Simulation of full-size songbird HVC nucleus with spiking Hodgkin-Huxley type neuron models using CUDA and openMP. Yevhen Tupikov(yzt116)

Abstract

We simulated full-size songbird HVC with spiking Hodgkin-Huxley two-compartment HVC-RA and single compartment HVC-I neurons using CUDA and openMP. openMP implementation outperforms CUDA for small network size, while for large networks of more than a quarter HVC size CUDA is faster. For real HVC size with 20000 HVC-RA and 8000 HVC-I neurons CUDA is 2 times faster than openMP with 24 threads. It demonstrates that CUDA is well suitable for simulating dynamics of realistic size networks represented by complex neuron models and shows potential of running big simulations on a single computational node instead of using supercomputer capabilities.

Introduction

Neural networks have become an extremely popular tool used in machine learning research. Significant efforts were put to make network learning procedure efficient. Training of simple, almost linear neurons commonly used in artificial neural networks comes down to dense matrix-matrix multiplications for which GPU capabilities are well suitable. In contrast, biological neural networks usually use much more complex non-linear neuron models to capture the rich behavior of real systems. Neurons are commonly described by a system of ordinary differential equations, inspired by a classical research [1]. Communication between neurons is event-driven where neuron spikes trigger signal propagation. Therefore simulating biological neural networks can be split into two separate steps: 1.) Numerical integration of differential equations for neuron dynamics; and 2.) propagation of spikes to target cells. Depending on the network size and type of dynamics one can have different regimes, where either numerical integration or spike propagation steps consume the most computational resources. In this paper we investigate the efficiency of two different parallel approaches: openMP and CUDA of simulating real size songbird HVC

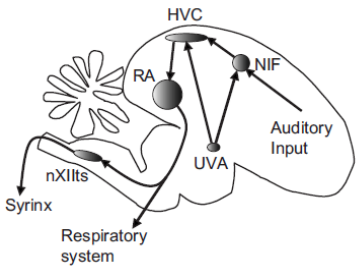

Figure 0.1: Main neuronal pathways in songbird song production system. Adapted from [3]

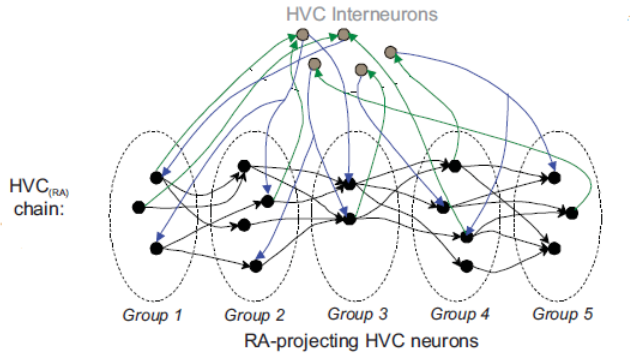

Figure 0.2: Synfire chain model of song production. Adapted from [3]

nucleus. We found that CUDA implementation is consistently faster for large network sizes with two times better performance for real HVC size.

Model

HVC nucleus in songbirds is believed to be the main center that controls the song production (see Fig.0.1). In particular, it is thought that precise and trial-to-trial reproducible spike patterns in HVC encode time in a song. There are two type of neurons that are related to song production: excitatory HVC-RA neurons that project to area RA and drive the song; and inhibitory HVC-I neurons that don't project outside HVC (thus also referred to as interneurons). In HVC there are around 20000 HVC-RA active during the song and 8000 HVC-I neurons. We model HVC-RA neurons as two-compartment Hodgkin-Huxley neurons and HVC-I neurons as a single compartment Hodgkin-Huxley. The details of the models can be found in [2]. Systems of ordinary differential equations for neurons are solved with classical Runge-Kutta order 4 numerical method. In the model neuron spikes if its membrane potential crosses the threshold value. For communicating spikes we use synaptic point model: spiked source neuron instantaneously sends signal to its targets, which update their input synaptic conductances. For more realistic results, both HVC-RA and HVC-I neurons receive stochastic external inputs, which are modeled as Poisson distributed input events. It results in fluctuations in membrane potentials of HVC-RA neurons and spontaneous activity of interneurons at around 10 Hz.

