Exploring Multi-level Parallelism for Large-Scale Spiking Neural Networks

Vivek K. Pallipuram, Melissa C. Smith, Nimisha Raut, and Xiaoyu Ren

Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA

Abstract*— *Several biologically inspired applications have been motivated by Spiking Neural Networks (SNNs) such as the Hodgkin-Huxley (HH) and Izhikevich models, owing to their high biological accuracy. The inherent massively parallel nature of the SNN simulations makes them a good fit for heterogeneous computing resources such as the General Purpose Graphical Processing Unit (GPGPU) clusters. In this research, we explore multi-level parallelism offered by heterogeneous computing resources for large-scale SNN simulations. These simulations were performed using a two-level character recognition network based on the aforementioned SNN models on NCSA's Forge GPGPU cluster. Our multi-node GPGPU implementation distributes the computations to either CPU or GPGPU based on task classification and utilizes all the available multi-level parallelism offered to ensure maximum heterogeneous resource utilization. Our multi-node GPGPU implementation scales up to 200 million neurons for the two-level network and achieves a speed-up of 355x over an equivalent MPI-only implementation.

Keywords **–** GPGPU cluster, Neural Networks, Performance, Scalability, Multi-level Parallelism

1 Introduction

Spiking Neural Networks (SNNs) are very popular in the neuroscience community for modeling the mammalian brain to understand its functional and operational principles. The ability of spiking neurons to reproduce most of the neuronal properties with high accuracy makes them amenable for brain related studies [1]. Biologically inspired SNNs are now popular in other fields such as pattern recognition [2], artificial intelligence [3], and smart control of power grids [4]. Among several SNN models, the Izhikevich model [5] and the Hodgkin-Huxley (HH) model [6] are considered to be highly biologically accurate [1]. The Izhikevich model is the most recent and computation efficient model, whereas the HH model is the oldest and highly computation intensive model.

With the advent of General Purpose Graphical Processing Units (GPGPUs) in the field of high performance computing, the current trend is to extract concurrency from heterogeneous computing resources such as GPGPU clusters. The current state-of-the-art heterogeneous systems are composed of several thousands of compute nodes, where each node is equipped with multiple CPU cores in conjunction with one or more GPGPU accelerators. Since GPGPUs have been established in the literature as viable architecture choice for SNN simulations [7, 8], GPGPU clusters are lucrative options for large-scale SNN simulations and related studies. Although these heterogeneous systems can provide substantial performance for massively parallel simulations, much of their computing resources are often under-utilized due to poor tuning strategies. To achieve optimal utilization of the heterogeneous resources, it is imperative to perform efficient load-balancing between the CPU cores and GPGPU devices, which requires exposing all available parallelism in the application and effective memory and bandwidth utilization.

With the above as motivation, in this research, we investigate multi-level parallelism for large-scale massively parallel SNN simulations. These simulations were performed using a two-level character recognition network based on [2]. The two-level network capable of recognizing 48 alpha-numeric characters was developed using the Izhikevich and HH models. The two-level network was mapped on the *National Center for Super-Computing Applications* (NCSA) 32-node Forge GPGPU cluster and scaled to over 200 million neurons. The focal contributions of this work are:

1) Exploration of multi-level parallelism for spiking neural networks in the form of GPGPU+MPI+OpenMP implementations.
2) Demonstration of optimal load-balancing between the CPU cores and GPGPUs.
3) Scaling of the two-level network beyond the single-node capability.
4) Scalability and performance analysis on the Forge cluster.

Our heterogeneous parallel implementations were able to achieve significant application speed-ups, as high as 355x for the computation intensive HH model and 4x for the computation efficient Izhikevich model over equivalent MPI-only implementations on the Forge cluster. The performance of the two SNN models was found to depend on the computation-to-communication requirements of the SNN models.

The rest of the paper is organized as follows. Section 2 provides a brief background on the two SNN models, the two-level network, and an overview of the GPGPU architecture. Section 3 discusses related work. Section 4 details the experimental set-up and network mapping. Section 5 presents the results and analysis, and the paper is concluded in Section 6 with conclusions and suggestions for future work.

