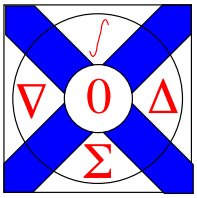

Computational Science Technical Note **CSTN-149**

Simulating Anaesthetic Effects on a Network of Spiking Neurons with Graphics Processing Units

A. Leist and C. J. Scogings and K. A. Hawick

2012

The structure and emergent behaviours of neuronal networks remain important unknowns but can be investigated by computer simulation of biologically plausible networks of microscopically simple individual neurons. We describe a software model developed to simulate in excess of 10^6 individual Izhikevich neurons with connectivities of in excess of 100 connections per neuron. We simulate the effect of adjusting some of the microscopic neuronal parameters and observe emergent oscillatory phenomena that relate to the introduction of anaesthetic drugs on the collective neuronal system. We report some preliminary computational performance results and comment on the feasibility of simulating realistic sized collective networks of cortical spiking neurons.

Keywords: GPU; CUDA; spiking cortical neurons; anaesthesia

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-149,
        author = {A. Leist and C. J. Scogings and K. A. Hawick},
        title = {Simulating Anaesthetic Effects on a Network of Spiking Neurons with
                Graphics Processing Units},
        booktitle = {Proc. International Conference on Bioinformatics and Computational
                Biology (BIOCOMP'12)},
        year = {2012},
        pages = {236-242},
        address = {Las Vegas, USA},
        month = {16-19 July},
        publisher = {CSREA},
        institution = {Computer Science, Massey University},
        keywords = {GPU; CUDA; spiking cortical neurons; anaesthesia},
        owner = {kahawick},
        timestamp = {2012.05.03}
}
```

Technical Report CSTN-149

Simulating Anaesthetic Effects on a Network of Spiking Neurons with Graphics Processing Units

A. Leist, C.J. Scogings and K.A. Hawick
Computer Science, Institute for Information and Mathematical Sciences,
Massey University, North Shore 102-904, Auckland, New Zealand
Email: { a.leist, c.scogings, k.a.hawick }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

March 2011

ABSTRACT
The structure and emergent behaviours of neuronal networks remain important unknowns but can be investigated by computer simulation of biologically plausible networks of microscopically simple individual neurons. We describe a software model developed to simulate in excess of 10^6 individual Izhikevich neurons with connectivities of in excess of 100 connections per neuron. We simulate the effect of adjusting some of the microscopic neuronal parameters and observe emergent oscillatory phenomena that relate to the introduction of anaesthetic drugs on the collective neuronal system. We report some preliminary computational performance results and comment on the feasibility of simulating realistic sized collective networks of cortical spiking neurons.

KEY WORDS
GPU; CUDA; spiking cortical neurons; anaesthesia

1 Introduction

The problem of understanding neuronal processes and structures in the brain [1, 2] is a long standing one. Of particular interest are those emergent collective properties that are thought to arise from the complex network structure [3] and nature of the brain rather than necessarily from microscopic details of individual neurons. One process of particular interest is the manner in which cortical neural activity [4] rises and falls in states of consciousness. Campbell and others have suggested an intriguing way to study this through simulating the action of an anaesthetic drug [5] on individual neurons [6–8] linked together in an artificial network structure.

Although there are many outstanding questions and unknowns concerning real brain structure we have constructed a simulated neural network structure on the assumption that there are likely some emergent properties that we may observe due to the sheer size of a suitably simulated network of many interacting individual neurons. In this paper we discuss some preliminary simulation software development work in scoping the computational feasibility of simulating large ensembles of individual neurons [9] that are arranged in structures and with connectivities that are at least plausible if not biologically justified in detail [10].

In particular we describe our use of massively data parallel computing techniques and graphical processing units (GPUs) with many individual cores, to simulate around 10^6 individual neurons arranged in regular and small-world interconnected networks [11–13]. We focus on the use of the Izhikevich neural model [14] of cortical spiking neurons [15] for the work reported here. Our simulation software apparatus could be readily adapted to use other neuronal models such as the Hodgkin-Huxley neuronal model [16].

Our article is structured as follows: In Section 2 we summarise the properties of the individual neuronal model we use. The simulated network structures are described in Section 3 and details of our data-parallel Compute Unified Device Architecture (CUDA) implementations for GPUs are given in Section 4. We discuss some of the emergent properties and features of our simulated system in Section 6, in which we also offer some tentative conclusions and suggested areas for further work.

2 Neuronal Model

We use the Izhikevich model [14] to simulate the spiking and bursting behaviour observed in cortical neurons [17]. Although more biophysically meaningful models do exist, including the well known Hodgkin-Huxley [16] model, the computational demands of these models are significantly higher and the size of the simulated neuronal networks is thus more limited. The aim of this paper is to lay the computational foundations for further studies of large-scale neuronal networks, specifically in terms of the spatial and temporal effects of anaesthetic drugs on the neural interactions. Size matters for simulations of such complex systems, as some macroscopic behavioural patterns only emerge for large systems with many microscopic interactions or when system properties are analysed over several length-scales. This is not to say that the quality of the model is not relevant, of course, as a system that does not exhibit realistic behaviour is essentially useless. However, in [15], Izhikevich compares a number of commonly used models and shows that the model proposed in [14] is computationally much cheaper than the Hodgkin-Huxley model, but nevertheless capable of reproducing realistic spiking and bursting behaviour.