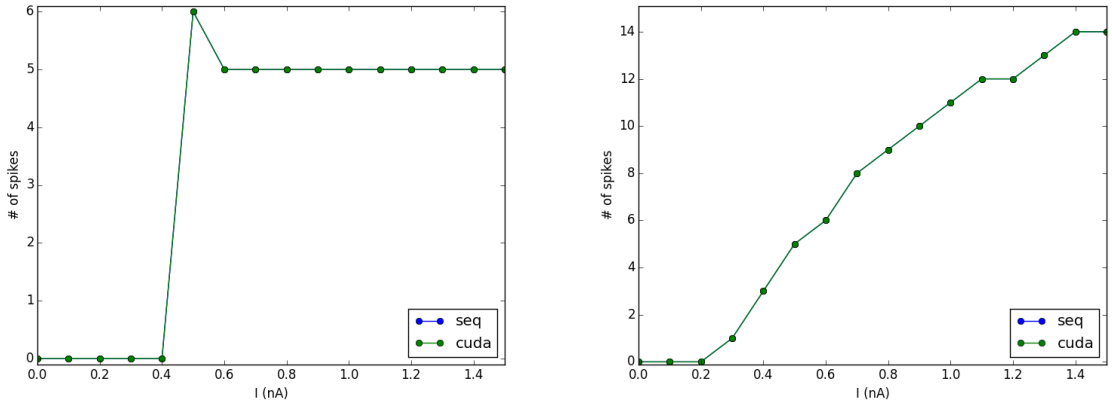

Figure 0.3: Left: HVC-RA response to the current pulse of 20 ms duration with different amplitude injected to dendritic compartment. CUDA and sequential versions produce same results. Right: HVC-RA response to the same stimulus injected to somatic compartment.

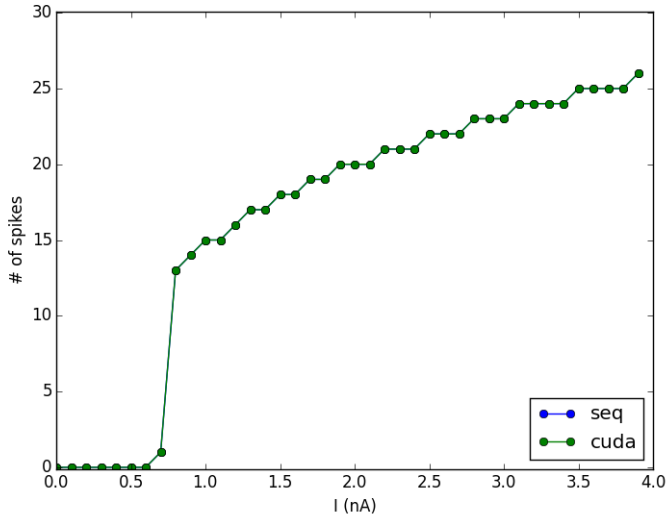

Figure 0.4: HVC-I response to the current pulse of 100 ms duration with different amplitude injected to dendritic compartment. CUDA and sequential versions produce same results.

We model precise and reproducible activity of HVC-RA neurons with popular synfire chain model. In this model neurons are organized in layers, where each layer is connected to the next one (see Fig.0.2). When the first layer is ignited by external input, it excites the second layer and, pretty much like a sequence of falling dominos, activity propagates along the chain. Each layer here represent a single time step in a song. All HVC-RA neurons in one layer are connected to all HVC-RA neurons in the next layer. Connections to and from interneurons are made randomly with probabilities $p_{RA \to I} = 0.002$ and

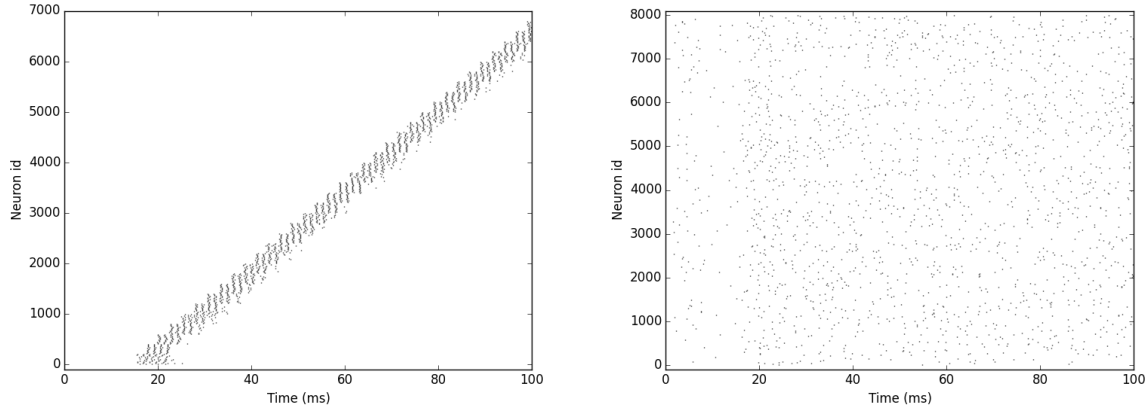

Figure 0.5: HVC neuron activity for 100 ms. Left: Typical example of activity of HVC-RA neurons. Right: Typical example of activity of HVC-I neurons.

$p_{I \to RA} = 0.01$. Connection strengths are sampled from uniform distributions: $G_{RA->RA} = [0, 0.02] mS/cm^2$, $G_{RA->I} = [0, 0.50] mS/cm^2$ and $G_{I->RA} = [0, 0.10] mS/cm^2$.

