GPU Implementation of Neural-Network Simulations based on Adaptive-Exponential Models

Alexandros Neofytou*§, George Chatzikostantis*§, Ioannis Magkanaris*,
George Smaragdos†, Christos Strydis† Dimitrios Soudris*
*MicroLab, School of Electrical and Computer Engineering, National Technical University of Athens, Greece
{alex.neofytou, iomaganaris}@gmail.com, {georgec, dsoudris}@microlab.ntua.gr
†Neuroscience dept., Erasmus Medical Center, The Netherlands
{g.smaragdos, c.strydis}@erasmusmc.nl
§Corresponding Author

***Abstract*—Detailed brain modeling has been presenting significant challenges to the world of high-performance computing (HPC), posing computational problems that can benefit from modern hardware-acceleration technologies. We explore the capacity of GPUs for simulating large-scale neuronal networks based on the Adaptive Exponential neuron-model, which is widely used in the neuroscientific community. Our GPU-powered simulator acts as a benchmark to evaluate the strengths and limitations of modern GPUs, as well as to explore their scaling properties when simulating large neural networks. This work presents an optimized GPU implementation that outperforms a reference multicore implementation by 50×, whereas utilizing a dual-GPU configuration can deliver a speedup of 90× for networks of $20,000$ fully interconnected AdEx neurons.**

***Index Terms*—computational neuroscience, neuromodelling, GPU, simulation**

I. Introduction

Neuroscientific research has been recently making significant leaps forward in unveiling the intricacies of the human brain [1], [2]. Currently, there is an acute research interest in studying the behaviour of single-neurons, as well as small networks of neurons and eventually brain-sized populations of neurons. Software tools exist aimed at simulating neuronal clusters ranging in size from a single neuron to networks matching a small animal's brain in size [3]. Utilizing modern approaches in high-performance computing to help advance neuroscientific goals constitutes the domain of computational neuroscience. To this end, this work will help neuroscientists gain better intuition of a commonly-used class of neural models when porting to high-performance accelerators, particularly in the case of computationally demanding neural networks.

This paper is organized as follows: In Section II, a concise presentation of related work in the field is given. Section III describes the simulator presented in this paper. Section IV details the GPU porting of the simulator. Section V contains the experimental setup and evaluation of the simulator. Finally, the paper concludes in Section VI, where a brief summation of the most important points covered in this work is given.

II. Related Work

Significant works concerning high-performance computational neuromodelling have been presented in relevant literature. The Computer Science and Cognitive Science departments of the University of California Irvine have implemented a biologically realistic Spiking Neural Network (SNN) simulator that runs on a single GPU [4]. The simulator allows user-defined configuration of the exact model architecture in C++. An NVIDIA GTX-280 simulates $100,000$ neurons with $50M$ synaptic connections, firing at an average rate of 7 Hz, reaching up to 26× speedup over the CPU version.

The Imperial College of London developed NeMo, a platform designed to operate on "highly parallel commodity hardware in the form of graphics processing units (GPUs)" [5]. The implemented GPU kernel delivers up to $400M$ spikes per second, which corresponds to a real-time simulation of around $40,000$ neurons, each connected through $1,000$ synapses, with a mean firing rate of $10Hz$.

The BRIAN Computation Laboratory of the Department of Computer Science and Engineering of the University of Nevada presented an updated version of their NeoCortical Simulator, an open-source CPU/GPU simulation environment for large-scale biological networks [6]. The simulator is currently able to simulate $1M$ neurons and $100M$ synapses in quasi real time using eight nodes, each having 2 video cards.

Similarly to the aforementioned works, we opt for employing GPUs in our simulator design as the computational fabric of choice. This work is the first, to our knowledge, to extensively study the scaling differences between two different AdEx network configurations; an externally stimulated network as well as a self-stimulated one. Furthermore, the simulator is part of an effort to develop an acceleration platform for computationally challenging neuroscientific simulations [7].

