Hybrid CPU-GPU Model Based Simulation of Spiking Neural Networks Using a Look-up Table

Sreenivasa.N, S. Balaji

Abstract *Brain is one of the most complex human organs. For long this has intrigued a number of researchers from various disciplines in the world. A lot of research has been done on brain since early days especially from the physiological, psychological angle. However, the advances in the computing technologies especially the High Performance Computing platform has opened several new avenues for researchers to carry out their research. In the in-silico approach of modelling the brain, the simulation is performed by constructing the networks which draw inspiration from the simple neuron model. We have performed a study on the hybrid CPU-GPU model based simulation of Spiking Neural Network. Our current work is to study the scalability by simulating the entire pipeline on a GPU.*

Keywords: *Izhikevich Model, Spiking Neural Networks, Spiking Neuron, CUDA.*

I. INTRODUCTION

The brain has been the driving factor behind the revolution in Artificial Intelligence (AI). This has led to designing of a number of applications based on AI. The artificial neural network models have inspired a lot of researchers to make an endeavour towards this and many have succeeded. A lot of research effort has been put into Machine Learning and then with the advent of powerful hardware researchers have ventured into Deep Learning starting with, Convolutional Neural Networks (CNN) then eventually graduated to Artificial Neural Networks (ANN) and now several proposals have been put forth to study the mechanism of brain using computational models based on Spiking Neural Networks (SNN).

The authors in the works [3][4] state that the synchronization of the spiking neurons in the brain facilitate the manifestation of the computational activities in the brain such as learning, the attribute of being attentive etc. Furthermore, ever since the electroencephalogram (EEG) rhythms have observed the synchronization activity of these spiking neurons in the brain has garnered tremendous traction. The authors in [3] state that the relation between the synchronization activity and the cognitive processes has become very active. Spiking Neural Networks have introduced a new paradigm shift in characterizing the neural networks. Unlike the firing rate models, the SNNs inherently are imbibed with the high precision time structure created by spike trains. This very characteristic manifests attributes like temporal binding which happens due to neurons firing in a synchronized way [2][5] and the feed-forward propagation nature.

The authors of work [6][8] states that the SNNs can be model using multiple characteristics of the brain architecture as this model have a high biological fidelity. The spikes can be generated in SNNs using their inherent property of performing an event driven processing which results in a faster response. In this work, we investigate the synchronization and its scalability on the GPU using a cache look-up table approach. The authors in [4] claim that the activation of a group of neurons which drives the synchronization which in-turn generates the neural code will lead to development of new learning techniques.

Even though we can, in principle, can say that the SNNs can be used in multiple applications, in a near real-time or a real-time scenario it becomes imperative to put some efforts in understanding the nitty-gritties of the SNN so that we can optimally scale up the performance of the SNNs on a large network or neurons. Scaling up huge networks on conventional computing systems with CPU is not possible since the CPU has to handle both event driven and time driven activities. The context switch can get very expensive as the network grows and hence this is not a very approachable model. With parallel processing models using MPI or OpenMP though we can parallelize some of the computations, the performance enhancement is not very significant and there is always an overhead in handling large number of threads more so when we want to simulate a very large network.

To overcome this challenge we study the performance impact by using a cache table model. The Compute Unified Device Architecture (CUDA) is a parallel processing framework from NVIDIA which enables programmers to leverage the parallelism capabilities of the GPU. Here are some reasons to leverage GPUs for simulating SNNs:

(a) A very high degree of multi-threading with multiple threads running in parallel across the cores of the GPU.
(b) One of the most important reasons is that the GPUs by design are highly efficient at context switching which significantly reduces the idle time of the machine.
(c) The hardware is naturally suited to perform special class of mathematical operations involving trigonometric equations and large matrices very efficiently

Revised Manuscript Received on April 15, 2019.