2 Background

2.1 Spiking neural networks

Over the last 50 years, several models [9] have been proposed that capture the spiking mechanism within a neuron. In this paper, we examine two of the most biologically accurate spiking neuron models for implementation on the GPGPU cluster. In what follows, we give a brief chronological overview of these two popular SNN models, namely the Hodgkin-Huxley and Izhikevich models.

The Hodgkin–Huxley (HH) model is considered to be the most accurate and the most important model in the neuroscience community till date. As mentioned in [1], the model involves 4 equations and 10 parameters describing various neuron current activation and deactivation. The model takes 120 flops per 0.1 ms time-step and hence 1200 flops/1 ms for the update. In our research, we have used 0.01 ms time-step for the neuron update.

In [5], Izhikevich developed a simple and very computation efficient spiking neuron model that has similar accuracy as the HH model. Izhikevich successfully reduced the complex HH model equations to a 2-D system of ordinary equations. Izhikevich's model requires only 13 flops per neuron update and still sufficiently reproduces a majority of neuronal properties. The equations are found in [5]. In our research, we have used a 1 ms time-step (13 flops/1 ms update) for neuronal dynamics update for the Izhikevich model.

The time-steps used in our research for the models discussed are in the valid range of time-steps that are deemed sufficient for reproducing biologically relevant neuron dynamics [1].

Table I provides the Flops/Byte ratio for the two models, which is pertinent to the performance analysis of the two models. The Flops/Byte ratio is an algorithm specific value and is defined as the ratio of the number of floating-point operations required for a complete neuron update (level-1 and level-2) to the overall bytes requested (all model parameters, firing vector and block firing vector) for all of the neuron updates [8].

Table I. Flops/Byte ratio for the two models

Model	Flops/neuron required for the complete neuron update	Flops/Byte ratio
HH	246	9.84
Izhikevich	13	0.9997

2.2 The two-level network

The two-level character recognition network used in this research is based on [2] and the network used to test the models is shown in Figure 1. The task of the network is to detect images from a training data set of 48 images. The level-1 neurons act as an input collection layer and the level-2 neurons act as output collection layer. Each neuron in level-1 corresponds to a pixel in the input image; hence the number of neurons in the input level is equal to the total number of pixels in the test image. The number of neurons in the output layer, level-2, is equal to the number of images in the database. When an input image is presented to level-1, each neuron evaluates its membrane potential based on the pixel level presented and the neuron model chosen. This process is referred to as the *evaluation of neuron dynamics*. If the pixel is "on," a constant current is supplied to the neuron for membrane potential evaluation. The input current equation for a level-2 neuron is:

$$I_j = \Sigma w(i,j) f(i) \qquad (1)$$

In (1), I_j is the net input current to the neuron j in level-2, $w(i,j)$ is the weight of the synapse connecting neuron i in level-1 and the neuron j in level-2. A neuron in any level is said to have "fired" if its membrane potential crosses the threshold value for the selected neuron model. The research presented in this paper accelerates the recognition phase of the network by implementing all of the level-1 neurons on the GPGPU devices, while the level-2 neurons (input current accumulation and dynamics) are implemented on the host processors as will be discussed later in Section 4.

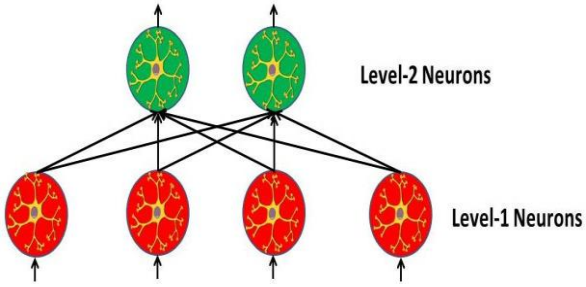

Figure 1. Two-level character recognition network

2.3 The GPGPU architecture

The Compute Unified Device Architecture (CUDA) programming framework views the GPGPU architecture as an array of streaming multi-processors (SMPs). Each multi-processor contains a set of scalar processors (referred to as CUDA cores), a double-precision (DP) unit, shared memory for thread cooperation, and texture addressing and texture fetch units. While a single thread is executed on a CUDA core, a group of threads called a *thread block*, is executed on the SMPs. Threads in a thread block can synchronize with each other using shared memory. The Fermi architecture has brought a lot of innovation versus previous Nvidia architectures: 512 CUDA cores organized as 16 SMPs with 32 cores each sharing a L2 cache. The SMPs have two sets of 16 CUDA cores, 4 special function units for transcendental functions, 16 load/store units, a hefty register file, and a configurable 64 KB L1 cache/shared memory. The GPGPU device has the capability of supporting 6 GB of GDDR5 DRAM memory. The Fermi-based Tesla M2070