The model uses four parameters, which can be adjusted to produce different spike patterns, such as those observed for real excitatory and inhibitory neurons. These spikes – which are also called action potentials – produce an electrochemical impulse that is transmitted to connected cells. Action potentials created by excitatory cells depolarise the membrane potentials v of their neighbouring cells and, thus, decrease the distance to their spike thresholds, making them more likely to "fire" an action potential of their own. Spikes created by inhibitory cells, on the other hand, hyperpolarise the membrane potentials and increase the distance to the threshold. Neurons are connected through transmitting fibres called *axons* and receiving fibres called *dendrites*. The *synapse* is the axon-dendrite junction. A postsynaptic neuron receives a postsynaptic potential after a suitable delay δ from the time the presynaptic action potential has been generated. But the postsynaptic potential does not apply all at once, it rather diminishes exponentially over a period of time as suggested in [18]. Our implementation uses separate washout tables $W_{e,i}$ for **e**xcitatory and **i**nhibitory neurons, which define the gradual washout as $W(t) = Ae^{-t/\tau}$, where t is time measured in simulation steps, $A_{e,i}$ regulates the voltage amplitude and $\tau_{e,i}$ defines the exponential washout rate. The washout table is chosen to be of length 3τ, which allows the action potential to diminish to about 5% of its base value before it stops having any effect.

In addition to the incoming currents from action potentials generated by presynaptic neurons, every neuron also receives an input current I. This is used to simulate currents received from sources external to the cortex, for example other parts of the brain – like the thalamus – or any other part of the nervous system. I is calculated individually for every neuron and at every time step as $I = I_{e,i} + I_{noise} + I_{boost}$, where $I_{e,i}$ is the base current for all excitatory (I_e) or inhibitory (I_i) cells. $I_{noise} = n_{e,i} \times r_{norm}$ is random noise computed from a base noise value $n_{e,i}$, times a normally distributed random variable. $I_{e,i}$ and I_{noise} are used to simulate a constant source of activity that drives the cortex. I_{boost}, on the other hand, is meant as a temporary boost, like an external shock delivered to the neuronal network. It is applied to a fraction of all cells selected at random during each simulation step that the boost is active. Although I_{noise} can be negative, the sum of these inputs I is not allowed to fall below zero.

The effects of anaesthetic drugs are modelled using parameters $\lambda_{A(e,i)}, \lambda_{\tau(e,i)}$ and $\lambda_{I(e,i)}$, which are defined individually for excitatory and inhibitory neurons. Different combinations of these values can be used to simulate different types of anaesthetics. The λ values can be updated between simulation steps. They modify the corresponding base values to get the effective values as follows:

For excitatory neurons:	For inhibitory neurons:
$A = A_e / \lambda_{Ae}$	$A = A_i \lambda_{Ai}$
$\tau = \tau_e / \lambda_{\tau e}$	$\tau = \tau_i \lambda_{\tau i}$
$I = I_e - \lambda_{Ie} + 1$	$I = I_i + \lambda_{Ii} - 1$

Thus, values of $\lambda > 1$ simulate drugs that have a dampening effect when applied to excitatory neurons and a strengthening effect when applied to inhibitory neurons. No drugs are administered when $\lambda = 1$.

3 Network Model & Data Structure

A 2-dimensional lattice is used to assign a unique global ID to each neuron – which can be calculated from its (x, y)-coordinates – and to restrict connections between neurons to a maximum distance r that is defined by the user. The one-way nerve connections are generated at random using a normal distribution

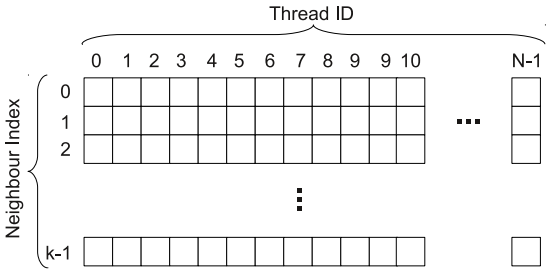

Figure 1: The one-way connections between neurons are stored in a 1D array of length $N \times k$ – here illustrated as a 2D array with N columns and k rows – where N is the system size and k is the in-degree. Each one of these arcs is only identified by the global neuron ID of its source node. The destination node is determined by the position in the array. As neurons are processed in the order of the thread IDs given in Figure 2, the source IDs are stored at index $n_{idx} \times N + tid$, where n_{idx} is the n's neighbour of the neuron that is processed by the thread with ID tid. This ensures fully coalesced memory transactions when accessing this array.

