GPU Simulation Framework and Olfactory Bulb Model

Michael Rule, Monday June 14, 2010

Contents

Introduction and Overview

This document describes a spiking point-neuron network simulator running on an NVIDIA Tesla T10 card. Simple examples of spiking neuron simulations are presented, and the utility of graphics processing unit (GPU) spiking simulation in discussed. A simple model of an olfactory system, containing mitral cells and granule cells, and taking input in the form of Poisson spike generators as olfactory receptor neurons (ORNs), was constructed and tested using this framework. No periglomerular circuitry or short axon cells were modeled. Basic runtime demonstration and benchmarks are presented, as well a other notes on the model implementation and behavior.

GPU framework

GPU accelerated code can utilize massively parallel processors with thousands of cores. Repetitive computations that might have to be performed repeatedly over a large data set can be performed simultaneously, greatly accelerating certain classes of problems.

The GPU computation framework can be useful for :

- rapid parameter sweeps over thousands of copies of a simple model
- parameter surfing moderate complexity models, where the size of the model is slowing down the process of refining the parameters.
- Running scaled up large network versions of small to moderate complexity models that have been verified. (For the largest simulations, the framework isn't quite fast enough for rapid parameter surfing, and doesn't have enough memory to run many simulations in parallel, but it can run the network faster than a single CPU core.)

Due to the considerable additional investment in the time of developing a simulation on the GPU, the GPU computational framework might not be useful for :

- Rapid prototyping of small models
- Parameter surfing very simple models
- Models where a firing rate model or other fast, non-spiking simulation will suffice
- Quick building of one-off models that will be run only a few times
- People who don't have time to invest in learning a new programming framework

The graphics processing unit used throughout this document is an NVIDIA Tesla-T10 card, which has 240 cores. Under some conditions, these cores can execute additional instructions while waiting for memory reads and writes.

Using this pipelining technique, up to 512 threads can run per core. No more than 122,880 threads can run in parallel on a single Tesla-T10 device. The total number of parallel units in a large simulation is typically much higher than this, so it is easy to saturate the parallelism of available on the T10 device.

The SpikeStream and AtomicHedgehog projects provide application programming interfaces (APIs) for spiking neuron simulations from within the Python programming language. SpikeStream focuses on NVIDIA hardware, while AtomicHedghog is theoretically portable to other brands of graphics cards and even CPU clusters. Presently, an abstract interface had been written for defining new neuron models, making network connections, modeling synapse dynamics, and sending spikes. Several basic neuron models and synapses have been tested. There are future plans to implement spike timing dependent plasticity in AtomicHedgehog. SpikeStream is no longer under active development, but will be maintained. The simulations and demonstrations in this document focus on the SpikeStream API.

SpikeStream documentation :

```
file: SpikeStream/doc/build/html/index.html
file: SpikeStream/doc/build/latex/SpikeStream.pdf
file: SpikeStream/doc/source
file: SpikeStream/readme.pdf
file: SpikeStream/examples/*
```

The documentation for the SpikeStream API can be found in SpikeStream/doc. A generated website can be found at doc/build/html/index.html. This website is also hosted at http://spikestream.bitbucket.org/ as of June 14 20120. Additional information includes this document, and the source code for SpikeStream itself. AtomicHedgehog docmentation can be found at http://ahh.bitbucket.org. The SpikeStream/examples directory contains working SpikeStream demonstrations.

Links to Useful Reference Documentation

The following links provide necessary background for both the SpikeStream and AtomicHedgehog GPU simulation frameworks. Both frameworks are based on Python, which is an easy to learn interpreted language. Python has community supported scientific and numerical packages called “scipy” and “numpy”, as well as a plotting package “matplotlib”. Together, these packaged duplicate much of Matlab functionality. Scipy can also export matlab data files, if you prefer to preform analysis in Matlab. The OlfactoryBulb SpikeStream example contains a demonstration of saving simulation state for analysis in Matlab. The GPU acceleration component of SpikeStream and AtomicHedgeHog is written using PyCUDA and PyOpenCL, maintained by Andreas Klockner. Both CUDA and OpenCL are C-like languages that can be compiled to run on the GPU.

for an overview of the CUDA language and GPU computing, see
http://www.nvidia.com/object/what_is_cuda_new.html
http://developer.nvidia.com/object/gpu_programming_guide.html
http://en.wikipedia.org/wiki/CUDA

for an overview of the Python language see
http://docs.python.org/

for an overview of numpy see
http://mathesaurus.sourceforge.net/matlab-numpy.html
http://docs.scipy.org/doc/

for an overview of matplotlib see
http://matplotlib.sourceforge.net/

for an overview of PyCuda see
http://documen.tician.de/pycuda/

for an overview of iPython
http://ipython.scipy.org/moin/Documentation

for an overview of Sage
http://www.sagemath.org/

Basic Example 0 : Testing a single neuron

```
file: SpikeStream/examples/single_neuron.py
```

Running "ipython single_neuron.py" in the SpikeStream examples directory should produce figure 1A. This example runs a single Morris-Lecar neuron simulation. Examples 0,1,2 are not typical uses of spikestream : they are quite small and don't take full advantage of all the processors on the Tesla unit.