used in this research can theoretically offer 1.03 tera-flops of single-precision floating-point performance and 515 giga-flops of double-precision floating-point performance. More information on the GPGPU architecture and CUDA framework can be found in [10] and [11], respectively.

3 Related work

Several research activities have been motivated by the idea of modeling the neocortex. In [12], the authors have studied the mammalian brain neocortex in detail and were successful in simulating a rat-size cortex in 42% of real-time and a cat-size cortex in 23% of real-time on a 442 node Dell Xeon cluster. Their neuron model is in the integrate-and-fire (I&F) category, which according to Izhikevich, is insufficient for accurately reproducing neuronal properties [1]. In [13], the authors successfully used Izhikevich's model to simulate a cat-size cortical model with 10^9 neurons and 10^{13} synapses using a BlueGene/P machine with 147,456 processors and 144 TB of main memory. The authors claim that their simulation scale is roughly 1–2 orders of magnitude smaller than the human cortex and 2–3 orders of magnitude slower than real-time.

Alternative computing architectures such as GPGPUs are now being investigated for biologically realistic simulations. In [7], the authors implemented Izhikevich's random network on Nvidia's GTX-280 with 1 GB memory and achieved a speed-up of 26x over an equivalent software implementation for a 100K neuron network simulation. Their work discusses mapping strategies on the GPGPU to efficiently utilize the memory bandwidth and parallelism.

In [14], the authors investigated GPGPU cluster based implementations of the HH and Izhikevich models using a two-level network based on [2]. They reported GPGPU speed-ups of 24.6x and 177x for the Izhikevich and HH models, respectively over a 2.4 GHz dual-core AMD opteron processor. Their 16 GPGPU-based MPI implementation on a 32-node Tesla S1070 NCSA cluster was successful in scaling the network up to 150 million neurons and achieved 17910 ms runtime for the HH model.

Although our two-level network and experiments are similar to [14], our work is different in the following ways. In [14], the authors perform the level-1 neuron dynamics, level-2 current accumulation, and level-2 neuron dynamics calculations on the GPGPU device. In [14], the authors also claim that the image detection operation (checking if any of the level-2 neurons fired) is inherently serial, therefore it can be performed on the CPU host. As will be highlighted in Section 4, the implementation in [14] is not efficient since the level-2 dynamics calculation is fairly small (only 48 neurons in level-2), performing this operation on the GPGPU device is not warranted and renders the CPU resources under-utilized. Additionally, the evaluation of level-2 neuron currents on the GPGPU device requires the transfer of the large weight matrix to the GPGPU device memory, which is wasteful of the host-device bandwidth and device memory as explained in [15]. Our GPGPU-based MPI implementation achieves optimal load-balancing between the CPU cores and GPGPU devices by implementing the level-1 neuron dynamics on the GPGPU devices and level-2 neuron calculations (current accumulation and dynamics) on the host processors as will be discussed in Section 4. In addition, our work exploits parallelism across multiple levels of the heterogeneous architecture in the form of a complete GPGPU+MPI+OpenMP based implementation.

4 Experimental setup and network mapping

4.1 Experimental setup

NCSA's Forge GPGPU cluster was used in this research for the large-scale SNN simulations. The 153 tera-flop cluster is composed of 36 Dell PowerEdge C6145 servers; each server is connected to six Fermi-based Tesla M2070 GPGPUs via three PCI-e Gen2 X16 slots. Each server is equipped with two 2.4 GHz AMD Opteron Magny-Cours 6136 processors, eight cores each. The network interconnect is comprised of InfiniBand QDR. Our implementations were developed using CUDA 4.0 and OpenMPI 1.4.3 on Red Hat Enterprise Linux 6.

