Ideas for Brian on GPU

# Current status

Issam has written algorithms for spike propagation on the GPU which show a big speedup compared to using only the CPU. Findings so far:

* Computing an array of spikes using compaction algorithms is always very slow, and should be avoided if possible. This is true even if we use state of the art compaction algorithms like chag:pp.
* For spike propagation, it is not necessary to compute the array spikes, because you can combine threshold+propagation+reset into a single kernel launch. This is much faster.
* For low numbers of synapses per neuron, the optimal algorithm for spike propagation appears to be using atomic float operations. For larger numbers of synapses per neuron, it may be that loading data into shared memory and using an algorithm designed by Romain may be better (not yet tested). (For dense matrices, there’s only one, obvious algorithm.)

# Immediate prospects

Adding delays and STDP should be possible with the current algorithms.

* Delays would be implemented as in Brian with a cylindrical event queue, and this still works on the GPU. This cylindrical event queue may change the balance between which algorithm is faster for low/high connectivity, as more data would need to be read into shared memory, making block sizes smaller.
* STDP mostly requires back-propagation, and this should be possible to implement in the same sort of way of forward propagation. STDP with delays may be more complicated.

# Incorporating GPU code into Brian

All the elements are now there to do this at least for some simple cases, but some work is needed to put the code into a form that can be used with Brian. Some issues that arise in the general case that need to be considered (in no particular order):

* Need code to convert data on the CPU into data on the GPU – most importantly for Connection objects.
* Initialisation can be done on the CPU or GPU – could write some algorithms for faster initialisation on the GPU (but this can be done later).
* There needs to be a way to communicate between the CPU and GPU for mixed simulations in which not all the code runs on the GPU (e.g. user code). In this case, we may need to do things like generating a spikes array even if it is not as efficient, because the CPU code requires it. We can have both systems (see Figure below).
* We need to have multiple versions of the GPU objects, e.g. for sparse connections we need a low and high connectivity version.
* At some point, we will need SpikeMonitor and StateMonitor objects. StateMonitor will be straightforward – can SpikeMonitor be written in a way that works nicely with the combined Threshold+Propagation+Reset in one kernel launch (to avoid having to compute the spikes array)?
* In the CPU version of Brian, we have a SpikeContainer object that is used for (amongst other things) handling subgroups. Is handling subgroups straightforward with Issam’s code or not? Might need to do some code generation here, i.e. building a custom kernel that depends on the set of subgroups, connections, etc.
* We haven’t considered network operations – these can be done on the CPU – but maybe we can also have an option for users to provide GPU code versions of them?
* If all Brian objects have GPU versions of them, and no CPU/GPU interactions are necessary, we can run everything on the GPU more efficiently (no Python needs to be run at all). Or alternatively, we can buffer it – e.g. run one second of biological time only on the GPU, do some CPU/GPU interactions and some Python stuff, then run another second of biological time, etc.

Figure - structure of CPU/GPU interactions

State update

Threshold + Propagation + Reset

Threshold

Propagation

Reset

GPU

CPU

values

Array: spikes

# Things to be done

Issam’s GPU code needs to be cleaned up and restructured to allow us to use it with Brian (and therefore with PyCUDA) and with code generation. I suggest the following basic structure: follow the boxes in the figure above. Have one file for each box. These files and the functions within them should have names corresponding to the names in the boxes (state update, threshold, etc.) to fit with the standard naming scheme in Brian.

Each file/function will probably be used with code generation. For example, the state update code will almost entirely be generated from user equations. We need some standard way of writing these functions as templates so that the final version can be generated from them. For the moment, inserting comments into the files explaining where code could be added will do.