Sreenivasa.N, Research Scholar-Jain University, Dept. of Computer Science & Engineering., Nitte Meenakshi Institute of Technology, P.O. Box 6429, Yelahanka Bengaluru-560064, India

S. Balaji, Centre for Incubation, Innovation, Research and Consultancy, Jyothy Institute of Technology, Tataguni, Off Kanakapura Road, Bengaluru-560082, India

These benefits for GPUs make it very conducive to perform many parallel simulations involving many thousands of neurons as a thread in the GPU. Nevertheless, these points still do not address the bottleneck conditions very well in a sense that as and when the size of the neural network increases exponentially the memory bandwidth proportionately decreases. Furthermore, the SNN model of biological functions tend be more communication intensive than compared to computation and thus even though the hardware intrinsically supports mathematical functions and hence the performance sometimes tend to be constricted by maximum bandwidth that can be achieved by GPUs.

In the current work, we explore various avenues to enhance the performance of the simulation of Spiking Neural Networks. We now intend to further explore the Izhikevich models to simulate the large SNNs and study their performance. We will still face some challenges in leveraging GPUs to accommodate the cache / look-up table that contains the time-driven events which were handled by CPU.

(a) The degree of effective parallelism
(b) Optimizing the fan-in/fan-out neuron connectivity
(c) Optimal strategies to leverage the limited memory of GPU by using sparse representation of the neural networks

We further intend to focus on using a GPU cluster to simulate the large SNNs and study the effect of hype-parameter tuning on the scalability of the simulations.

II. METHODOLOGY

Before we get into the methodology, we will look into the components involved in the simulation of large-scale SNNs. The major components are outlined in Fig 1 (as mentioned below):

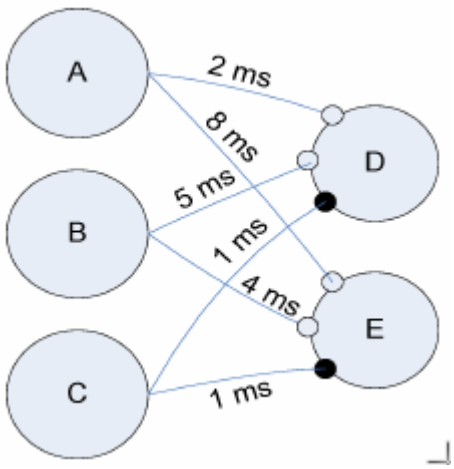

Fig. 1: A Simple Illustration of Cortical Networks

Neurons are labelled as alphabets and the axonal delay is also illustrated. Small circles represent synaptic neurons. From the above figure, it is trivial to note that the major components involved in simulating large scale SNNs are:

- Neurons required for the processing of spikes,
- Inter-spike communication (axons and dendrites),
- Synapses to bridge the neurons to facilitate learning and information storage.

For a better and more efficient simulation, it is very important to generate a variety of neural responses. To this end, in the present work, we leverage Izhikevich's model to model the neuronal dynamics. This is done because. as compared to conventional Integrate-and-Fire (I&F) model, Izhikevich's model generates a large variety of neurons and also it is computationally less intensive compared to Hogdin & Huxley Model (H&H) model [1].

In Izhikevich model[1], the neurons are characterized using the dimensionless constants, the membrane potential and recovery variable.

Let

v be the membrane potential of the neuron
u be the recovery potential
a, b, c, d be the dimensionless constants,

Then the neurons of the Izhikevich model can be represented as follows:

$$v' = 0.04v^2 + 5v + 140 - u + I \quad (1)$$

$$u' = a(bv - u) \quad (2)$$

$$\text{if } (v \geqslant +30mv) \text{ then } v = c \text{ and } u = u + d \quad (3)$$

Fig. 2 below shows the neural response. The loss-less cable models are used to simulate the axons characterized by distance dependent conduction delay as shown in Fig. 1.