that is centered on the postsynaptic neuron, that is, the end of the arc in graph terminology. The standard deviation of the distribution is set to $\sigma = r/3$ and the distance is strictly restricted to $\leq r$. This results in a network structure where most connections are relatively short, with a decreasing number of longer distance connections. While the number of outgoing connections varies between neurons, every neuron has the same number of incoming connections. This, together with the fact that arcs are stored in the adjacency-list of the postsynaptic cell as described in the caption of Figure 1, is an important optimisation that significantly improves the utilisation of the GPU's memory bandwidth. It makes it possible to use an information pull-model for the transmission of action potentials. This process is described in more detail in Section 4. Note that even though the neurons are addressable by their coordinates in the grid, the graph itself is far from regular, due to the way neighbours are selected. It is possible to change the graph structure without any modifications to the algorithm, as long as the invariant of having the same number of incoming connections per neuron is maintained. This flexibility allows for plenty of future experimentation and a small-world network [12, 19] may be a particularly interesting candidate.

The lattice structure is also used to determine which CUDA thread gets to process a particular neuron when it is time to evolve the simulation by another step. This mapping is illustrated in Figure 2. It

0	1	2	3	16	17	18	19
4	5	6	7	20	21	22	23
8	9	10	11	24	25	26	27
12	13	14	15	28	29	30	31
32	33	34	35	48	49	50	51
36	37	38	39	52	53	54	55
40	41	42	43	56	57	58	59
44	45	46	47	60	61	62	63

Figure 2: The mapping of CUDA threads to neurons. The numbers represent the ID of the CUDA thread (tid) that processes a particular neuron. The actual implementation uses blocks of size 16×16 instead of the 4×4 blocks shown here. Independent from the tid, the 2D lattice is used to assign a global ID to each neuron. This global ID is implicitly defined by the position in the grid using row-major ordering from the top left to the bottom right. Note that the lattice structure does not reflect the actual neuronal network, it is only used to assign a unique global ID to each neuron and to restrict the neighbour selection to a given radius r.

is chosen to maximise the data locality for threads within the same thread block when querying information from neighbouring cells, taking advantage of the texture cache available on CUDA GPUs[1] and the knowledge that cells located close to each other on the lattice are more likely to share some of their neighbours than cells separated by a larger distance.

Whether a neuron is excitatory or inhibitory as well as the exact parameter values that determine its spike patterns are all determined when the graph is generated using the approach suggested in [14]. This leads to an approximately 4:1 ratio of excitatory to inhibitory cells. The type information needs to be available when the state of a neuron is queried, because postsynaptic neurons need to know what effect an incoming action potential has on their membrane potential. To do this efficiently, the type and the current firing state of each neuron are bit-packed into a single byte. All these bytes are stored in a two-dimensional array D_d that is addressed using a cell's (x, y)-coordinates. 2D texture fetches are used to retrieve the data when iterating over the adjacency-lists to identify any incoming action potentials. As only

[1] See [20, 21] for details about the CUDA architecture.

the two least-significant bits are used, the data could be compressed even more by storing the information of four neurons in a single byte. However, this is not done in the current implementation, as it is expected that future versions will make use of this space to store additional state information.

While all elements in D_d are read many times during each simulation step, once for every nerve connection originating from the respective neuron, the following arrays are all used to store data that is only read and updated by the thread that processes the cell it belongs to:

- V_d and U_d record the current values of the Izhikevich variables v (membrane potential) and u (membrane recovery).
- Δ_d stores the synaptic delay δ for each neuron. A minimum delay of $\delta_{min} = 5$ and a maximum delay of $\delta_{max} = 15$ milliseconds are used in the current implementation. The delay is defined on the postsynaptic neuron and not on the link itself. This simplification reduces the memory requirements for the delay terms from $\mathcal{O}(Nk)$ to $\mathcal{O}(N)$. The values are randomly initialised within the given range.
- EV_d is the extra volts array. It records the effective input voltage from action potentials – both excitatory and inhibitory – generated by all presynaptic neurons over the last $w + \delta$ simulation steps, where w is the current washout table length. Because $w = 3\tau$ and τ can be modified by $\lambda_{\tau(e,i)}$, a maximum washout table length W_{max} is defined at compile time. Based on this, $EV_{max} = W_{max} + \delta_{max}$ space is allocated for every neuron.
- TV_d records the type values that define the spike dynamics of individual neurons.

As each neuron only requires its own data, these arrays are indexed using the thread ID tid and transactions are fully coalesced. The only exception to this are transfers to and from EV_d during `Phase1` of the simulation, which are only partially coalesced. The reason being that the index used to access EV_d during this phase depends on the neural delay δ, which is initialised randomly for each cell.

Next, memory for T instances of the CUDA implementation of the 64-bit random number generator `Ran` from Numerical Recipes [22] is allocated. T is the number of CUDA threads used to process the system. The optimal value for T depends on the execution hardware, but it should be a large power of two smaller or equal to the system size, which is always a power of two itself. Every thread thus processes an equal number of neurons. $T = 2^{19}$ is used for the performance measurements, except when $N < 2^{19}$, in which case $T = N$.