4.2 SNN network mapping

In this sub-section, we first provide the details of network mapping for the single-GPGPU implementation that is subsequently extended to a GPGPU-based MPI implementation.

As discussed in section 2.2, level-1 is the most computation intensive layer of the network since the number of neurons is equal to the number of pixels in the input image; therefore these operations are performed on the GPGPU device. Each GPGPU thread evaluates the dynamics of a single level-1 neuron. Therefore, the number of GPGPU threads created is equal to the number of level-1 neurons. The GPGPU device then provides the host processor with the level-1 neuron firing information, the *global firing vector*, which is used by the host processor to obtain the level-2 neuron currents and dynamics. The level-2 computations (current accumulation and dynamics) are implemented on the host processor since the level-2 neuron computations constitute less than 5% of the total computation overhead and, implementing the level-2 dynamics on the GPGPU would require the transfer of the weight matrix to the GPGPU device memory. Hence any computational improvement obtained by implementing level-2 neuron dynamics will be insufficient to amortize the communication overhead involved in transferring the large weight matrix to the GPGPU device. The single-GPGPU implementation was optimized with memory level, instruction level, and execution configuration level optimizations as mentioned in [15].

The host-device bandwidth was further optimized using a *block firing vector* concept introduced in [8]. The block firing vector is implemented in the device shared memory to avoid transferring the global firing vector in each algorithmic time-step. The block firing vector is similar to the global firing vector but instead acts as a collection of flags for thread blocks. Since the threads are collected in thread blocks of size: *blocksize*, the block firing vector is *blocksize* magnitude smaller than the global firing vector, and hence can be transferred from the device to host in each time-step with minimal overhead. If at any time-step the block firing vector contains information of a firing event, only then will the entire global firing vector be transferred from the device to host and then read by the host. Figure 2 illustrates the block firing vector concept.

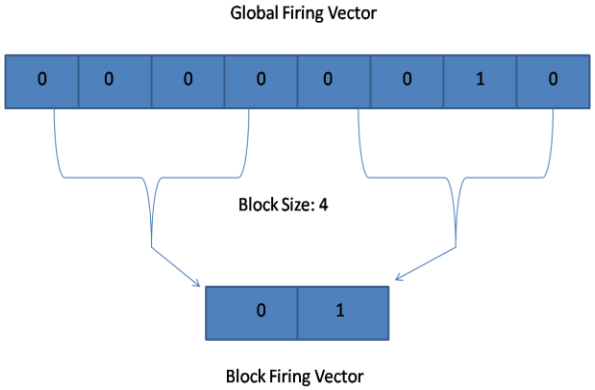

Figure 2. The concept of block firing vector

The single-GPGPU implementation is then extended to a GPGPU-based MPI implementation. The MPI ranks were assigned in node-packing fashion, meaning the ranks are packed into nodes. The nodes were configured with a maximum of six MPI processes per node. This configuration allows for 1:1 CPU core-to-GPGPU ratio at each node and potentially reduces long distance inter-node communication. The GPGPU devices were allotted to the CPU cores using modulo rule where an MPI process with rank *n* is coupled with the GPGPU device number, *n modulo 6* [16]. Future work will investigate the impact of other CPU core-to-GPGPU ratios on application performance.

The GPGPU-based MPI orchestration is as follows. The MPI rank 0 acts as the master process that scatters the level-1 neuron inputs to all other processes. The level-1 neuron parameters are initialized to the SNN model specific constant values at each MPI process, and hence require no MPI communication. Each CPU-GPGPU pair works as an independent unit where the GPGPU device evaluates the partial level-1 neuron dynamics and the host processor evaluates the partial level-2 currents using the firing vector obtained from its designated GPGPU device. The partial level-2 currents from each MPI process are then accumulated at MPI rank 0 where the complete level-2 neuron dynamics are evaluated and the image detection decision is made. The level-2 neuron computations on the hosts were accelerated using OpenMP. Figure 3 elucidates the orchestration of the GPGPU-based MPI implementation discussed in this sub-section.