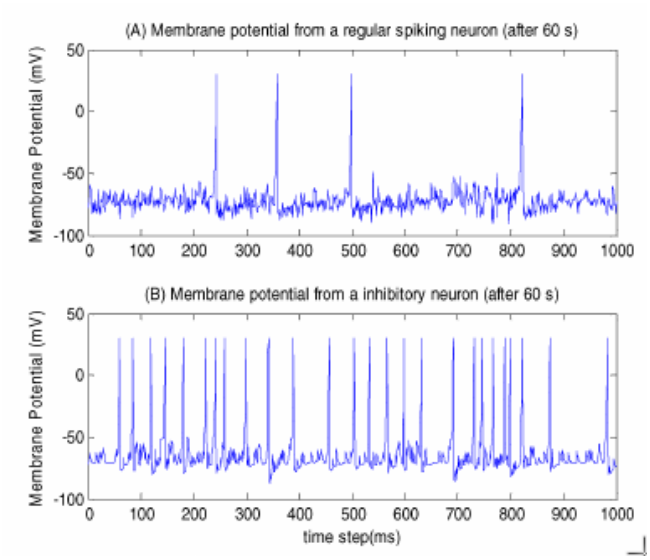

Fig 2: Neuronal Response

Let us consider Fig. 1 which represents the axonal conduction delay. When any neuron 'n_f' fires the recipient neuron 'n_r' receives the event only after the associated delay'd'. Such axonal conduction delays contribute in generation of spatial temporal neural firing pattern [5][6] which is stable and is time locked in nature. The synapses between 2 neurons help characterize the strength of connection between them. The STDP rule states that the timing intervals among the firing at the pre-synaptic and post-synaptic side characterizes the degree and sign of synaptic modification. This mechanism drives the synaptic connections to contend with the others to gain domination over the firing of post-synaptic neuron. The authors of work [4][5] suggest that this kind of contention characterizes the stability of firing patterns which can be leveraged for simulating large scale SNNs.

III. THE GPU ARCHITECTURE

The following Fig.3 shows a generic view of the CUDA GPU architecture from NVIDIA [14]. A GPU, at the hardware level is composed of multiple Streaming Multiprocessors (SM). As mentioned earlier, GPUs intrinsically are very efficient when it comes to floating point calculations which is due to the presence of floating-point scalar processors in each SM. Each SM generally is made up of 8 such SPs. Apart from all these, a GPU consists of a Special Function Unit (SFU) which handles multi-threaded instructions and is divided into shared memory which is user managed and a cache memory of 16KB which can further be divided into 8KBs each for a constant cache and texture cache.

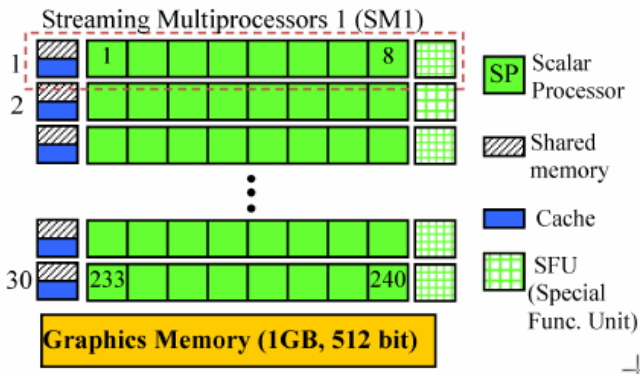

Fig 3. Simplified CUDA Architecture

In our research work, we use a single NVIDIA GeForce GTX 1060 GPU that consists of 1286 CUDA cores and 10 SMs each operating at 1.27GHz. Every SM has a hardware thread scheduler that schedules a group of threads for processing. If any of the threads in this group has set of instructions that makes and external context switch, then a new thread group is spawned by the thread scheduler. With the kind of attributes mentioned above, it is simple to notice that the GPU maintains many active threads at any instant of time and this makes it possible to execute parallel instructions. Also, the scheduler can manage the memory latency trade-off by switching the warps. This GPU has a total of 6GB GDDR5 memory and a bandwidth of 192 GBPS.