III. Application Design

We focus on a class of models that capture the timings and propagation of signals within a neuronal network, the Integrate-and-Fire models. A widely adopted and flexible model is the Adaptive Exponential model, also known as "AdEx". The model consists of two equations, one describing dynamics of the membrane potential and one describing adaptation. Introduced by Brette and Gerstner in 2005 [8], the model has become popular in modern research due to its

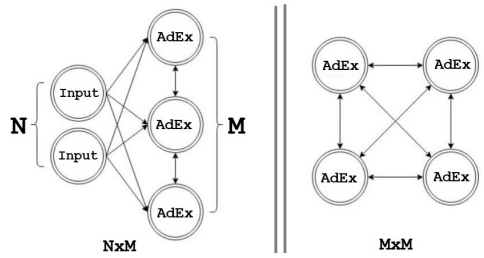

Fig. 1. Two types of network setups examined by the simulator. A layer of **N** spiking neurons feeds a layer of **M** AdEx neurons (**N**×**M** case); or **M** AdEx neurons form interconnections and interact with one another (**M**×**M** case).

ability to reproduce qualitatively several electrophysiological classes described "in vitro". Synapses in the neuronal network are modelled after Spike-Timing Dependent Plasticity (STDP), a biological process that adjusts the strength of neuronal connections in the brain.

A. Implementation in x86

The simulator has been initially designed in C for x86 CPUs. An implementation of the described model [9], detailing a small-scale experiment of 100 spiking neurons providing stimulus to a single AdEx neuron, acted as our baseline. We chose this implementation due to its scalable nature and well-maintained, self-contained code repository, which eased its adaptation into a more flexible and configurable simulator.

Our simulator allows for any number (referred to as number **N**) of input spiking neurons to feed stimulus to any number (referred to as number **M**) of AdEx neurons. Furthermore, our simulator can also support self-feeding networks of AdEx neurons (refer to Figure 1). This category of networks can display interactions between the AdEx neurons not present in **N**×**M** networks and are more difficult to scale due to the significantly higher amounts of memory transactions imposed by synaptic activity between the AdEx neurons.

Input neurons are represented by **N** integers. An array of integers of length equal to the amount on neurons in the network (i.e. either **N**+**M** or **M**) is used to represent spike occurrences on the corresponding neurons, named SpikeArray. SpikeArray also holds all necessary AdEx neuron data. AdEx Neurons are represented as a struct which contains the double-precision variables for voltage threshold, membrane potential, input current and the models adaptation variable. Furthermore, the struct holds an integer signifying whether a neuron has fired a spike (*SpikeSetting* in Figure 2).

The STDP synaptic model necessitates a struct that records all synapse-related data. Both neuron and synapse data are stored as two arrays of structs (AoS). This layout boosts spatial locality compared to a single struct of arrays (SoA).

In each step, the simulator processes three functions. Function *SolveNeurons* is responsible for the update of AdEx neuron values. The function also checks neuronal membrane potential for spike generation purposes. In case of spike firing, function *UpdatePreSynapses* updates synapses connecting the firing neuron to other neurons and calculates the propagating current to post-synaptic neurons. Finally, function *UpdatePostSynapses* calculates synaptic variables when postsynaptic AdEx neurons fire. CPU performance of these functions is boosted by introducing OpenMP pragmas.

IV. Application Acceleration

A. GPU Porting

The simulator presented in Section III was accelerated in a GPU using the CUDA library. The three main functions mentioned in Section III-A were re-implemented as GPU kernels. Computations of the model use data specific to each neuron and synapse, with few data dependencies between the respective data structs. As such, the approach chosen for GPU was assigning neurons and synapses to CUDA threads in a singular fashion. Depending on the GPU kernel, a CUDA thread will handle the computations concerning a single neuron, or a single synapse. This approach offers good usage of the GPU's computational resources while keeping synchronization overheads low.

In order to process data in a GPU kernel, data must be passed to the device. Array allocation and copying requires array elements to be in adjacent memory addresses, as there is no CUDA operation to handle discontinuity between elements. Furthermore, row alignment in 2D CUDA arrays is critical to memory transaction speed; *cudaMallocPitch* pads rows with extra bytes if necessary. Transforming the 2D arrays to a 1D representation is another option in order to avoid using *cudaMallocPitch*.