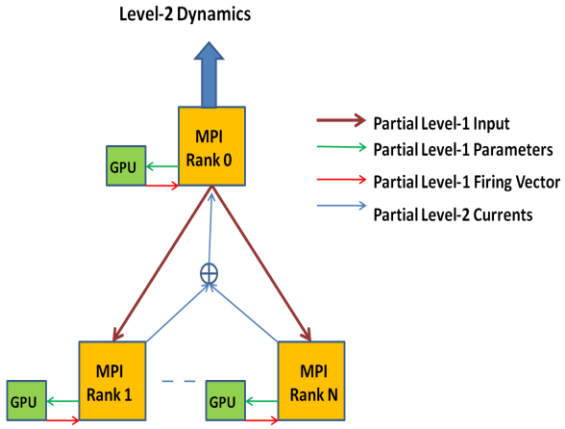

Figure 3. GPGPU-based MPI orchestration

5 Results and analysis

In this section, we present the results for the GPGPU-based MPI implementation of the two-level network developed using the HH and Izhikevich SNN models. We discuss the application runtime values, show the overall runtime breakdown in terms of GPGPU time, CPU time, and communication time for a 32 CPU Core-GPGPU device pair cluster configuration, and compare the GPGPU-based MPI implementation with an equivalent MPI-only implementation. A single CPU Core-GPGPU device pair shall henceforth be referred to as a host-device pair. For the two SNN models, the cluster configuration was varied from 2 up to 32 host-device pairs. We first present the results for the computation intensive HH model followed by the results for the Izhikevich model.

5.1 Hodgkin-Huxley model

The statistical-average runtime values for different cluster configurations versus the network size using the HH model are given in Table II. These runtimes correspond to those measured by the master process, MPI rank 0, which distributes the tasks and makes the final image detection decision. The implementation successfully scaled the two-level network to 200 million neurons using a configuration of 32 host-device pairs with a statistical-average runtime of 2416 milliseconds. The dashes in the table indicate that the problem will not fit in the GPGPU device memory resulting in a configuration failure for that particular neural network size.

Table II. HH model: Statistical-average runtime values (in ms)

Cluster Configuration	**Network Size (in millions)**			
	9.73	**51.8**	**92.16**	**207.36**
2	1256	-	-	-
4	742	3516	-	-
8	486	1928	3396	-
16	412	1168	1990	-
32	356	784	1195	2416

As seen in Table II, the scalability of the implementation improves with the increase in network size. We define the runtime *improvement ratio* as the ratio of runtimes of two successive cluster configurations for a given network size. For a network size of 9.73 million neurons, the runtime improvement ratio is 1.7 for 2 vs. 4 host-device pairs, 1.5 for 4 vs. 8 host-device pairs, 1.2 for 8 vs. 16 host-device pairs, and 1.15 for 16 vs. 32 host-device pairs. However, for a large neural network size, 51.8 million neurons, the improvement ratios are better with values 1.8, 1.65, and 1.5 for 4 vs. 8, 8 vs. 16, and 16 vs. 32 host-device pairs, respectively. The above scaling behavior is expected since the amount of computations per GPGPU device decreases with the host-device pair scaling. Consequently, for smaller network sizes, the GPGPU computations are not sufficient to amortize the necessary CPU computations and MPI communications.

Figure 4 further supports the scalability explanation given above. The figure provides the overall runtime broken into: GPGPU time (kernel time and host-device transfer time), CPU time (level-2 currents and dynamics), and MPI communication time for a 32 host-device pair configuration versus the network size. For a small network configuration of 13 million neurons, CPU time dominates the GPGPU time owing to a relatively small number of computations per GPGPU device. As the network size increases, the number of computations per GPGPU device increases significantly, thereby making the GPGPU time dominant with respect to the overall runtime.