Some of the Key Performance Indicators (KPI) that affects the performance of a large-scale SNN on CUDA GPUs is discussed below:

(1) **Parallelism**: This is one of the trickiest parameters to handle on a GPU. The degree of parallelism that can be achieved on a GPU is very important parameter and can seriously affect the performance of the application on the GPU. One needs to be very careful while mapping the data to GPU for computation because an in-efficient mapping can lead to inefficient utilization of the GPU and thus lower the performance of the application. One of the vital KPIs of a GPU are stalling condition and to manage the trade-off it is imperative to launch a large number of threads in-order to ensure that the GPU is not stalled which, otherwise could lead to a lower performance of the application.

(2) **Memory Bandwidth**: For the most optimal performance, it is very vital to achieve a peak memory bandwidth. To this effect we need to ensure that each processor of the GPU should have uniform access to the memory. Uniform memory access facilitates coalescing operations.

(3) **Memory Consumption**: The design and choice of data structures of the simulator has a strong impact on the memory bandwidth and the scaling of SNN simulation experiments. In order to achieve optimal scaling performance we have tried to use the strategies to ensure minimized memory consumption. These include sparse connectivity and by incorporating Address-event-representation format for loading firing data. Some other techniques which involve compression can also be used to eliminate redundancy for memory optimization.

(4) **Multi Thread attributes:** The CUDA GPUs by design select a warp of 32 threads which are executed using a single instruction register. It is important to note here that if all the threads execute the same instruction then the warp as a whole would be executing just one instruction which means maximum performance. One of the major bottlenecks would be in a situation where not all threads of the same warp execute the same instructions and few different threads might have different branching within the same warp. This leads to a very poor performance as the GPU's hardware is sub-optimally utilized. This situation is known as multi-threaded divergence.

It is very vital at this point to note that all the aforementioned KPIs are proportionately dependent on each other and for a holistic optimization we need to optimize all the KPIs.

IV. THE GPU MAPPING

There are certain strategies that can be employed to efficiently map the SNNs onto the GPUs. As discussed in previous sections it is very important to design the pipeline which is in alignment with the inherent hardware of the GPU in order to achieve maximum performance.

4.1. Parallelism

To achieve maximum parallelism in scaling SNN we need to address the three different systemic parallelisms such as neuronal, synaptic and neuronal-synaptic. The strategies that have been employed are discussed below:

4.1.1 NEURONAL [8]-[16] LAYER PARALLELISM

The idea here is to be able to compute each neuron in parallel by mapping them onto the processing element of the GPU. However, it is important to note that the synaptic computations are carried out sequentially on the processing element of the GPU. As discussed in the previous section, this results in multi-thread divergence because each synaptic computation may have its own branching and it may not execute the same instructions and this is rendered ineffective.

4.1.2 Synaptic [14]-[17] Layer Parallelism

Similar to the idea of neuronal parallelism, the idea here is to achieve computational parallelism at the synaptic layer. Imagine a neuron 'n' has 'n_s' number of synaptic connections. The idea here is to update all the n_s number of synaptic connections updated in parallel. Note that these computational instructions are distributed across the processing elements of the GPU. It is to be noted that since neuronal computations are carried out sequentially the degree of parallelism gets limited. The amount of parallelism achieved in this context is measured by the number of synaptic connections that a neuron has which are also updated in parallel. This intuitively means that denser the network, more computationally expensive it is and lower the performance.

4.1.3 Neuronal-Synaptic Layer Parallelism

This is a combination of the aforementioned parallelism strategies. The neuronal approach is used whenever a neuron fires and the information has to be updated so that all the threads in the warp will be executing the same instruction and thus end up achieving maximum parallelism. Furthermore, the synaptic approach is used when the spikes are generated which results in updating of all the connected synapses which are distributed across the processing elements of the GPU [18][19].

Since the shared memory is available on the GPUs, it proves to be very conducive to use that for storing the cache table which stores the synaptic network information and hence minimizing the context switch between CPU and GPU. This mapping can be done within the SMs by leveraging the shared memory and due to its fast synchronization.