Basic Example 1:Running multiple copies of a single neuron

```
file: SpikeStream/examples/multi_neuron.py
```

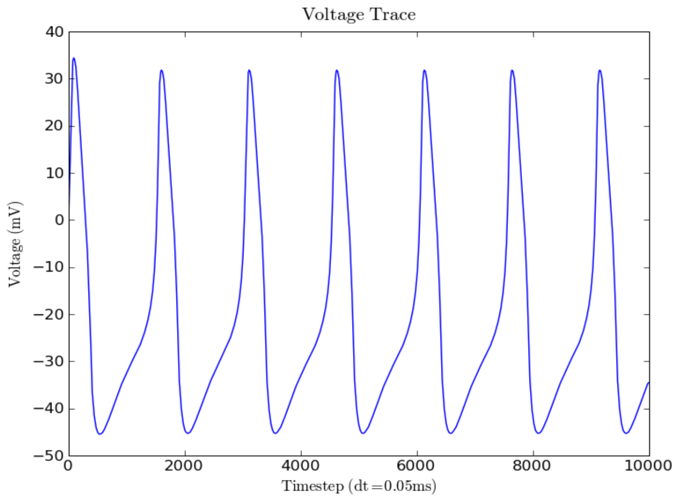
Running "ipython multi_neuron.py" in the SpikeStream examples directory should produce figure 1B. This example is simply example 0 set up to do a parameter sweep over different current inputs to the neuron model.

Basic Example 2 : Testing some more neuron models

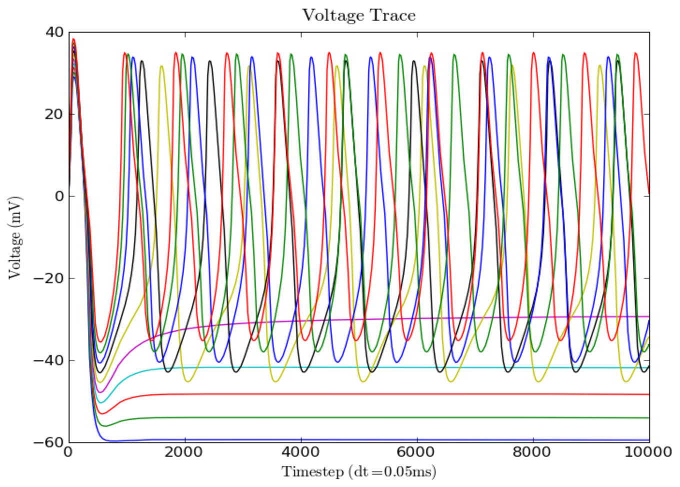
```
file: SpikeStream/examples/demo_models .py
```

Running "ipython demo_models.py" in the SpikeStream examples directory produces figure 1C. This demo feeds a 400Hz poisson spike train into several neuron models, including a leaky-integrate-and-fire (LIF), a LIF with a refractory period, and four parameter settings of the Izhikevich neuron model
(www.izhikevich.org/publications/whichmod.htm).

A



B



C

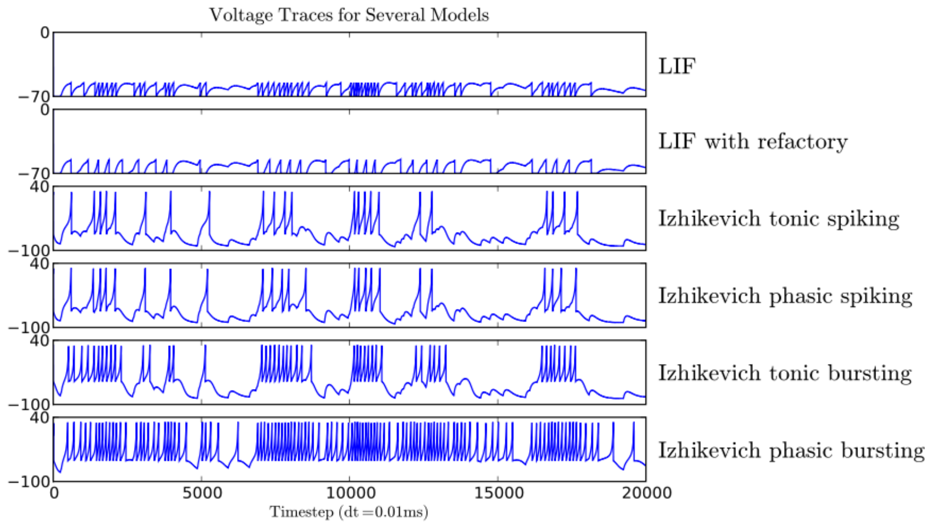


Figure 1 : plots from basic example scripts

The Model Olfactory Bulb

```
file: SpikeStream/examples/olfactory_bulb_multi_odor.py
file: SpikeStream/examples/olfactory_bulb_single_odor.py
```

This model of an Olfactory bulb contains three populations of point neurons : Olfactory Receptor Neurons (ORNs), Mitral Cells (MCs), and Granule Cells (GCs). In practice, all model parameters, including the number of each cell type, and the number and distribution of connections between layers, are variable and have been changed in the process of looking for specific behaviors, or making trade-offs between accuracy and computation time.