Simulation data is transferred to and from the device at the beginning and end of the simulation; processed data is stored locally in the GPU throughout the simulation. However, function *UpdatePostSynapses*, mentioned in Section III-A, requires the calculation of an average value between multiple synaptic variables. In a GPU kernel, this operation is difficult to parallelize; an initial implementation sends relevant data back to the CPU host for processing, followed by the result propagated back to the device in order to complete kernel calculations. This splits the *UpdatePostSynapses* into two GPU kernels, as it can be seen in Figure 2. This implementation significantly increases memory transactions; an alternative GPU kernel for calculation of the value in the GPU, using a library called "Thrust", allows the bypassing of memory transactions. The library allows GPU "reduction". After spawning $N/2$ GPU threads, the sum of two out of N elements is calculated by each thread. This step is then repeated by spawning $N/4$ threads and operating on the $N/2$ results produced by the first step. Iteratively, the reduction summation is achieved in $\log_2 N$ steps.

Moreover, in the interest of data locality, data arrangement presented as an Array-of-Structs (AoS) in Section III-A was transformed to a Struct-of-Arrays (SoA). In an SoA arrangement, all data concerning one model parameter are stored consecutively, for all neurons or synapses. As such, when GPU threads process a particular model parameter, they access adjacent memory addresses.

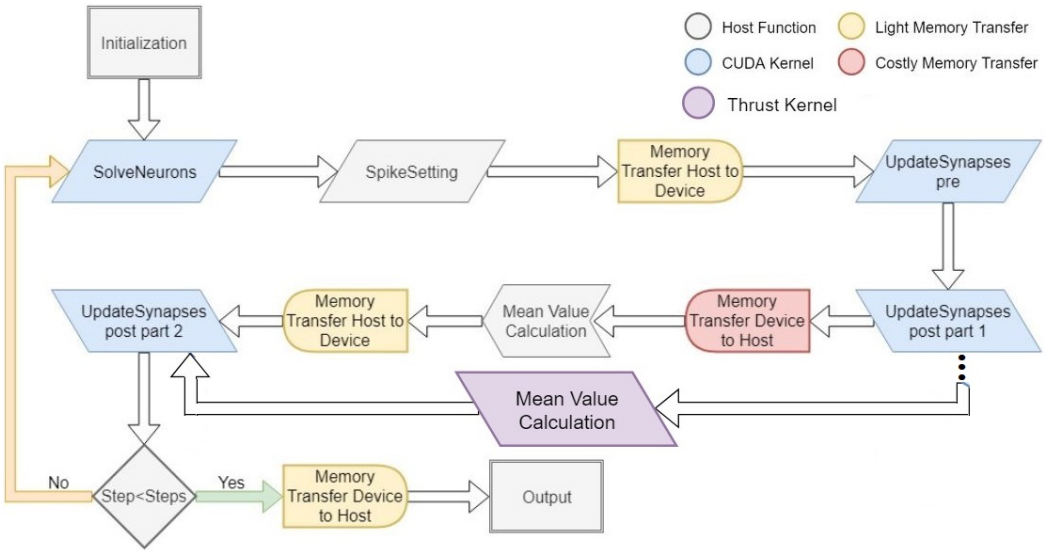

Fig. 2. Flow diagram for the GPU implementation of the simulator. A reduction summation kernel using the "Thrust" GPU library allows bypassing of costly host-device memory transactions.

Furthermore, the synaptic SoA, which requires the most significant amount of accessing and processing, is allocated on the on-chip GPU memory space, shared by all threads in a GPU block. In order to reduce time spent in memory accesses, threads first store data in shared memory and then store results back to global memory. Despite the overhead of using the shared memory as an intermediate between the GPU threads and the global memory, shared memory "banks" offer reduced memory access conflicts that improve execution time.

B. Large-Scale Network Support on the GPU

The amount of AdEx neurons (and STDP synapses) that can be processed are limited by the device's memory, since data is stored locally in the GPU, with an upper limit of $12,500$ interconnected AdEx neurons in an $\mathbf{M}\times\mathbf{M}$ network, whereas $\mathbf{N}\times\mathbf{M}$ networks require significantly less memory. In an effort to surpass this mark, two different approaches were explored for $\mathbf{M}\times\mathbf{M}$ networks, depending on the available hardware.

