International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-5, June 2019

Empirical Analysis of Spiking Neural Networks Using CPU, GPU and GPU Clusters

Sreenivasa.N, S. Balaji

ABSTRACT--- Many attempts have been made to study the neural networks and to model them. These attempts have led to the development of neural network simulation software packages such as GENESIS and NEURON which have been the de-facto simulators for some time now. However, further studies have found that one of the major hindrances in using the afore-mentioned simulators is speed. These simulators use time driven technique which isolates the mimicked time to brief time periods and in every progression the factors of neural states are estimated and reiterated through a numerical examination strategy. This method includes complex calculations which do not foster the development of scalable neural systems; hence, there is a demand for quick re-enactment of neural systems which led to an alternative strategy known as event driven simulation. This technique processes and appraises the neural state factors when another event alters the typical advancement of the neuron, that is, the point at which information is created. In the meantime, it is realized that the data communication in neural networks is done by the purported spikes. Less than 1% of the neurons are at the same time dynamic which catalysis the efficiency of event-driven Spiking Neural Networks (SNN) simulation. Brain is one of the most complex human organs. For long this has intrigued several researchers from various disciplines in the world. A lot of research has been reported on brain since early days especially from the physiological and psychological angles. However, the advances in the computing technologies especially the High-Performance Computing (HPC) 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 present an empirical study on the hybrid CPU-GPU model-based simulation of SNN which are based on our works [1][2].

Keywords - SNN, CPU-GPU co-processing, high performance computing, neural networks, numerical analysis.

1. INTRODUCTION

The SNN is a branch of artificial neural networks that thoroughly mimic the natural neural networks [3][8]. The simplified model of an SNN is illustrated in Figure 1. Neurons are represented by N_j and the presynaptic links X_i of a neurons are labelled as small circle W_i. Then the synaptic weight formula is as follows:

$$\text{Synaptic weight[j]} = \sum_1^n X_i \, W_i.$$

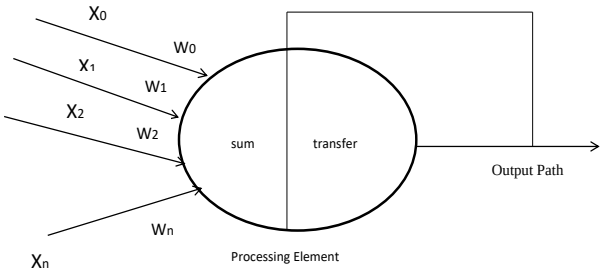

Figure 1: Simplified Spiking Neural Network

SNNs also comprise the concept of time in addition to the neuronal and the synaptic state in their operating model [15]. Unlike the multi-layer perceptron, the neurons in this model fires only when the threshold of the membrane potential is reached [14] [16]. A signal is generated as the response to the firing of neurons travels to other neurons thus altering the potentials depending upon the generated signal [6][7]. SNNs aim to link the gap between neuroscience and machine learning with biologically realistic prototypes of neurons to execute computations. As the name suggests spikes are mathematically discrete events [11]. Membrane potential of a neuron is the most important biological parameter whose differential equations can be used to determine the occurrence of spike in a neuron. When a neuron reaches a certain threshold potential, it spikes that causes the potential to be reset and Leaky Integrate-and-Fire (LIF) is used to model this particular behaviour of neurons [10]. Also, since SNNs are inherently sparse we can leverage the corresponding network topology to model the connections in an SSN.

The overall architecture of the SNN in general is as shown in Figure 2.

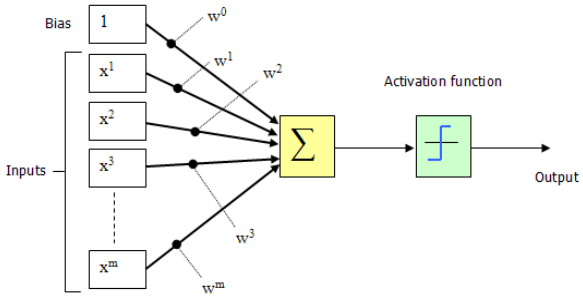

Figure. 2: SNN Architecture

The spike trains offer us enhanced ability to process spatio-temporal data or real-world sensory data. The spatial aspect refers to neurons that are connected to other neurons which are near to them and hence these inherently process chunks of the input independently

Manuscript published on 30 June 2019.
***** Correspondence Author (s)
Sreenivasa.N, Research Scholar-Jain University, Dept. of Computer Science & Engineering Nitte Meenakshi Institute of Technology, Yelahanka Bengaluru-560064, India. (E-mail: meetcna@yahoo.co.in)
S. Balaji, Centre for Incubation, Innovation, Research and Consultancy Jyothy Institute of Technology, Tataguni, Bengaluru-560082, India. (E-mail: drsbalaji@gmail.com)

(similar to a Convolutional Neural Network would using a filter). The temporal aspect refers to the fact that spike trains occur over time, so what we lose in binary encoding, we gain in the temporal information of the spikes [9]. This permits to naturally process temporal data without the extra complexity that Recurrent Neural Networks (RNNs) improve. In fact, the spiking neurons are fundamentally more influential computational units than conventional artificial neurons [23]-[25].

There has been a lot of on-going research in this area: Francisco Naveros et al [5], EDLUT, Hamid Soleimani at el, Jesus A. Garrido et al, Govind Narasimman et al [6]. However, the major challenges faced were speed of simulation and the scalability of the simulation model which still persist. In this work, we have studied the existing models and proposed a hybrid CPU-GPU co-processing model to address the speed and scalability of simulation challenges. The brain has been the driving factor behind the revolution in Artificial Intelligence (AI). This has led to designing of several 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 (CNNs) then eventually graduated to Artificial Neural Networks (ANNs) and now several proposals have been put forth to study the mechanism of brain using computational models based on Spiking Neural Networks (SNNs).