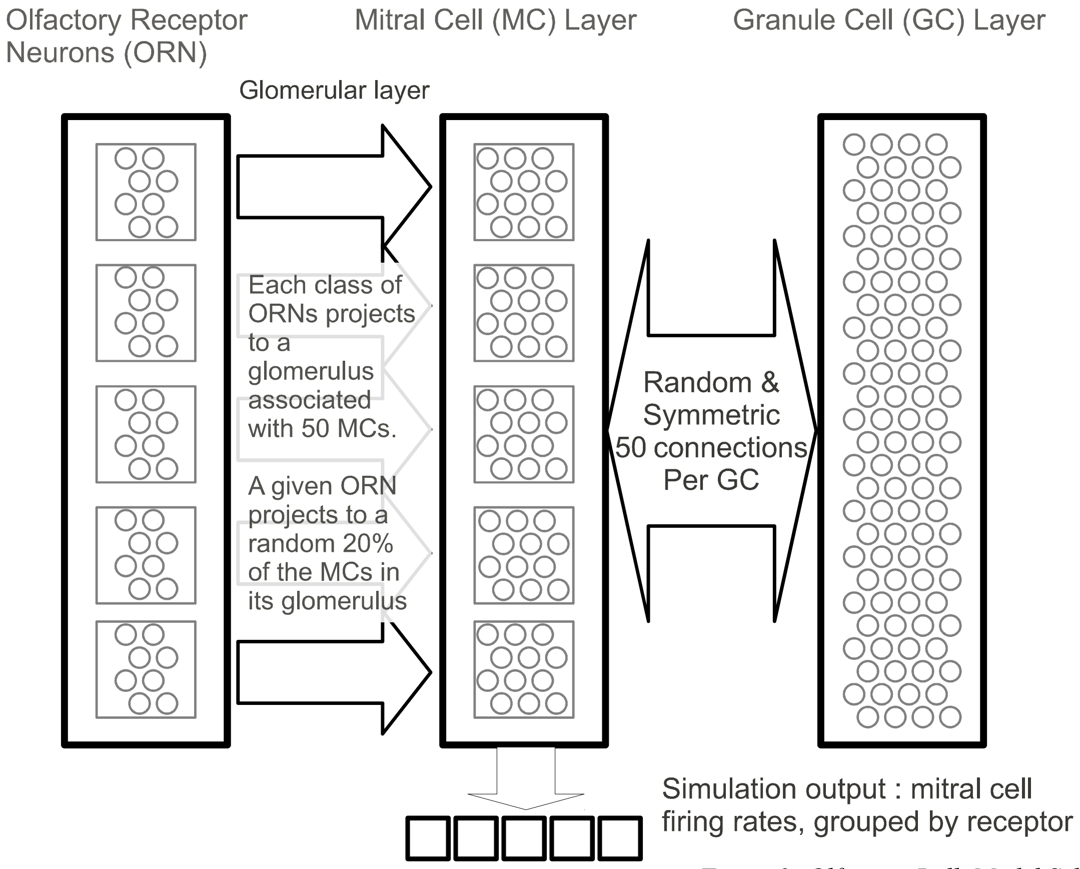

Olfactory Receptor Neurons (ORN)
Mitral Cell (MC) Layer
Granule Cell (GC) Layer
Glomerular layer
Each class of ORNs projects to a glomerulus associated with 50 MCs.
A given ORN projects to a random 20% of the MCs in its glomerulus
Random & Symmetric 50 connections Per GC
Simulation output : mitral cell firing rates, grouped by receptor

Figure 2: Olfactory Bulb Model Schematic

Olfactory Receptor Neurons (ORN)

In this model, there are typically 10 ORNs per receptor type, with up to a few hundred receptor classes. All ORNs are inhomogeneous poisson spike generators. Odor binding is simulated using the Hill equation, assuming non-cooperative binding, to compute the fraction of activated receptors given the pKd of an odorant and its concentration. The computed fraction bound is scaled linearly to a Poisson rate, with typically 100% bound corresponding to 200Hz firing. Each receptor type is represented by multiple ORNs, and each ORN projects to multiple Mitral cells in the corresponding glomerulus.

$$bound = \frac{concentration}{concentration + 10^{-\mathrm{pKd}}}$$

$$rate = bound \cdot \mathrm{max\_ORN\_rate}$$

$$p(fire) = \Delta t \cdot rate$$

Mitral Cell (MC) Layer

Typically, 50 MC per glomerulus are used. The Izhikevich mitral cell model from "Dynamical Systems In Neuroscience", section 8.4.5, is used to model this population. Mitral cells receive excitatory input from ORNs, send excitatory output to granule cells, and receive inhibitory inputer from granule cells. Typically, when considering the output of the simulation, the spikes of all mitral cells in a given glomerulus were pooled and used to estimate the firing rate response of a mitral cell associated with a given olfactory receptor.