In the case of a single GPU, larger networks need to be divided in smaller sub-networks whose data can "fit" in the GPU memory. In each simulation step, the sub-networks are processed sequentially. Data dependencies between the sub-networks must be minimal or, in the best case, absent. The few dependencies between data in different sub-networks must be resolved and calculated in the CPU. The GPU kernels are called in a loop, where each iteration processes a sub-network. The sub-network's data is passed to the device and after kernel execution, data is transferred back to the host and device memory is overwritten with the next iteration's data. This approach allows a single GPU to handle larger network sizes but there is a vast performance cost caused by memory transfers of the sub-network data between device and host in every timestep.

When more than a single GPU are available in a single socket, the amount of on-GPU memory that can be utilized is increased respectively. The AdEx and STDP model permit independent concurrent execution of different chunks of the data on separate devices. The neuronal network can be divided in two halves, each allocated to a different device. CUDA function *cudaMemcpyAsync* allows asynchronous memory allocation for parallel memory transfers.

V. Performance Exploration

TABLE I
Range of Parameters Explored

	Parameter Range	
Parameter Name	**M**×**M** Networks	**N**×**M** Networks
M (AdEx neurons)	100 to $20,000$	$5,000$ to $50,000$
N (input neurons)	N/A	200 to $1,000$
Connectivity (%)	25% to 100%	25% to 100%

In this Section, we will present an evaluation of the performance offered by the designed simulator. As mentioned in Section III, two types of experiments have been tested. The first type of network ($\mathbf{N}\times\mathbf{M}$) consists of a layer of $\mathbf{N}$ spiking neurons feeding a layer of $\mathbf{M}$ AdEx neurons. The second type ($\mathbf{M}\times\mathbf{M}$) is a network of $\mathbf{M}$ AdEx neurons with interconnections between them. Details on the nature of the experiments and the exact hardware used to measure performance will be detailed here. The results of the experimentation will then, be presented and elaborated on.

A. Experimental Setup

Performance measurements were collected by running simulation experiments on a dual GPU node. The GPU node

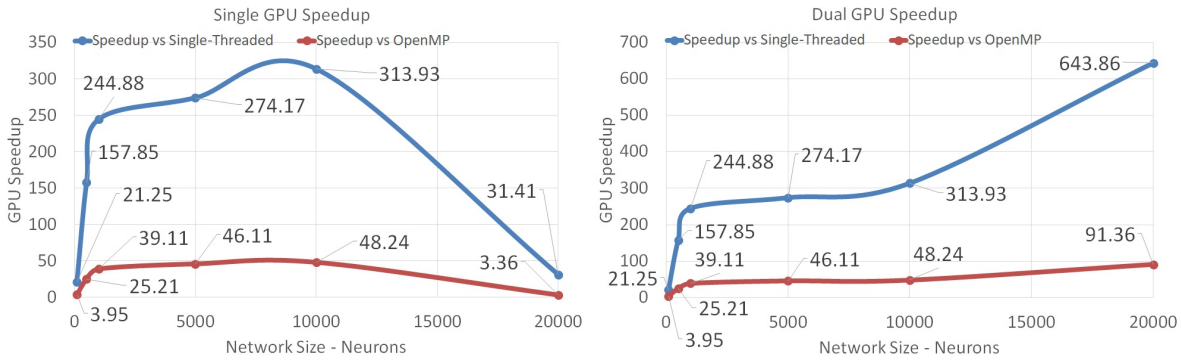

Fig. 3. **M**×**M** network size exploration.

TABLE II
SPEEDUP OF **N**×**M** NETWORKS BY SIZE (SINGLE GPU)

	Network Size Parameters, **N** input neurons, **M** AdEx neurons								
	N=200			N=500			N=1000		
	M=5k	M=10k	M=50k	M=5k	M=10k	M=50k	M=5k	M=10k	M=50k
Speedup vs Single-Threaded	325.89	339.48	351.2	346.98	346.61	354.34	382.5	351.65	388.67
Speedup vs OpenMP	54.95	53.87	57.57	58.31	55.91	55.36	65.94	57.64	61.69

contains 2 CPUs of the Haswell - Intel(R) Xeon(R) E5-2660v3 type, each equipped with 10 cores and 64 GBs of memory. The GPU node features two NVIDIA Tesla K40 accelerators with 12 GB accelerator memory each. The CUDA version used for the GPU implementation was **7.5** with CUDA capability **3.5**. The CPU implementation was also provided as a baseline to evaluate the GPU implementation against. The CPU node used contained dual Ivy Bridge Intel Xeon E5-2680v2 processors, with 2.8 GHz operating frequency, 10 cores per processor and 64 GBs of RAM.

