez, Li Wang, Bilal Zonjy, Curtis Tatsuoka, Kenneth A. Loparo, and Samden D. Lhatoo. Neuropigpen: A scalable toolkit for processing electrophysiological signal data in neuroscience applications

- using apache pig. *Frontiers in Neuroinformatics*, 10:18, 2016. ISSN 1662-5196. doi:10.3389/fninf.2016.00018. URL <https://www.frontiersin.org/article/10.3389/fninf.2016.00018>.
- F. Tadel, S. Baillet, J. C. Mosher, D. Pantazis, and R. M. Leahy. Brainstorm: a user-friendly application for MEG/EEG analysis. *Comput Intell Neurosci*, 2011:879716, 2011.
- Segundo Guzman, Alois Schlögl, and Christoph Schmidt-Hieber. Stimfit: quantifying electrophysiological data with python. *Frontiers in Neuroinformatics*, 8:16, 2014. ISSN 1662-5196. doi:10.3389/fninf.2014.00016. URL <https://www.frontiersin.org/article/10.3389/fninf.2014.00016>.