4.2. Impact Minimization

As stated in the previous sections, when the threads of the same warp have their own branching instead of executing the same instructions, warp divergence occurs which limit the performance of the parallelism that can be achieved. The impact of the divergence is determined by the branching condition i.e larger the cycles more the impact of divergence. It is very important to keep the divergence impact minimum to achieve a better scalability of SNNs. This can be done by employing a buffer to store the information of the diverging loop in the shared memory / cache table with the condition of delayed processing until information is available for all the threads and keep the operations atomic.

4.3. Representation of Parameters

One of the strategies employed to map the SNN pipelines on to GPUs is to represent the network parameters in a sparse manner. Consider a SNN with *N* neurons, *M* Synaptic Connections and an axonal delay of *D*, it is trivial to note that the required memory in case of non-sparse representation will be $O(NDM)$. Furthermore, it can be proved that by employing the sparse image of the required memory can be brought down to $O\big(N(M+D)\big)$. The strategies used to this representation are as shown in Fig. 4.

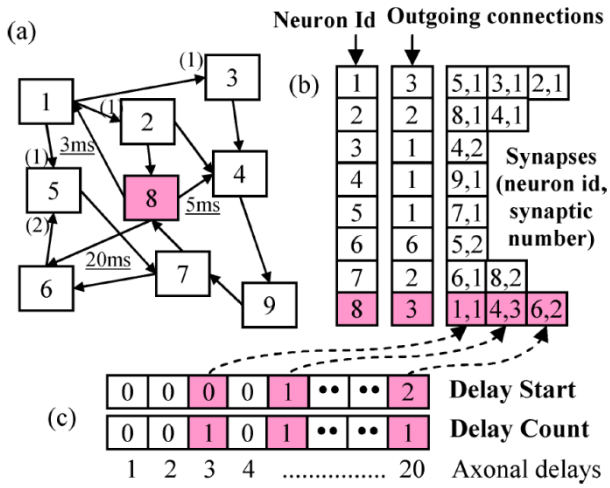

Fig. 5: (a) A Sample Network (b) Neuron Ids and Post-Synaptic Connections (c) Delay Information for Neuron with Id 8.

From Fig 5 it can be noticed that each neuron has a unique id and the post-synaptic connections are also mentioned. The pair of neuron id and synapse id <N_{id},S_{id}> is used to uniquely identify a synaptic connection denoting from where the synaptic connection is originating from. Whenever there is a spike it is not instantaneously transferred, rather, reaches the destination neuron after the corresponding axonal delay associated with that neuron. These delays are stored in the data structures named Delay Start and Delay Count. It is to be noted here that the delay count at an index *i* indicates the number of neurons with an axonal delay of *i* ms. Also, in Delay start the i^{th} element is the first synaptic connection with a delay of *i* ms.

4.4. Event Queue

In SNN simulation experiments, generally circular queue data structure is used [8][15] to store the firing information from the neuron. The subsequent set of synaptic events which results from a neuron firing event are added onto the circular event queue. Unlike the circular queue mechanism, in the Annual Equivalent Rate (AER) format an address-time pair <$addr_n$,$time_n$> is used to represent a spike. The downside of this is that we must unnecessarily store the time at which the event was fired along with addresses of each firing neutron. We have leveraged the AER format for this representation. The AER approach has two tables: firing count table and firing event table.

The former indicates how many excitatory neurons that have fired up until the current time. However, storing this is not enough and hence along with the number of neurons fired we also store which neurons have fired recently. Intuitively, this means that we need far lesser amount of memory compared to traditional mechanisms. Furthermore, we must periodically update the table to get rid of the old values to avoid dirty read conditions. Fig. 6 shows a representation of the improved AER format.

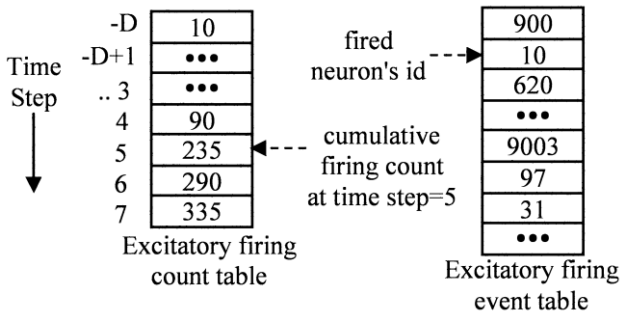

Fig. 6: Improved AER Format

4.5. GPU Simulation

Fig. 7 shows an overview of how SNNs are simulated on the GPU.