Finally, arrays V_{sum} and F_{sum} are used to record partial sums of the membrane potentials and firing rates of all excitatory neurons. For reasons explained in the following section, these arrays are of length $32\times$(number of thread blocks).

4 CUDA Implementation

The simulation is split into two distinct phases and each of these phases is implemented as a CUDA kernel. Every simulation step executes both phases, advancing the simulation by one millisecond of model time. The control flow and the secondary tasks processed by the host system are described by the pseudo-code in Algorithm 1. The main task of `Phase1` is to update the membrane potentials v using the equations proposed by Izhikevich [14] for all neurons based on the various input currents reaching each cell at the current time step. This includes the external currents modelled by I as well as the inputs from all presynaptic neurons that have generated an action potential during time steps $[-(\delta+w), -(1+\delta)]$ from the current time as recorded in EV_d. When a neuron's membrane potential reaches 30mV, then it fires a new action potential. This event is recorded by setting the respective bit in array D_d.

The only data that needs to be moved between the host and the device at every time step – not counting kernel parameters – are arrays V_{sum} and F_{sum}. The average membrane potential of all excitatory neurons is used to plot a pseudo-EEG and to compute the power spectral density. The firing rate is plotted as an additional visual indicator of the current cortical activity. Making the data immediately available to the host makes it possible to monitor the simulation as it is running and to store the generated data for later reuse. As mentioned before, both of these arrays are of length $32\times$(number of thread blocks) and not of length N. The reason for this is that kernel `Phase1` performs a parallel reduction to compute the sum of the respective values for all neurons processed by the threads that belong to the same thread block. The reduction is done in the fast on-chip shared memory and the process is only stopped when the number of elements in the input reaches the warp size of 32 threads, at which point the first warp in the thread block writes the partial sums to V_{sum} and F_{sum} respectively. These ar-

Algorithm 1 This host function is called to evolve the simulation by `STEPS` simulation steps. The generated values v_{avg} and f_{avg} are the average membrane potential and the average firing rate of all excitatory neurons at the current time step. The former can be used to plot a pseudo-EEG and to compute the power spectral density.

determine the current washout table length w
for $s \leftarrow 1$ to `STEPS` **do**
  do in parallel on the device using T threads: call kernel `Phase1`(EV_{idx})
  wait until `Phase1` is completed
  increment EV_{idx}, the index into the extra volts array EV_d //wraps around when it reaches the end of the array
  copy V_{sum} from device memory to host memory //asynchronous, can overlap with kernel `Phase2`
  copy F_{sum} from device memory to host memory //asynchronous, can overlap with kernel `Phase2`
  do in parallel on the device using T threads: call kernel `Phase2`(EV_{idx}, w)
  wait until V_{sum} and F_{sum} have been copied to host memory
  $v_{avg} \leftarrow f_{avg} \leftarrow 0$ //the average voltage v and firing rate f of all excitatory neurons
  for $i \leftarrow 0$ to $(32 * [\text{number of thread blocks}])$ **do**
    $v_{avg} \leftarrow v_{avg} + V_{sum}[i]$
    $f_{avg} \leftarrow f_{avg} + F_{sum}[i]$
  end for
  $v_{avg} \leftarrow v_{avg}/ntype0$ //$ntype0$ is the number of excitatory neurons
  $f_{avg} \leftarrow f_{avg}/ntype0$
  wait until `Phase2` is completed
end for

rays are then copied to host memory, where the CPU can sequentially perform the remaining summation. The memory copies and CPU processing can be overlapped with the execution of kernel `Phase2` and, therefore, do not add to the overall runtime.

In `Phase2`, every neuron queries the current state of all its neighbours using texture fetches from array D_d as discussed in the previous section. To be able to do this, the global IDs of the presynaptic neurons – which can be used to compute the textures coordinates – are looked up from each neuron's adjacency-list using fully coalesced data transfers as explained in Figure 1. The data from D_d is then used to determine the number of new excitatory and inhibitory action potentials generated by all neighbours. Then, the next w values of the extra volts array EV_d are updated according to the number of inputs, using the values provided in the washout tables $W_{e,i}$ to compute the resulting excitatory and inhibitory effects over time. The fact that all threads update the next w elements of their neuron's extra volts array, starting from the same offset into EV_d, is very important. It means that all w reads and w writes per neuron performed during this phase are fully coalesced. This easily makes up for the single read and write per cell that is only partially coalesced in `Phase1`. The washout tables are stored in constant memory and can be accessed very quickly. The entire process of using the extra volts array is visualised in Figure 3.

As the performance results given in the next section show, the system size is mainly limited by the on-board memory of the GPUs used to run the simulation. In order to be able to simulate much larger systems, or to reduce the execution time, a multi-GPU implementation has been developed. It can be executed either on a single host machine with multiple GPUs or on a cluster of machines with one or more GPUs each. OpenMPI is used for the communication between cluster nodes. Figure 4 describes how the neuronal network is split into multiple components and how the communication time between GPUs can be at least partially hidden by computation.