$$\text{Spike : if V} \geq 35 \ \{ \text{V}=-50 \ \}$$
$$\frac{\partial V}{\partial t}=0.025((V+55)(V+50)+0.5(V_d-V)-U+I)$$
$$\frac{\partial V_d}{\partial t}=0.01235(V-V_d)$$
$$\frac{\partial U}{\partial t}=0.4((V<-48\,?\,0:20(V+48))-U)$$
$$I=2G_e^1(-10.0-V)+2G_i^1(-70.0-V)$$
$$\frac{\partial G_e^0}{\partial t}=-\frac{2}{\tau_i}G_e^0+\Delta g_e\sum_{\forall \text{ e spikes}}\delta(t-t_{\text{e spike}})$$
$$\frac{\partial G_i^0}{\partial t}=-\frac{2}{\tau_i}G_i^0+\Delta g_i\sum_{\forall \text{ i spikes}}\delta(t-t_{\text{i spike}})$$
$$\frac{\partial G_e^1}{\partial t}=\frac{2}{\tau_e}(G_e^0-G_e^1)$$
$$\frac{\partial G_i^1}{\partial t}=\frac{2}{\tau_i}(G_e^0-G_e^1)$$

ORN to MC feed-forward connections
Connections between ORNs and MCs are excitatory. Each ORN class is associated with a glomerulus of ~50 mitral cells, and projects onto a random 20% of MCs the given glomerulus. We use an alpha synapse model with tau=5ms, reversal potential of -10mV, and Δg=0.9μS (Δg refers to the instantaneous change in conductance upon receiving a spike).

Granule Cell (GC) Layer
Typically, the model used 5 GCs per MC. Granule cells were inhomogeneous Poisson spike generators, where the rate function was a scaling of a slow rising response to incoming spikes. Incoming spikes trigger slow rise and fall in the firing rate of a granule cell, with a peak at about 200ms. Granule cells connect symmetrically to a large number of mitral cells (n=50 to 200).

$$\frac{\partial G_g^0}{\partial t}=-\frac{3}{\tau_g}G_g^0+\Delta g\sum_{\forall \text{ incoming spikes}}\delta(t-t_{\text{spike}})$$
$$\frac{\partial G_g^1}{\partial t}=\frac{3}{\tau_g}(G_g^0-G_g^1)$$
$$\frac{\partial G_g^2}{\partial t}=\frac{3}{\tau_g}(G_g^1-G_g^2)$$
$$rate=G_g^2\cdot\text{granule\_rate\_scalar}$$
$$p(fire)=\Delta t\cdot rate$$

MC to GC connections

In the model, mitral cells connect to granule cells with symmetric feed-forward excitation and feed back inhibition. Self-inhibition of granule cells is not modeled. There are n≥5 GCs for each MCs in the current model. Each GC connects to n (n=50 in these simulations) randomly drawn MCs. A given MC will connect to many GCs. The MC to GC synapse is modeled as a very slow rising response to incoming spikes as part of the GC model. This response function is explained in the next section, and effectively integrates to :

$$g(t) = (t/\tau)^3 e^{3(1-t/\tau)}$$

This process reaches a peak of activation at t=200ms. The GC to MC is modeled as a fast inhibitory alpha synapse with tau=1ms, reversal potential of -70mV, and Δg=0.3s.

On-line Implementation of a Generalized Alpha-Function Synapse:

A generalized form of the simple alpha-function synapse $t/\tau\, g(t)=e^{1-t/\tau}$ might look like

$$g(t)=(t/\tau)^n e^{n(1-t/\tau)}, n\in\mathbb{N} \quad (1)$$

On-line GPU simulations cannot evaluate synapse response functions by convolution: an on-line representation is required. The exponential synapse can be implemented on-line by the differential equation $\tau\dot{g}=-g$. A similar on-line equation exists for the alpha synapse. The family of synapses above can also be implemented on-line, essentially by chaining together exponential synapses. State variable n exponentially decays toward the value of state variable n-1 with time constant τ . Up to a scaling of time and amplitude, the solution for the nth state variable is the nth equation from the above family.

$$\tau\dot{g}_0=-g_0$$

$$\tau\dot{g}_k=g_{k-1}-g_k$$

This solves to:

$$g_n(t)=\left(\frac{t}{\tau}\right)^n \frac{e^{-t/\tau}}{n!} \quad (2)$$

These have a peak at time $n\tau$. To rescale the time-constant so that the synapse peak arrives at time τ , use $\tau'=\tau/n$ as a modified time-constant. In this case, the on-line differential form of this synapse family becomes:

$$\frac{\tau}{n}\dot{g}_0=-g_0$$

$$\frac{\tau}{n}\dot{g}_k=g_{k-1}-g_k$$

The magnitude of the peak is also decreased for larger values of n. The amplitude of the peak is:

$$\frac{n^n}{n!}e^{-n} \underset{n\to\infty}{\approx} \frac{1}{\sqrt{2\pi n}}$$

If you want to normalize the peak amplitude, divide the output of the final conductance state variable by the above constant:

$$g_{out}=g_n\frac{n!}{n^n}e^n \underset{n\to\infty}{\approx} g_n\sqrt{2\pi n} \quad (3)$$

To verify that this actually works, equation (2) with $\tau'=\tau/n$ substituted, and scaled as in (3) does simplify to equation (1) :

$$g_n(t)=\left(\frac{t}{\tau/n}\right)^n \frac{e^{-t/(n\tau)}}{n!}\frac{n!}{n^n}e^n=\left(\frac{t}{\tau}\right)^n e^{n(1-t/\tau)}$$