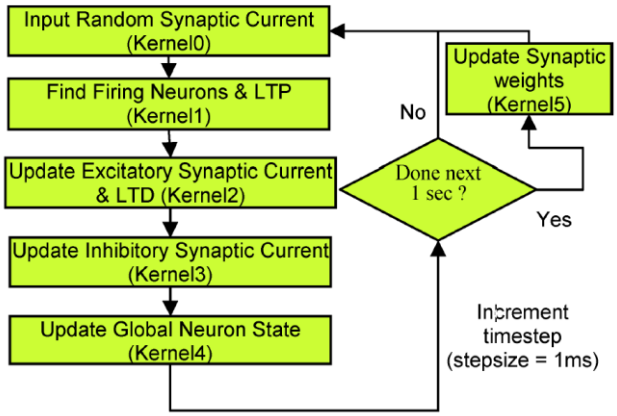

Fig. 7: Overview of SNN Simulation on GPU

Due to the intrinsic parallelism support of the GPU, we can execute each 'kernel' in parallel. The role of CPU in such experiments is to facilitate the lifecycle of the kernel. Intuitively, this means that the CPU is responsible for the creation, deletion and facilitation for the execution of the kernel. We have designed an asynchronous communication between CPU and GPU in the sense that once the kernel is launched by the CPU it does not have to wait until kernel sends a request for termination. This allows the CPU to work independently of GPU as well as have a concurrent execution. We have chosen the block sizes to be in range of 30-120 for experimentation purpose with a thread count of 128 per block. The change in performance that was noticed for blocks of size greater than 60 was very negligible.

We have used a combination of N excitatory and inhibitory neurons. It is to be noted that these neurons are randomly connected and the ratio of excitatory versus inhibitory neurons is 4:1, with each neuron's degree of post-synaptic connectivity to be M. The rate of change of synaptic-weights is captured by the kernel as shown in Fig. 7.

V. RESULTS

5.1 Memory

Table 1 shows the memory calculations for an SNN with N neurons, M synaptic connections per neuron and a max axonal delay D.

Table1: Memory Calculations

SNN Components	Memory Required in Words
Neuron information	$5N + \frac{MN}{32}$
Synaptic weights and STDP	3NM
Network representation, delays	(NM+ND+3N)
Firing Info	50N
Firing Info (Buffer)	5N

Considering 32 bit floating point and NVIDIA GeForce GTX1060 6GB GDDR5 we were able simulate a network of 250k neurons with M = 250.

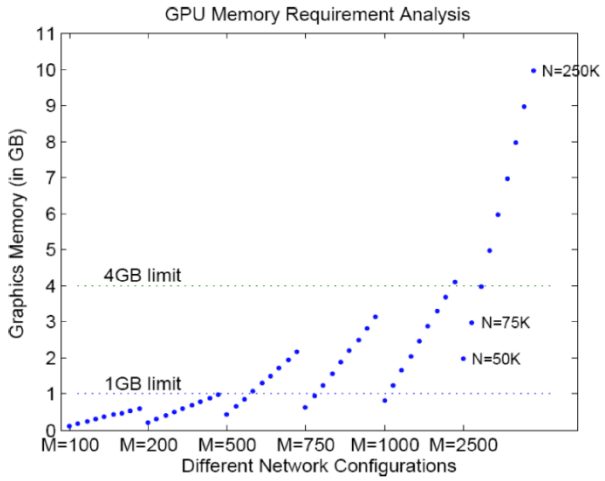

Fig. 8: Memory Requirements for varied SNN Configurations

5.2 Scalability

As shown in Fig 8, we can see the performance for various value of N and M = 200. Also, Fig. 8 shows the average firing rate obtained under various conditions. We can see an increase in the firing rate has a positive impact on the speedup. A steady speedup is observed because the performance primarily depends on memory bandwidth of GPU and for a large value of N it seems to saturate.