Please refer to [23] for more details about the CUDA implementations of the different simulation phases and the modifications necessary for the cluster implementation. The reference provides detailed code listings and additional information about the model and its configuration options. It also describes the signal processing techniques that are used to obtain a power spectrum from the generated pseudo-EEG.

5 Performance Results

This section shows how the CUDA implementation performs when executed in batch-mode. In this mode, the results are not visualised in real-time. Instead, the average membrane potentials and firing rates are written to a file. While the graphical user interface is very useful when testing the effects of certain parameter combinations on the model, more extensive parameter value range scans are generally performed in batch-mode, with an automatic extraction of relevant metrics following the simulation run. A configu-

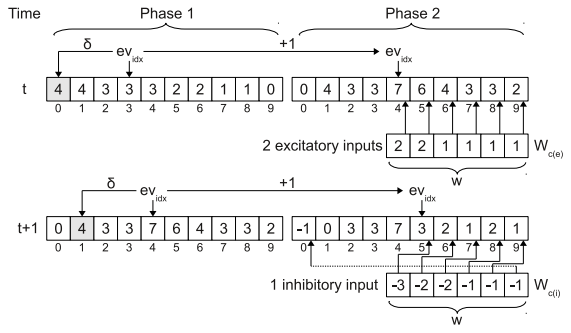

Figure 3: This diagram illustrates how the extra volts array EV_d for a single neuron is indexed and modified over the two phases of the simulation. Note that the actual implementation interleaves the arrays of all neurons and uses a stride of N between indices to facilitate coalesced memory transactions. The neuron's delay is $\delta = 3$ and the current washout table length is $w = 6$ in this example. At time step t, kernel `Phase1` reads the input value from index $ev_{idx} - \delta = 3 - 3 = 0$, computes the new membrane potential and resets the value in EV_d to 0. Index ev_{idx} is incremented before kernel `Phase2` is called. This kernel first queries all neighbours and finds that two of them are firing an excitatory action potential. It then proceeds to add $2\times$ the values from the excitatory washout table $W_{c(e)}$ to the corresponding w elements of the extra volts array, beginning with index $ev_{idx} = 4$. The next simulation step $t + 1$ repeats this procedure, but finds that the neuron is now receiving a single inhibitory input. `Phase2` thus adds $1\times$ the values from the inhibitory washout table $W_{c(i)}$ to the correct values in EV_d.

ration file can be used to specify the exact settings for each time step. Most importantly, the values for each of the λ parameters can be modified to define the beginning and end of periods during which a particular drug effect is being simulated. This mode also gives a more accurate measure of the actual performance of the simulation code itself.

A number of different GPUs are used to compare the execution speed on two generations of CUDA devices. The GTX260 provides 896 MB of device memory and 216 CUDA cores and represents the GT200 series of GPUs. All other devices are based on the Fermi-architecture. The GTX480 and GTX580 provide 1536 MB of memory and have a total of 480 and 512 CUDA cores respectively. The clock speeds and memory bandwidth of the GTX580 are approximately $8 - 10\%$ higher than those of the GTX480. The professional M2070 offers a full 6 GB of device memory, 448 CUDA cores and uses a slightly more moderate

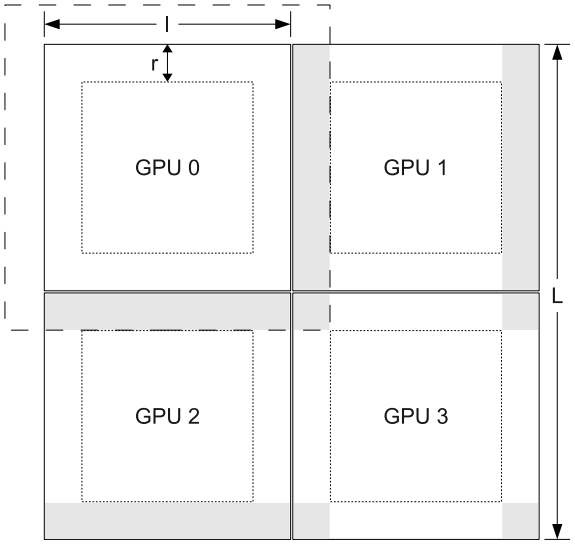

Figure 4: This figure illustrates how the neurons are divided up to be processed in a multi-GPU setup. The maximum neighbour distance r determines the size of the border region that needs to be exchanged between devices processing neighbouring components before `Phase2` can be completed. The neurons located in the core region, however, can be processed independently from all other devices. To exploit this, each GPU concurrently processes its core region and performs the data exchange to obtain the information for its local border on devices that support this feature.

clock speed and bandwidth. All results presented in this section are averaged over 10 independent simulation runs. Error bars representing the standard deviations are smaller than the symbol size.