Incidentally, I suspect that as n tends to infinity, this function approaches a log-normal distribution, but I can't prove it. This implies the following identity :

$$\lim_{n\to\infty} n\ln(x)+n^2(1-x^{1/n})=-\ln(x)^2/2$$

$$\lim_{n\to\infty} x^n e^{n^2(1-x^{1/n})}=x^{-\ln(x)/2}$$

Simulating Odor Stimuli

Dissociation constants for a range of odor-receptor pairs were unavailable. Consequentially, simulated binding data were used. It had not been verified that simulated odor or odor mixture binding affinities are realistic. Unit-less pKds were drawn from a normal, zero mean, unit variance distribution as in figure 3A. These simulated stimuli were presented in (unit-less) concentrations ranging from 10^{-4} to 10^{4} . To compute the fraction of a simulated odor bound to a given receptor type, the Hill equation was used, assuming non-cooperative binding. This generated a bimodal distribution of fractions-bound as in figure 3B. This fraction bound was scaled by 200Hz, and taken as the rate value for an inhomogeneous Poisson process as a stand-in for ORNs.

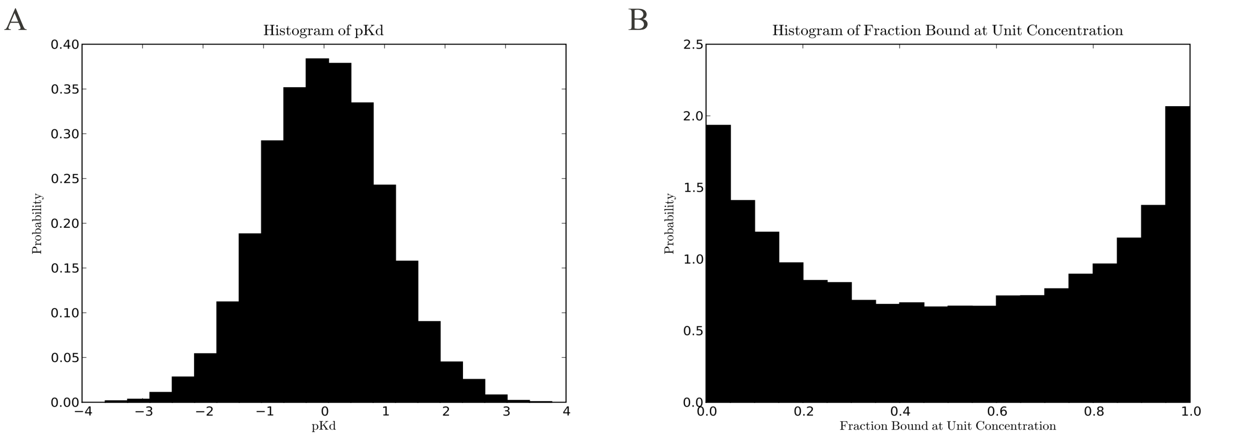


Figure 3 : Odor Stimuli Distributions

Demonstration of Simple Model Output

Demonstration setups of the olfactory bulb model are present in the SpikeStream examples directory. This model is set up to run, in parallel, several odors at a range of concentrations, in an olfactory bulb model with adjustable physical parameters. Figure 4 shows plots of various network statistics for two seconds of simulated odor presentation. Figure 5 shows how the mean network activity varies as the concentration of a simulated odor stimulus varies.