In parameter Table I, **M** has been used for the number of AdEx neurons modeled in the network, whereas **N** stands for the number of spiking neurons feeding the AdEx neurons in two-layer **N**×**M** networks. **M**×**M** networks have been significantly more computation-heavy than **N**×**M** networks and as such, their M^2 size has been a limiting factor. Connectivity is expressed as a percentage out of all possible network connections.

B. Experimental Results

The execution times achieved by the GPU will be compared against the CPU implementation and expressed as speedup over the CPU respective data point. For clarity purposes, in the case of **M**×**M** networks, the results will be presented in graph format, whereas **N**×**M** network performance will be displayed in table format. For the simulation runs in this section, spiking frequency was set to 1 kHz, meaning spikes appear in every timestep, as a measurement of the most demanding case computationally. For the same purpose, connectivity in all cases is also set to 100%.

The *SolveNeurons* GPU kernel occupies a minimal percentage of the overall simulation time. This results in synaptic operations being the ones dictating simulator performance. In **M**×**M** experiments, a heavier computational load results in greater speedup, as long as there are enough CUDA threads for the GPU thread grid to cover the entire synaptic grid of the simulation. Maximum GPU acceleration on the **M**×**M** simulation runs score more than 300× speedup compared to a single-threaded CPU simulation and close to a 50× speedup against an OpenMP-based parallel CPU implementation.

On the other hand, for **M**×**M** networks of 20,000 neurons, network partitioning is necessary in order to meet GPU memory requirements. Due to the penalty described in Section IV-B, a single GPU only achieves a 31× speedup over a single-threaded CPU version of the simulator and improves on the OpenMP parallel CPU version by less than 4×. By using a second GPU in Figure 3, more than 90× acceleration over the OpenMP CPU implementation is achieved.

In Table II **N**×**M** simulation results, speedup rates follow a similar trend, with higher maximum speedup rates achieved for sufficiently large networks. Due to connections existing solely between input neurons and AdEx neurons, only the relevant variables of synapses that connect such neuron pairs are updated, whereas for **M**×**M** networks, all AdEx neurons update their status in every simulation step. Therefore, in the **N**×**M** case, updating synapses based on postsynaptic STDP expressions is omitted. As the postsynaptic GPU kernels present the heaviest workload, **N**×**M** simulations are lighter for the GPU, slightly increasing performance. Maximum GPU acceleration on the **N**×**M** cases surpasses 380× speedup against a single-threaded CPU and reaches 60× acceleration against the parallel CPU implementation.

In Figure 4, we alter network connectivity from fixed 100%

TABLE III
SPEEDUP OF **N**×**M** NETWORKS BY NETWORK DENSITY (SINGLE GPU)

	Network Size Parameters, **N** input neurons, **M** AdEx neurons								
	N=200			N=500			N=1000		
	M=5k	M=10k	M=50k	M=5k	M=10k	M=50k	M=5k	M=10k	M=50k
100% Connectivity	325.89	339.48	351.2	346.98	346.61	354.34	382.5	351.65	388.67
25% Connectivity	183.32	190.48	198.61	195.14	194.92	236	194.93	202.5	230.35

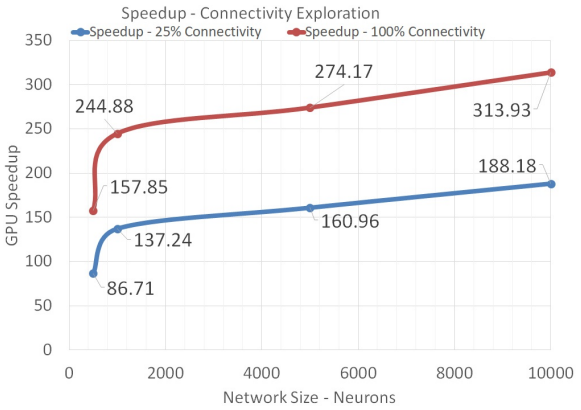

Fig. 4. Single-GPU acceleration for different **M**×**M** network densities.