Figure 5 shows the results for a range of system sizes with fixed in-degree $k = 100$ for all neurons. The maximum neighbour distance $r = 64$. Not every system size can be simulated on all devices due to the different amount of memory available on each GPU. The multi-GPU implementations require a system size of $N \geq 256^2$, as the local dimension length l has to be at least $2r$. Unsurprisingly, the GTX580 is the fastest GPU. It is capable of computing one second of simulated time (1000 time steps) in the cortical model with $N = 262,144$ neurons and over 26 million neural connections in about 1.85 seconds of real time. Although the M2070 is slower than the GTX580, its large DRAM makes it possible to process systems of up to 4.2 million neurons on a single device.

The setup using four GTX480 GPUs, all installed in separate x16 PCIe slots of the same host system, shows how well the multi-GPU implementation scales given a large enough system size. They com-

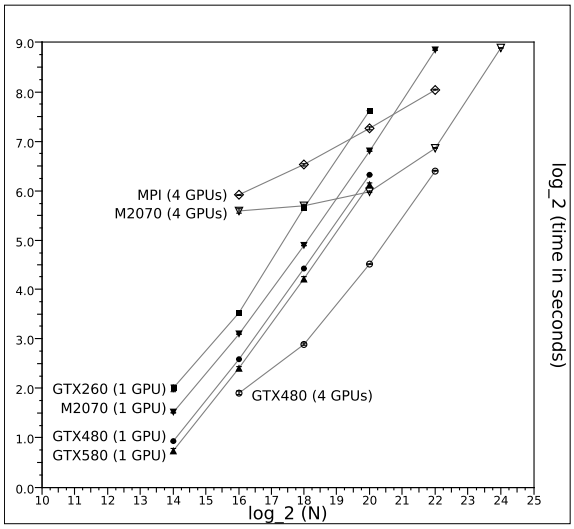

Figure 5: The execution times for $10,000$ simulation steps with system sizes ranging from $N = 2^{14}$ to $N = 2^{24}$. The in-degree is set to $k = 100$, which adds up to a maximum of $\approx 1.68 \times 10^9$ neural connections in the largest system. The the maximum neighbour distance $r = 64$.

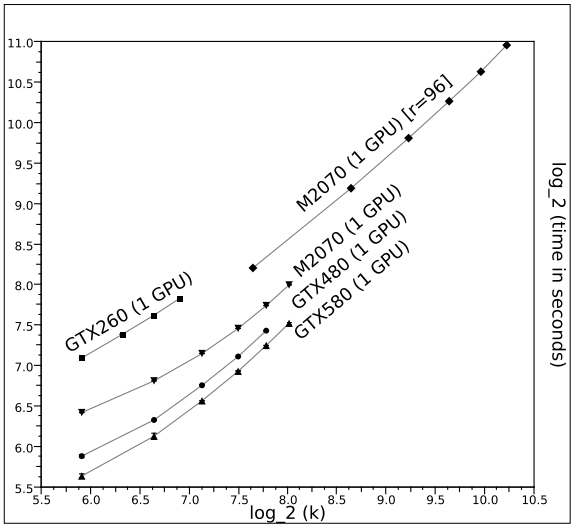

Figure 6: The execution times for $10,000$ simulation steps with in-degrees ranging from $k = 60$ to $k = 1200$. The system size is $N = 2^{20}$ and the maximum neighbour distance is $r = 64$, except for the second result set for the M2070 ($k = 200$ to 1200), which uses $r = 96$.

plete the simulation 3.5 times faster than a single GTX480 when running the largest system supported by both configurations. Both the MPI implementation and the node with four M2070s, which have to share a single x16 PCIe bus, scale very well, but have a much higher communication overhead. They perform best when the system size is large, as the ratio of neurons located in the border regions to those located in the core regions decreases, which enables the devices to overlap more of the data transfer times with computation. The large amount of device memory on the four M2070s makes it possible to process a system of over 16 million neurons with a total of over 1.6 billion neural connections.

Figure 6 shows the execution times for various in-degrees, with a constant system size of $N = 2^{20}$ neurons and a maximum neighbour distance $r = 64$ where not explicitly marked otherwise. No results are given for multi-GPU configurations, as they can only increase the degree at the expense of the sub-system size processed by individual devices. The results offer no surprises, except that the GTX480 runs out of memory when $k = 260$, whereas the GTX580, which is supposed to have the same amount of device memory, completes the simulation successfully. For the large degrees of up to $k = 1200$ that are possible with the M2070, a value of $r = 96$ is used to increase the size of the pool of possible neighbours. This also shows the effect of the neighbour distance on the performance, as a larger value of r means that the texture fetches are less likely to result in a cache hit.

6 Discussion & Conclusions

We have proposed a data-parallel implementation of a neural network model that is based on Izhikevich type neurons. The model is designed with the intention to simulate and analyse neural processes involved in anaesthesia. We are interested in large scale simulations with millions of neurons and many more neural connections to facilitate emergent behaviour that may not be visible at smaller scales. The implementation described in this article merely lays the algorithmic foundation for further studies, as a significant computational effort is still required to find parameter value combinations that work well together and produce realistic neural activity patterns. This is particularly difficult, as many of the system parameters are correlated with each other, which leads to disproportionally strong reactions to relatively small parameter changes.