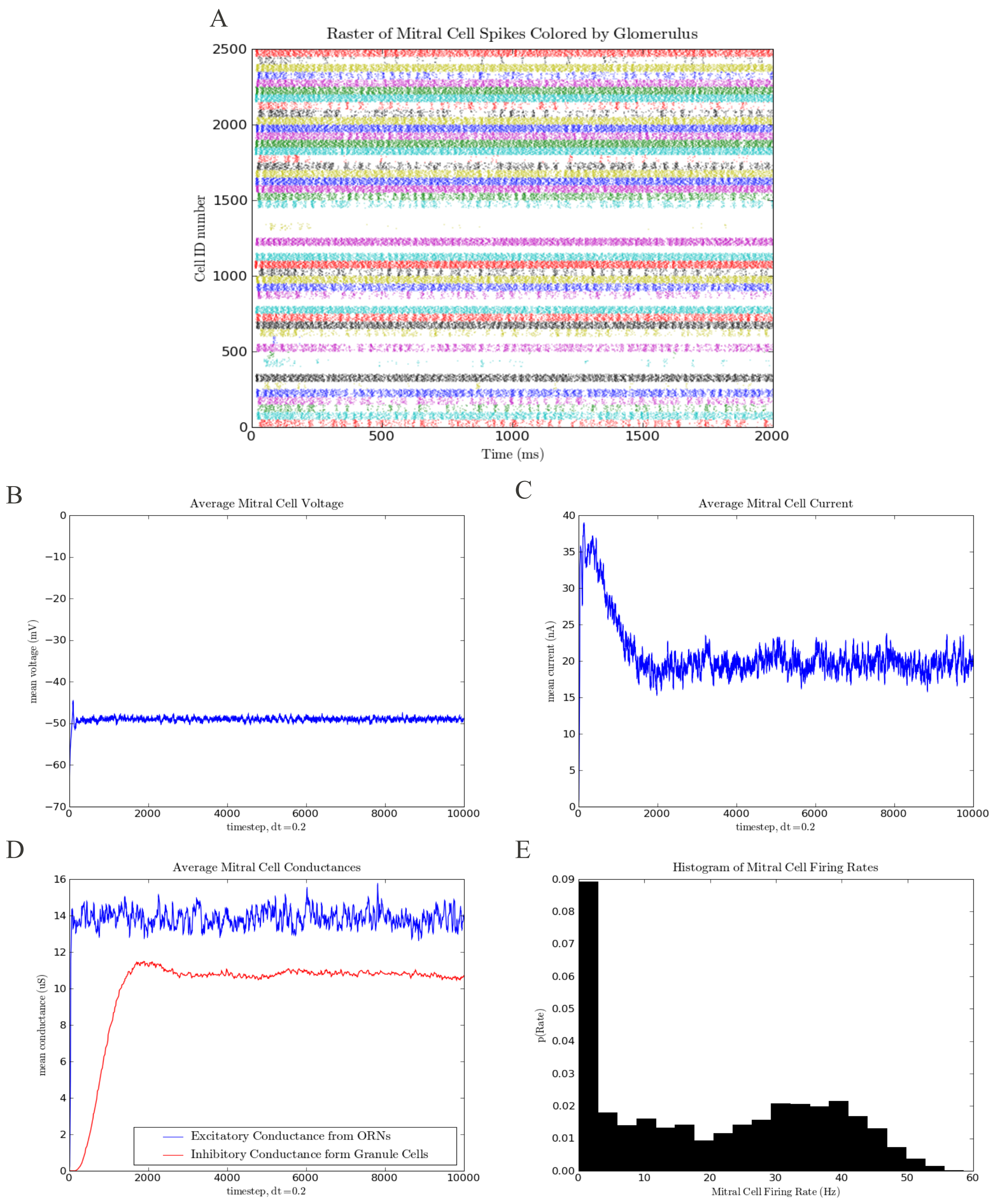


Figure 4, Single Odor Simulation Data : Two-second single odor stimulation of a network with 50 glomeruli. A : Spike raster of mitral cell population, colored by glomerulus. B : Average membrane potential in mitral cell population. C : Average current in the mitral cell layer. D : Average excitatory and inhibitory input conductances to mitral cells. E : histogram of mitral cell firing rates.

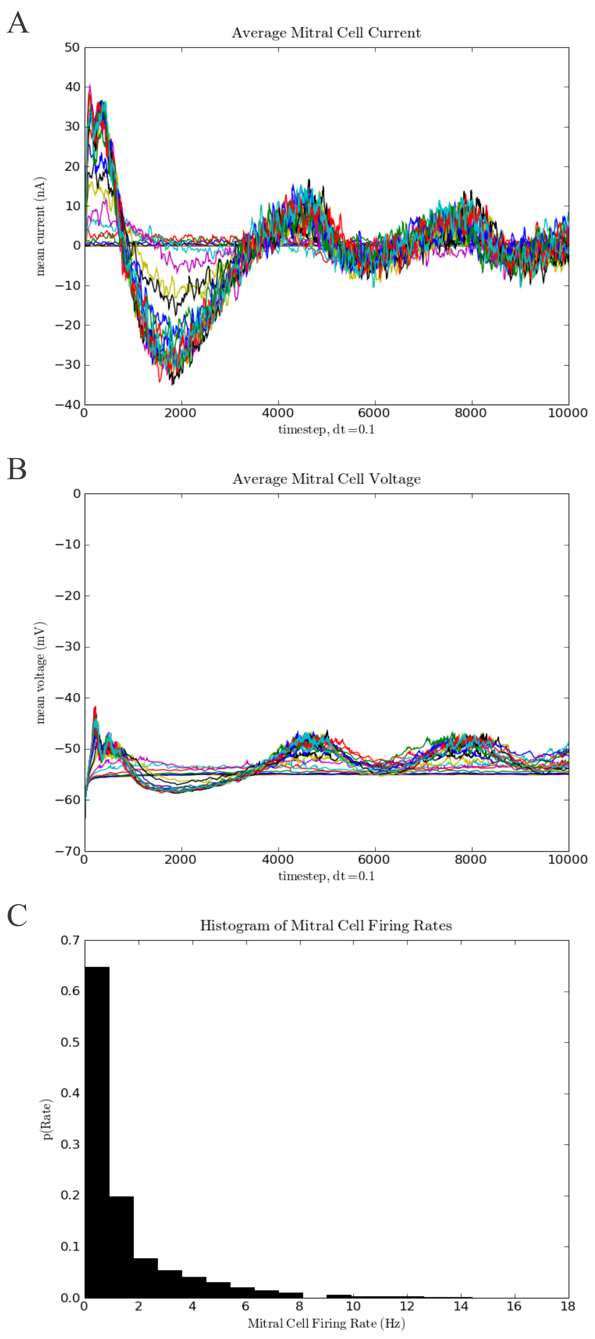


Figure 5 : Multiple Concentrations At Once : Half-second presentation of a single odor stimulus, simulated at a range of 25 concentrations from 10^{-4} to 10^{4} . As concentration increases, so does the magnitude of the population response and the duration of ringing oscillations at the time-scale of granule cell inhibition. A : Average current for mitral cell population across all 25 odor concentrations. B : Average voltage across mitral cell population for all 25 odor concentrations. C : histogram of mitral cell firing rates across all odor presentations.