values to only 25%. This change in **M**×**M** networks means that each AdEx neuron is connected to only a quarter of the network, whereas in **N**×**M** networks, each input neuron feeds spikes to a quarter of the AdEx neurons. Figure 4 denotes the decline in acceleration when network connectivity is significantly reduced. Inactivity in the network translates to less GPU threads being active in any simulation step, since there is a one-to-one mapping of GPU threads to network synapses. On the other hand, the CPU can take advantage of inactivity by processing a smaller total number of synapses, resulting in a reduced performance gap.

VI. CONCLUSION

In this work, neuronal networks based on adaptive-exponential models with STDP-modelled synapses have been ported to GPUs for acceleration. Two network configurations have been used, a bilayer network of input and AdEx neurons and a self-feeding AdEx network. After implementing the model originally found in a neuromodelling library (modelDB) in C for CPU, we documented the process of optimizing its performance in GPU.

The implementation was evaluated against its single-threaded CPU implementation, as well as an OpenMP-parallel version. We identified a threshold in the performance of a single GPU at $12,000$ neurons due to the limitations of the available device memory. In an effort to tackle this obstacle, increasing hardware to a dual-GPU configuration can accelerate the simulation of a network of $20,000$ AdEx neurons, fully interconnected with one another, by a factor greater than $90\times$, when compared to the OpenMP-parallel CPU simulation.

Furthermore, the bilayer category of networks explored in this paper proved to be a significantly lighter workload, allowing larger network sizes without the use of dual GPUs, since the amount of synaptic data to be processed was smaller. Performance patterns were similar in both categories, with bilayer networks of input and AdEx neurons offering slightly larger speedups than the self-feeding AdEx networks. In both categories, reducing connectivity density in the network benefitted the CPU more than the GPU, thus reducing the achievable GPU speedup rate.

ACKNOWLEDGMENT

This research is supported by European Commission H2020 project EXA2PRO for FETHPC-02-2017 “Transition to Exascale Computing” (Grant agreement ID: 801015). The work was also supported by computational time granted from the Greek Research & Technology Network (GRNET) in the National HPC facility “Advanced Research Information System - ARIS”.

REFERENCES

[1] H. Markram, “The human brain project,” *Scientific American*, vol. 306, no. 6, pp. 50–55, 2012.
[2] T. R. Insel, S. C. Landis, and F. S. Collins, “The nih brain initiative,” *Science*, vol. 340, no. 6133, pp. 687–688, 2013.
[3] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, “The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses,” in *Conference on High Performance Computing Networking, Storage and Analysis*, 2009.
[4] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, “A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors,” *Neural networks*, vol. 22, no. 5-6, pp. 791–800, 2009.
[5] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, and W. Luk, “Nemo: a platform for neural modelling of spiking neurons using gpus,” in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2009, pp. 137–144.
[6] R. V. Hoang, D. Tanna, L. C. Jayet Bray, S. M. Dascalu, and F. C. Harris Jr, “A novel cpu/gpu simulation environment for large-scale biologically realistic neural modeling,” *Frontiers in neuroinformatics*, vol. 7, p. 19, 2013.
[7] G. Smaragdos, G. Chatzikonstantis, R. Kukreja, H. Sidiropoulos, D. Rodopoulos, I. Sourdis, Z. Al-Ars, C. Kachris, D. Soudris, C. I. De Zeeuw *et al.*, “Brainframe: a node-level heterogeneous accelerator platform for neuron simulations,” *Journal of neural engineering*, vol. 14, no. 6, p. 066008, 2017.
[8] R. Brette and W. Gerstner, “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity,” *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.
[9] R. P. Costa, R. C. Froemke, P. J. Sjöström, and M. C. van Rossum, “Unified pre-and postsynaptic long-term plasticity enables reliable and flexible learning,” *Elife*, vol. 4, p. e09457, 2015.