The implementation of the model also demonstrates how some inherently irregular problems can be optimised for the data-parallel architecture of modern GPUs. This is especially true when it is possible to design the model with this architecture in mind. One such example discussed here is the use of an infor-

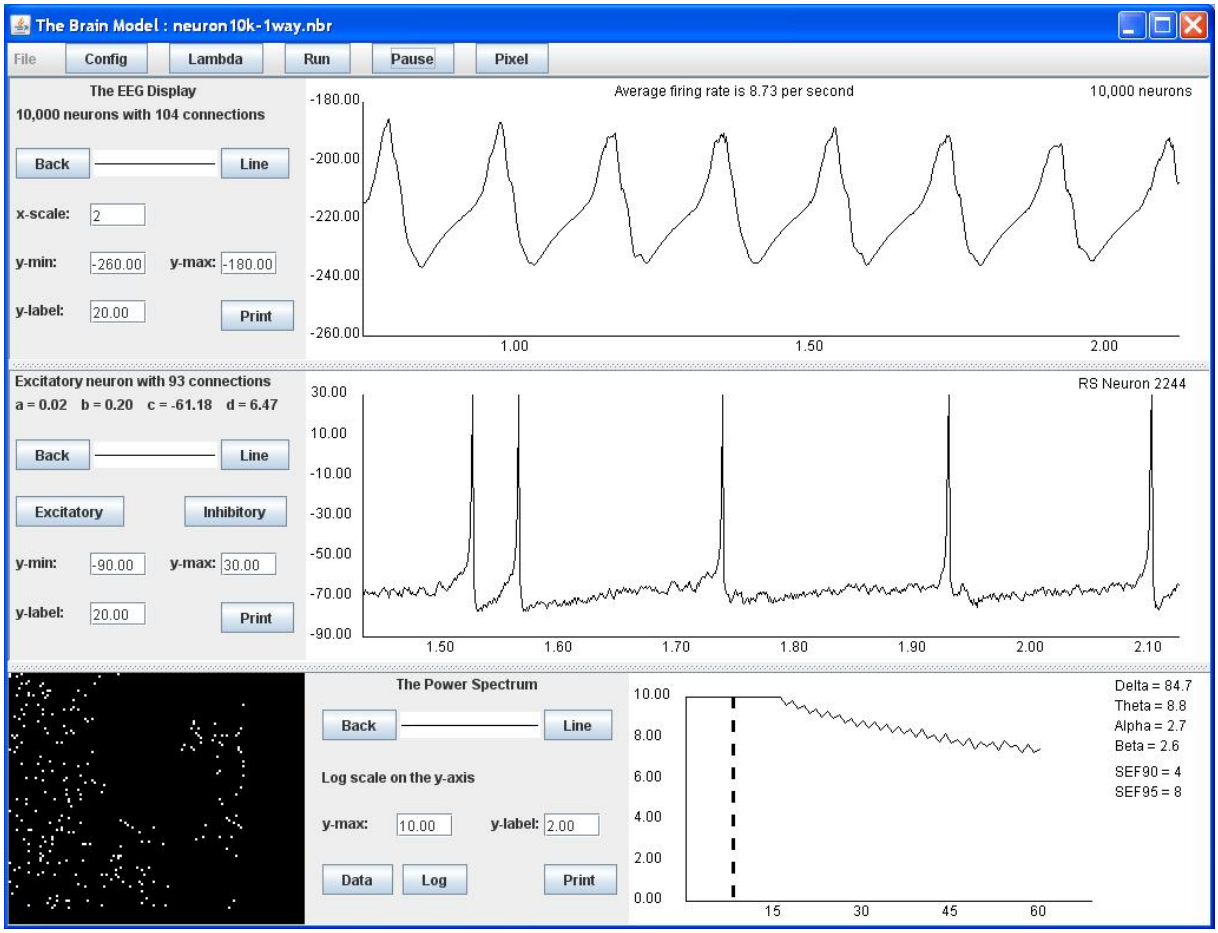

Figure 7: This image shows one of the two GUIs developed to visualise the output generated by the simulation. While this version is written in Java, a C/OpenGL version is used to interface directly with the CUDA implementation of the algorithms. The plots show the pseudo-EEG (top), the membrane potential of a random neuron (middle) and the power spectrum (bottom). The field in the bottom left corner illustrates the action potentials generated by a 200×200 subset of the neurons in real-time. This field can be expanded to show all neurons, which has proven to be useful to visually observe firing patterns that may be of interest to the modeller.

mation pull model for the transfer of action potentials along neural pathways and the decision to use the same in-degree for every neuron. Although this is not biologically realistic, we believe that it does not have a significant effect on the behaviour of the model, as the out-degrees remain randomly distributed. These decisions dramatically improve the simulation performance on GPUs, as they reduce the divergence of threads and increase the utilisation of the memory bandwidth.