Figure 5 provides the speed-up of the GPGPU-based MPI implementation over an equivalent MPI-only implementation. The 32 host-device pair configuration was able to achieve a speed-up of 281.8x over the equivalent MPI-only implementation for the largest SNN network size. Table III provides the speed-up values for many of the intermediate network sizes tested. As shown in Figure 5 and Table III, the speed-up over the equivalent-MPI implementation increases with the increase in network size for all of the cluster configurations. The increase in speed-up is due to the amortization of CPU and MPI communication times by the GPGPU computations due to the increased number of GPGPU computations required by the increasing network size. The speed-up values are particularly large for the HH model due to its high Flops/Byte ratio requirements (see Table I). This supports the claim that applications with high Flops/Byte ratios are highly suited for GPGPU-based implementations [8].

Further inspection of Figure 5 and Table III reveals that for a fixed network size, the speed-up of the GPGPU-based MPI implementation over the equivalent MPI-only implementation falls as the number of processors increases. As explained previously, a significant number of computations are required to fully utilize the compute capabilities of the GPGPU device; hence large cluster configurations observe lower speed-ups for smaller network sizes.

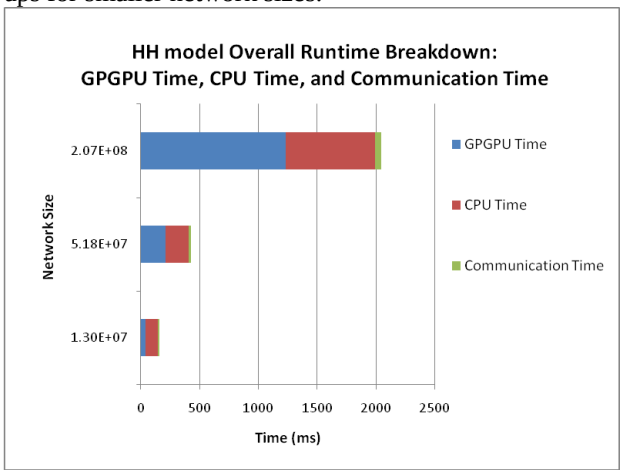

Figure 4. HH model: Overall runtime breakdown for 32 host-device pair configuration

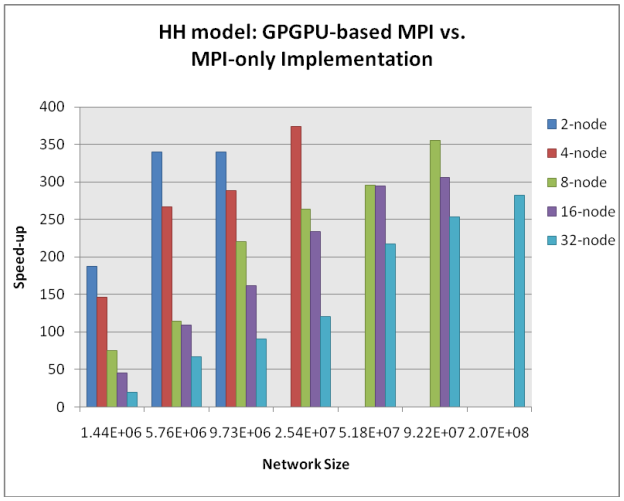

Figure 5. HH model: GPGPU-based MPI vs. MPI-only implementation

Table III. HH model: GPGPU-based MPI vs. MPI-only implementation

Cluster Configuration	Network Size (in millions)			
	1.44	9.73	25.4	92.2
2	187x	340x	-	-
4	146x	288x	374x	-
8	75x	220x	264x	355x
16	44x	162x	233x	306x
32	20x	90x	120x	253x

5.2 Izhikevich model

The statistical-average runtime values for different cluster configurations versus the network size using the Izhikevich model are given in Table IV.

Table IV. Izhikevich model: Statistical-average runtime values (in ms)

Cluster Configuration	Neural Network Size (in millions)			
	9.73	51.8	921.16	207.36
2	180	-	-	-
4	142	928	-	-
8	118	952	725	-
16	96	363	453	938
32	96	491	253	498

Unlike the high Flops/Byte ratio HH model, strong scaling is not observed for the low Flops/Byte ratio Izhikevich model as seen in Table IV. In addition to the lower number of computations in the Izhikevich model (see Table I); the fall in the number of computations per GPGPU device further impedes the scaling performance. Figure 6 provides the overall runtime breakdown for the 32 host-device pair configuration in terms of CPU time, GPGPU time, and communication time.