VI. CONCLUSION

We have presented various techniques to achieve a near real-time mapping of large SNNs on GPUs. These techniques have opened a number of avenues to leverage the GPUs for such modelling activities. The GPU based cache / look up table model provides better flexibility and proves to perform better. The GPU implementation was up to 30 times faster than the CPU simulation. Though the performance is limited by memory-bandwidth, we can use GPU cluster in future to simulate up to millions of neurons.

REFERENCES

1. S Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson International Edition, 2009
2. H. Paugam-Moisy, S.Bohte,"Computing with Spiking Neuron Networks", Handbook of Natural Computing, Springer, Heidelberg, 2009.
3. X.J. Wang, "Neurophysiological and Computational Principles of Cortical Rhythms in Cognition", Physiol. Rev. vol. 90, pp. 1195-1268,2010
4. E.M.Izhikevich, Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting, The MIT press, 2007.
5. M. M. McCarthy, C. Moore-Kochlacs, X. Gu, E. S. Boyden, X. Han, N. Kopell, "Striatal Origin of the pathologic beta oscillations in Parkinson's disease", PNAS, vol. 108, pp.11620-11625, 2011.
6. E. M. Izhikevich, "Polychronization: Computation with spikes,"Neural Computation, vol. 18, no. 2, pp. 245-282, February 2006.
7. S. Song, and L.F. Abbott, "Cortical Development and Remapping through Spike Timing-Dependent Plasticity", Neuron, Volume 32(2),339 – 350.
8. T.Simon, A.Delorme, R.Van Rullen, "Spike-based strategies for rapid processing", Neural Netwosrks 14: 715-25,2001.
9. A.Rajagopal, Dharmendra S. Modha: "Anatomy of a cortical simulator", SuperComputing, 2007.
10. E. M. Izhikevich, "Simple model of spiking neurons," Neural Networks, IEEE Transactions on, vol. 14, no. 6, pp. 1569-1572, 2003.
11. Kepecs etal., 2002, "Spike-timing-dependent plasticity: Common themes and divergent vistas", Biological Cybernetics. v87. 446-458.
12. Kayvon Fatahalian and Mike Houston, "A Closer Look at GPUs", Communications ofthe ACM. Vol. 51, No. 10, October 2008.
13. P Merolla, J Arthur, B E Shi and K Boahen, "Expandable Networks for Neuromorphic Chips", IEEE Transactions on Circuits and Systems I, vol 54, No 2. pp. 301-311, February 2007.
14. NVIDIA Programming manual Version 2.0. See Appendix A for technical specificationsJahnke, T. Schonauer, U. Roth, K. Mohraz, H. Klar, "Simulation of Spiking Neural Networks on Different Hardware Platforms", International Conference on Artificial Neural Networks (ICANN), pps: 1187-1192, 1997.
15. H.E.Plesser,J.M.Eppler, A.Morrison, M.Diesmann, M.O.Gewaltig, "Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers", Euro-Par, 2007.
16. A.Morrison, C.Mehring, T.Geisel, A.Aertsen, and M.Diesmann, "Advancing the boundaries of high-connectivity network simulation with distributed computing",. Neural Computing. 2005 Aug; 17(8): 1776-801.
17. Ernst Niebur, Dean Brettle, "Efficient Simulation of Biological Neural Networks on Massively Parallel Supercomputers with Hypercube Architecture", NIPS 1993, pp: 904-910.
18. R. J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, R. Etienne-Cummings, et. al, "A Multi-Chip Neuromorphic System for Spike-Based Visual Information Processing,,", Neural Computation, vol. 19 (9), pp. 2281-2300,2007.
19. F.Bernhard, and R.Keriven, "Spiking neurons on GPUs", International Conference on Computational Science: Workshop on GPGPU, May 2006.