Typical Simulation Runtime Summary

Figure 6 shows how the simulation time scales as more neurons are added to the simulation. Note that the number of units in these simulations is significantly larger than the number of parallel threads possible on the GPU. Therefore, the run-time complexity scales similarly to a non-parallel simulation, accelerated by a factor proportional to the number of individual threads possible on the T10 device. This runtime complexity scales quadratically as more units are added. This is likely a combination of the effect that, as the network becomes larger, the number of connections in the network grows as the square of the number of units. Memory bandwidth restrictions may also contribute.

$$\frac{\mu s}{\text{frame}} = 0.0526x^2 - 0.0473x + 1.1378, x \text{ in millions of units}, r^2 = 0.9987$$

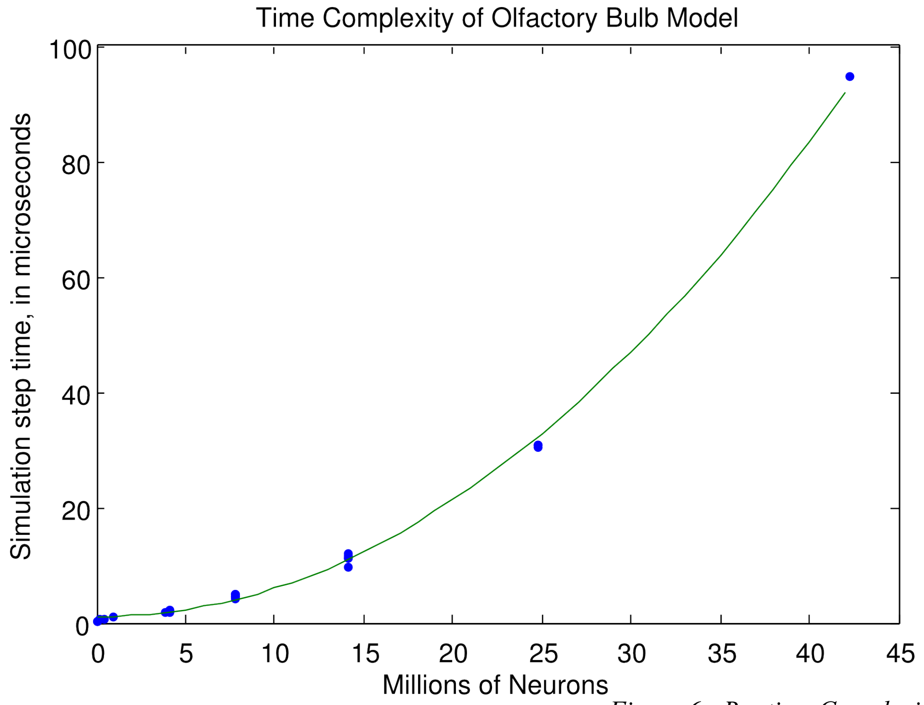


Figure 6 : Runtime Complexity of OB Model

Reading out data while the GPU simulation is running puts significant demands on memory bandwidth and can slow down a simulation. Using a model testing 100 odors for 10 receptor classes, recording all network spikes slowed the simulation speed from 630 frames per second to 308 frames per second. Turning on additional probes to record conductances, currents, and voltages in mitral cells, as well as binning spikes coming from glomeruli, slows the simulation speed to 30 frames per second.

The run-time of a simulation is a function of model structure, memory usage, spiking activity during simulation, and how the workload is distributed over the graphics card. GPU optimization and runtime are difficult to predict in general. SpikeStream makes only a minor attempt to optimize execution time, and does not use advanced optimizations of GPU memory access patterns. The use of atomic operations to send spikes causes serialization of threads if a single neuron receives more than one spike on a given time-step. We assume that such spike collisions are rare for realistic networks, but a dramatic slow-down occurs if the number of incoming spikes per neuron becomes large.

Transforming odor binding information into a concentration invariant form

It is possible to mathematically transform a vector representing the activation of olfactory receptors by an odor into a representation that is invariant of concentration. This same representation, if left un-normalized, will make the correlation between odor representation invariant of concentration, while preserving concentration information in the absolute magnitude of the vector.

Start with the hill equation, and assume non-cooperative binding of a single odorant or odor mixture "x". This equation can be used to compute the fraction of odorant bound ("b"), which ranges from 0 to 1.