As seen in Figure 6, the CPU time continues to dominate the kernel time as the network size increases, leading to sub-optimal performance for the Izhikevich model. The continued domination of the CPU time is due to the increased level-2 current computations as the network size increases. Although computations per GPGPU device also increase with the increase in network size, the increase is marginal due to nominal number of computations in the Izhikevich model.

Figure 7 and Table V present the performance comparison of the GPGPU-based MPI implementation and MPI-only implementation. The 32 host-device pair configuration attained a speed-up of 2.87x versus the 32-processor MPI-only implementation. As seen in Table V, the increase in speed-up with the increase in network size is marginal for all cluster configurations examined. The explanation for the fall in the speed-up with the increase in cluster configuration for fixed network size is the same as was given for the HH model.

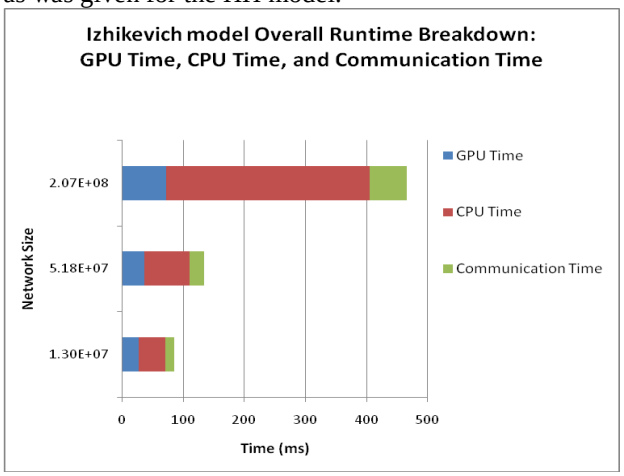

Figure 6. Izhikevich model: Overall runtime breakdown for 32 host-device pair configuration

The Izhikevich model is an interesting case for GPGPU-based MPI implementation. Although the application itself is massively-parallel, it involves only a nominal amount of computations. Therefore, the GPGPU computations do not amortize the increased CPU computation and MPI communication overhead as the SNN network size increases. As mentioned in this sub-section, the level-2 current evaluation for the Izhikevich model increases significantly with the SNN network size. One possible improvement is to implement the level-2 current evaluation on the GPGPU device for larger network sizes. The task would not only require meticulous handling of the GPGPU device memory for the inherent reduction operation involved, but also accommodation of the large weight matrix (48 * Network Size) in the GPGPU device memory. The Izhikevich model explored in this research serves well to highlight the importance of an optimal application-to-accelerator cluster match. It is claimed that applications should not only expose sufficient parallelism, but should also yield enough computations to fully utilize the compute capabilities of heterogeneous clusters. Nonetheless, our GPGPU-based MPI implementations produced performance advantages versus the equivalent MPI-only implementations as shown in this section.

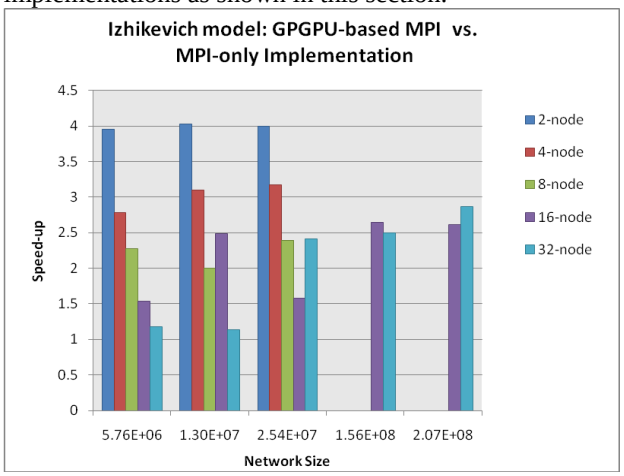

Figure 7. Izhikevich model: GPGPU-based MPI vs. MPI-only implementation

Table V. Izhikevich model: GPGPU-based MPI vs. MPI-only implementation

Cluster Configuration	Neural Network Size (in millions)			
	5.76	13	25.4	156
2	3.9x	4.0x	4.0x	-
4	2.8x	3.0x	3.2x	-
8	2.3x	2.0x	2.4x	-
16	1.5x	2.5x	1.6x	2.7x
32	1.2x	1.1x	2.4x	2.5x