As demonstrated, the proposed implementation can be used to simulate systems with more than four million neurons and one hundred times that many neural connections on a single CUDA capable graphics accelerator from the year 2011. A multi-GPU implementation that divides the computational workload and memory requirements between several devices has also been discussed. Although the simulation has only been tested with up to four devices, the implementation can scale to significantly larger compute installations.

The model offers a number of opportunities for future studies, such as the use of different graph structures – in particular small-world network based layouts – and the modelling of the delay terms directly on the neural connections. An extension of the model to include other regions of the mammalian brain would also be of interest.

Acknowledgements

We would like to acknowledge Dr. Douglas Campbell from Auckland Hospital, whose knowledge of the neuronal structure and chemical processes governing the mammalian cortex as well as the different phases observed in patients undergoing anaesthesia were invaluable to this project, and to thank him for suggesting it.

References

[1] D. Lindley, "Computational neuroscientists are learning that the brain is like a computer, except when it isn't." *Communications of the ACM*, vol. 53, pp. 13–15, 2010.

[2] L. F. Abbott, "Theoretical neuroscience rising," *Neuron*, vol. 60, pp. 489–495, 2008.

[3] Y. Fregnac, M. Rudolph, A. P. Davison, and A. Destexhe, *Biological Networks*. World Scientific, 2007, ch. Complexity in Neuronal Networks, pp. 291–340.

[4] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The cat is out of the bag: cortical simulations with 10^9 neurons,10^13 synapses," in *Proc. Conf. on High Performance Computing, Networking, Storage, and Analysis, Portland, OR*, 14-19 Nov 2009.

[5] N. P. Franks, "General anaesthesia: from molecular targets to neuronal pathways of sleep and arousal," *Nature*, vol. 9, pp. 370–386, May 2008.

[6] D. Campbell, "Use of simulated anaesthesia effect of drugs on individual neurons," Private Communication, April 2009.

[7] D. A. Steyn-Ross, M. L. Steyn-Ross, L. C. Wilcocks, and J. W. Sleigh, "Toward a theory of the general-anesthetic-induced phase transition of the cerebral cortex. ii. numerical simulations, spectral entropy, and correlation times," *Phys. Rev. E*, vol. 64, pp. 011 918–1–12, 2001.

[8] J. A. Talavera, S. K. Esser, F. Amzica, S. Hill, and J. F. Antognini, "Modeling the gabaergic action of etomidate on the thalamocortical system," *Anesth. Analg.*, vol. 108, pp. 160–167, 2009.

[9] H. R. Wilson, "Simplified dynamics of human and mammalian neocortical neurons," *J. Theor. Biol.*, vol. 200, pp. 375–388, 1999.

[10] F. Kepes, Ed., *Biological Networks*, ser. Complex Systems and Interdisciplinary Science. World Scientific, 2007, vol. 3, no. ISBN 978-981-270-695-9.

[11] J. Kleinberg, "Small-world phenomena and the dynamics of information," in *Proc. Advances in Neural Information Processing Systems (NPS) 14*, 2001.

[12] D. S. Bassett and E. Bullmore, "Small-world brain networks," *The Neuroscientist*, vol. 12, pp. 512–523, 2006.

[13] O. Sporns and C. J. Honey, "Small worlds inside big brains," *Proc. Nat. Acad. Sci.*, vol. 103, pp. 19 219–19 220, 2006.

[14] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, November 2003.

[15] ——, "Which model to use for cortical spiking neurons?" *IEEE Trans. on Neural networks*, vol. 15, no. 5, pp. 1063–1070, September 2004.

[16] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, pp. 500–544, 1952.

[17] E. M. Izhikevich, "Neural Excitability, Spiking and Bursting," *International Journal of Bifurcation and Chaos*, vol. 10, no. 6, pp. 1171–1266, June 2000.

[18] M. L. Steyn-Ross, D. A. Steyn-Ross, and J. W. Sleigh, "Modelling general anaesthesia as a first-order phase transition in the cortex," *Progress in Biophysics & Molecular Biology*, vol. 85, pp. 369–385, 2004.

[19] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, June 1998.

[20] *NVIDIA CUDA C Programming Guide Version 4.1*, NVIDIA® Corporation, 2011, http://www.nvidia.com/ (last accessed April 2012).

[21] A. Leist, D. Playne, and K. Hawick, "Exploiting Graphical Processing Units for Data-Parallel Scientific Applications," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 2400–2437, December 2009, CSTN-065.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes - The Art of Scientific Computing*, 3rd ed. Cambridge, 2007, iSBN 978-0-521-88407-5.

[23] A. Leist, "Experiences in Data-Parallel Simulation and Analysis of Complex Systems with Irregular Graph Structures," Ph.D. dissertation, Massey University, Auckland, New Zealand, November 2011. [Online]. Available: http://hdl.handle.net/10179/2992