$$\text{fraction bound} = b = \frac{[x]}{[x]+K_D}$$

Consider the problems of recognizing odors at very different concentrations [x]. For a single receptor and a single odorant, it is possible to exactly solve for the odorant concentration given the fraction bound. In reality a given receptor binds multiple odorants, and we do not know which odorant is currently binding. Therefore, K_D is not a known value. We can, however, compute $[x]/K_D$, even when the odor identity is unknown:

$$b = \frac{[x]}{[x]+K_D}$$

$$b([x]+K_D)=[x] \quad \Rightarrow \quad b[x]+bK_D=[x] \quad \Rightarrow \quad (1-b)[x]=bK_D \quad \Rightarrow \quad [x]=\frac{bK_D}{1-b}$$

$$\frac{[x]}{K_D}=\frac{b}{1-b}$$

Consider a model olfactory system with *n* distinct olfactory receptor subtypes, and many possible odors. Each odor is characterized by a unique vector of binding affinities for each receptor subtype

$$\boldsymbol{pKd_i}=(pKd_i^{0,} pKd_i^{1,}..pKd_i^{n}) \quad .$$

Let the binding of odorant *i* at concentration *x* be represented as as a length *n* vector

$$\boldsymbol{B_{i,x}}=(b_{i,x}^0, b_{i,x}^1, \ldots, b_{i,x}^n)$$

of fraction-bound values for all receptor subtypes. Solving for $[x]/K_D$ for each element leads to an alternative representation of odor binding :

$$\boldsymbol{A_{i,x}}=(\frac{[\mathrm{x}]}{K_{Di}^{0}}, \frac{[\mathrm{x}]}{K_{Di}^{1}}, \ldots, \frac{[\mathrm{x}]}{K_{Di}^{n}})$$

This can be written in terms of an odor-specific vector of the reciprocals of the dissociation constants for each receptor subtype, multiplied by the odor concentration as a scalar :

$$\boldsymbol{A_{i,x}}=[\mathrm{x}](\frac{1}{K_{Di}^{0}}, \frac{1}{K_{Di}^{1}}, \ldots, \frac{1}{K_{Di}^{n}})$$

Let

$$\boldsymbol{S_i}=10^{\boldsymbol{pKd_i}}=(\frac{1}{K_{Di}^{0}}, \frac{1}{K_{Di}^{1}}, \ldots, \frac{1}{K_{Di}^{n}}) \quad s.t. \quad \boldsymbol{A_{i,x}}=[x]\boldsymbol{S_i}, \quad \log_{10}(\boldsymbol{S_i})=\boldsymbol{pKd_i}$$

Many different forms of normalization will create a concentration invariant representation. Dividing the transformed vector by any global statistic dependent on concentration [x] will give you a concentration invariant representation.

For instance, normalization to magnitude 1 will send you into a concentration invariant space :

$$\frac{A_{i,x}}{|A_{i,x}|}=\frac{[x]S_i}{|[x]S_i|}=\frac{[x]S_i}{[x]|S_i|}=\frac{S_i}{|S_i|}$$

Alternatively, the z-score for any concentration will always equal the z-score of $\boldsymbol{S_i}$:

$$\frac{A_{i,x}-\mu_{A_{i,x}}}{\sigma_{A_{i,x}}}=\frac{[x]S_i-[x]\mu_{S_i}}{[x]\sigma_{S_{i,x}}}=\frac{S_i-\mu_{S_i}}{\sigma_{S_{i,x}}}$$

This result can be generalized to all functions that take information on fraction bound and transform it into a product of a function of the concentration and a function of a concentration invariant parameter.

$$f(b_i)=g([x])h(K_{D,i})$$

This includes the general case with cooperative binding exponents.

$$b=\frac{[x]^n}{[x]^n+K_D}\ \Rightarrow\ \frac{[x]^n}{K_D}=\frac{b}{1-b}$$

Real odor concentration values span several orders of magnitude, as do real K_D values. Therefore, it is generally not possible to work the the aforementioned transformed representations in real neurons with strictly positive firing rates limited realistically to a couple orders of magnitude representation (assuming a firing rate code). Even normalized representations will have a wide range of values owing to the variation in dissociation constants. Logarithmic representations might avoid some of these complications :

$$\log(\boldsymbol{A_{i,x}})=\log([x]\boldsymbol{S_i})=\log[x]+\log(\boldsymbol{S_i})=\log[x]+\boldsymbol{pKd_i}$$

Taking the z-scores of the log-transformed vector still gives you a concentration invariant response, and the result is an odor identity vector in terms of pKd. Subtracting out the mean is also sufficient. This suggests that some transformations of ORN binding information can transform odor information into a space where it is trivial to both classify, and determine the concentration, of a given odor percept.

$$\frac{\log(\boldsymbol{A_{i,x}})-\mu_{\log(\boldsymbol{A_{i,x}})}}{\sigma_{\log(\boldsymbol{A_{i,x}})}}=\frac{\log[x]+\boldsymbol{pKd_i}-\mu_{\log[x]+\boldsymbol{pKd_i}}}{\sigma_{\log[x]+\boldsymbol{pKd_i}}}=\frac{\log[x]+\boldsymbol{pKd_i}-\log[x]-\mu_{\boldsymbol{pKd_i}}}{\sigma_{\boldsymbol{pKd_i}}}=\frac{\boldsymbol{pKd_i}-\mu_{\boldsymbol{pKd_i}}}{\sigma_{\boldsymbol{pKd_i}}}$$

$$\boldsymbol{pKd_i}-\mu_{\boldsymbol{pKd_i}}=\log(\boldsymbol{A_{i,x}})-\mu_{\log(\boldsymbol{A_{i,x}})}=\log(\boldsymbol{B_{i,x}})-\log(1-\boldsymbol{B_{i,x}})-\mu_{\log(\boldsymbol{B_{i,x}})-\log(1-\boldsymbol{B_{i,x}})}$$