6 Conclusions and future work

In this research, we explored the multi-level parallelism offered by heterogeneous GPGPU clusters for large-scale SNN simulations in the form of a complete GPU+MPI+OpenMP implementation. A GPGPU-based MPI orchestration was presented that allows for optimal heterogeneous resource utilization for large-scale SNN simulations. The two-level network based on the HH and Izhikevich SNN models successfully scaled to 200 million neurons using a 32 host-device pair cluster configuration. In addition to providing significant speed-ups, as high as 355x over an equivalent MPI-only implementation, the GPGPU-based MPI implementation for the HH model scaled well with the SNN network size. Although, our GPGPU-based MPI implementation for the Izhikevich model performed slightly better than the MPI-only implementation, sub-optimal scaling was observed with increasing SNN network size. The results for the Izhikevich model implementation highlighted the

importance of an optimal application-to-accelerator cluster match for maximum application performance. It is claimed that applications should not only expose sufficient parallelism, but should also yield enough computations to fully utilize the compute capabilities of the heterogeneous cluster resources.

As future work, we plan to explore other cluster configurations with different CPU Core-to-GPGPU device ratios per node and investigate the performance of such configurations for large-scale SNN simulations. The future work also includes the development of performance prediction models for heterogeneous clusters that optimally matches the applications with appropriate heterogeneous cluster configurations. As encouraged by the large-scale simulation efforts presented in this research, we also plan to explore GPGPU cluster based smart control of power grids that employs biologically inspired SNNs as massively parallel computation engines.

7 Acknowledgement

This work was supported in part by the National Science Foundation under Grant No. CCF-0916387. The authors gratefully acknowledge the National Center for Super-Computing Applications (NCSA) for granting access to their computing resources.

8 References

[1] E. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?", *IEEE Transactions on Neural Networks*, vol. 15(5), pp. 1063-1070, 2004

[2] A. Gupta, L. Long, "Character Recognition using Spiking Neural Networks," in *Proc. IJCNN*, pp. 53 – 58, August 2007

[3] D. Surdilovic, J. Radojicic, M. Schulze, M. Dembek, "Modular hybrid robots with actuators and joint stiffness control", in *Proc. BioRob*, pp. 289-294, October 2008

[4] C.E. Johnson, "Spiking Neural Networks and their Applications", Ph.D dissertation, 2011

[5] E. M. Izhikevich, "Simple Model to Use for Cortical Spiking Neurons," *IEEE transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, November 2003

[6] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and application to conduction and excitation in nerve," *Journal of Physiology*, vol. 117, pp. 500-544, 1952

[7] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbauma, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Special issue of Neural Network, Elsevier*, vol. 22(5-6), pp. 791-800, July 2009

[8] V. K. Pallipuram, "Acceleration of Spiking Neural Networks on Single-GPU and Multi-GPU systems", Master's Thesis, 2010

[9] E. M. Izhikevich, "Dynamical Systems in Neuroscience," *MIT press*, Cambridge, Massachusetts, 2007

[10] "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf

[11] "NVIDIA CUDA Programming Guide", http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf

[12] C. Johansson and A. Lansner, "Towards Cortex Sized Artificial Neural Systems," *Neural Networks*, 20(1), pp. 48-61, January 2007

[13] 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," *Proceedings of SC '09*, Portland, Oregon, November 2009

[14] B. Han, T. M. Taha, "Neuromorphic models on GPGPU cluster", in *Proc. IJCNN*, pp. 1-8, July 2010

[15] V. K. Pallipuram, M. A. Bhuiyan, M. C. Smith, "A Comparative Study of GPGPU Programming Models and Architectures Using Neural Networks", *Journal of SuperComputing*, Springer Publications, DOI 10.1007/s11227-011-0631-3

[16] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu, "GPU Clusters for High-Performance Computing", *PPAC 2009*, in *Proc. IEEE Cluster 2009*, August 2009