Implementation

In both openMP and CUDA versions of the simulations I had two main subroutines. First performed a single step of numerical integration of neuron dynamics and second performed updates of target conductances in case when some neurons produced spikes. In order to save resources I allowed neurons to run independently for 10 steps of dynamics (0.1 ms) and after that applied target conductance update. That becomes important for large networks with many active neurons when number of events is huge. openMP implementation was easily parallelized by splitting numerical integration steps among different threads. The only problem was generation of random numbers on different threads to model stochastic neuron inputs. I handled the problem by allocating separate noise generators for each individual thread. Second step of target updates was also simple to extend to the case with multiple threads. Each source neuron knows its target, thus it knows how to update the targets in case it spikes. The only concern is when same target is accessed to be updated from different threads. I solved this problem by making update operations of target conductances atomic: only one thread at a time could make an update of the same cell.

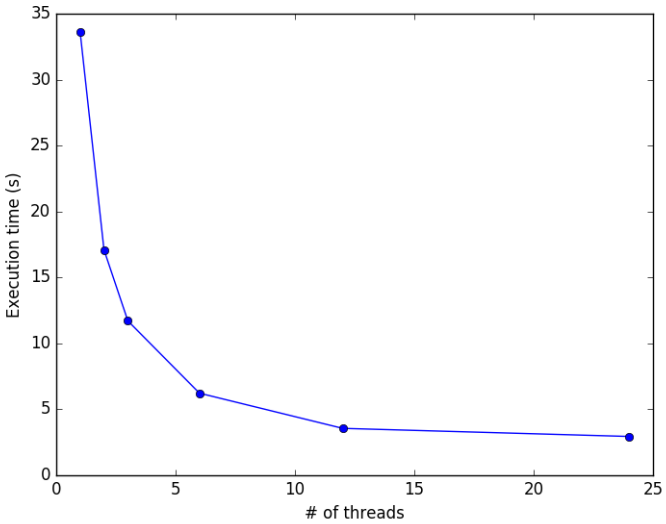

Figure 0.6: openMP version performance for different number of threads

CUDA version was parallelized in a similar way. Random numbers were generated using cuRAND library, which creates random generators for each thread. Update of target conductances was also done with atomic operation. However, since I used double floating precision in CUDA, efficient atomic operation was not supported by the standard. Thus I had to use less efficient atomic update suggested in the official manual.

To make sure that CUDA implementation was correct, I compared the responses of HVC-RA (see Fig.0.3) and HVC-I (see Fig.0.4) neurons to external inputs. Since they matched results for sequential versions I concluded that neuron implementation was correct. Further to ensure that targets were updated correctly I compared the results of identical network simulations obtained with openMP and CUDA. Typical results of the simulation are shown in the Fig.0.5. In the absence of stochastic inputs to neurons, spike patterns of simulations matched exactly. Thus I concluded that my CUDA and openMP implementations produce correct results.

For openMP version I analyzed how execution time scales with number of threads. Fig.0.6 shows that at 24 threads execution time almost flattens and further improvements in speed are hard to achieve.

To compare openMP and CUDA versions I run simulations of networks with different number of neurons, keeping all connection parameters constant. Comparison of openMP

and CUDA implementations shows that CUDA is performing worse for small network size (see Fig.0.7). However, for large networks with the size bigger than quarter of real HVC size, CUDA performs better and at real HVC size it is 2 times faster than openMP implementation.

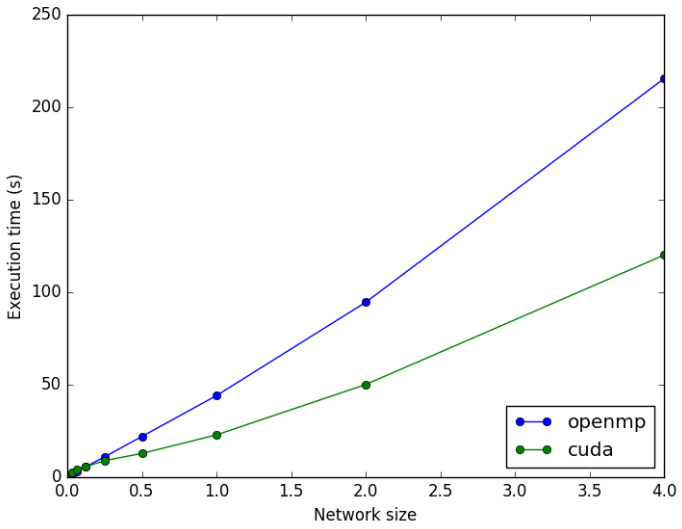

Figure 0.7: Comparison of execution times for CUDA and openMP for networks of different size

This results show that large number of cores in GPU handles large networks more efficiently than smaller number of openMP threads. Overhead in memory copying between host in device is small, since majority of time is spend in computations. Possible limitations are memory constrains of GPUs: you cannot allocate more memory than GPU has and his memory is much smaller than usual DRAM. Therefore simulations of really large neural networks (millions or even billions of neurons) are still hard to achieve.

References

[1] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of physiology, 117(4), 500-544.

[2] Long, M. A., Jin, D. Z., and Fee, M. S. (2010). Support for a synaptic chain model of neuronal sequence generation. Nature, 468(7322), 394-399.

[3] Jin, D. Z. (2009). Generating variable birdsong syllable sequences with branching chain networks in avian premotor nucleus HVC. Physical Review E, 80(5), 051902.