The authors in the works [5][6] 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 [4][7] and the feed-forward propagation nature.

The authors of work [8] [10] state that the SNNs can be modelled using multiple characteristics of the brain architecture as this model has 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 [6] 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.

Further measuring simulation speed in terms of performance [14], the researchers also recognized the efficiency of initializing and loading big scale models on neuromorphic organisms which remain computational challenge. i.e., cortical microcircuit model created by Diesmann and Potjans [18] took 11 hours for initialising and to load onto the SpiNNaker. This approves previous opinions [19] that neuromorphic prototype system is not effective to accelerate their initialization. The SpiNNaker and previous method of the Brain Scales computational systems consumes a substantial time and energy for setting network model on the host machine.

These issues propose that while evolving SNNs, extra flexible accelerators that can quicken the construction, initialising and the simulation of big scale SNN which are required. FPGA (Field Programmable Gate Array) are strategies consisting of numerous lookup tables centred on logic blocks, which are connected by programmable fabric. FPGA are used to build numerous hard wired SNNs accelerator [18], whereas Naylo demonstrated that they are also used to develop extra flexible and programmable accelerators along with comparable performance [19]. Though, the systems of this type could ideally use to accelerate the initialization and construction of SNN with the simulation, FPGA is not yet ordinary place in workstations and they lack hardware maintenance for floating point computations that makes them ideally not suited for simulating common classes of synapse and neuron models. On the other hand, GPUs are intended for high performance with fine grained parallelism. They substitute the huge coherent caches, trusted on present CPUs architecture to progress the performance, with many floating point computing units attached to high bandwidth memory. The programmable GPU are initially developed to quicken the translation of 3D graphics they usually includes by relating the same, individual computations to every pixel for an instance to compute its illumination. Though, GPUs acceleration proved to be very useful for quickening many other jobs, comprising the exercises of GPUs and deep learning schemes are currently used widely in present Artificial Intelligent systems. The use of GPU quickens to SNNs simulation and also there are numerous dynamic SNNs simulator projects targets GPU. CARLsims [21] is a C++ simulator by using NVIDIA CUDA however, as CARLsim not created on the code generation, it is problematic for the users without CUDA knowledge to add synapse models and novel neuron. EDLUT [22] is an event-driven CPU simulator for SNN, then is has evolved to hybrid CPU-GPU system for both event driven and time driven techniques. ANNarchy [21] is a based on code generation simulation which converts Python model images into CPU or GPU multi-core code with the focus on hybrid spiking models. Further the simulators has less development in the last 3to 4years comprises Myriad, NCS6 [22], and GeNN,NeMo, [21] aimed at assisting accelerated SNNs simulations on GPUs hardware. It is designed to do a balance among flexibility by permitting users to describe their model of synapses, neurons and competence in creating enhanced CUDA code for less apparently parallel phases of SNN parallel simulations such as the spike propagation.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Even though we can, in principle, 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 effort in understanding the nitty-gritties of the SNN so that we can optimally scale up the performance of the SNNs on a large network of 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 [26]-[34]. 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.

These benefits inherent in GPUs make them 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 to be more communication intensive 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 scale SNNs and study the effect of parameter tuning on the scalability of the simulations [18]-[22].

2. THE MODEL

The critical aspect of developing any simulator is its flexibility and accuracy. This is the driving factor for leveraging the parallelism feature of GPU and endeavour towards co-processing (CPU-GPU) model. The parallelism enables us to use the time-driven model and shorter integration time intervals to achieve better accuracy in real-time. In the conventional parallel processing model for SNNs as in NeMo and GeNN, the parallelization in GPU is driven by the CPU to initialize the simulation and, therefore, we cannot leverage the complete potential of the GPU for modelling such a simulation.

On the other hand, when CPU-GPU co-process the events, all the parallelizable tasks can be independently run on GPU, while the CPU handles the sequential events as in Brian [13] [15]. This intuitively means that when the CPU is processing the sequential events, the GPU updates the neural-state. The membrane potential ($V_{m\text{-}c}$) determines the neural state which can be expressed as:

$$C_m \frac{dV_{m-c}}{dt} = g_{AMPA}(t)(E_{AMPA} - V_{m-c}) + g_{GABA}(t)X(E_{GABA} - V_{m-c}) + G_{rest}(E_{rest} - V_{m-c})$$

where

C_m: the capacitance membrane

E_{AMPA} & E_{GABA}: reverse potential of every synaptic conductance

E_{rest}: resting potential

g_{AMPA} & g_{GABA}: conductance (decaying exponential functions).

The work provides a detailed explanation about the various parameters of a neural model. The state variables of a neuron i.e. $V_{m\text{-}c}$, g_{AMPA} & g_{GABA} help in defining the state of a neuron.

1) $V_{m\text{-}c}$- potential membrane, output spike is generated if the membrane potential reaches threshold value.
2) g_{AMPA} & g_{GABA} - excitatory and inhibitory conductance. These conductance variables modify the membrane potential $V_{m\text{-}c}$. These parameters are inversely proportional to the integration and directly proportional to the synaptic weight.

A parallel combination of a resistor whose conductance is g_L and a capacitor (C) is used to represent a neuron in the LIF model. To charge the capacitor to produce a potential V(t) a current source I(t) is used as synaptic current as an input. The current input I(t) charges the capacitor and when the potential exceeds the threshold potential V^{th} i.e ($V(t) \geq Vth$) , the capacitor discharges to the potential E_L which uses a switch controlled by voltage to simulate a biological neuron. This model can be represented using the differential equation as shown below:

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

$$C\frac{dV}{dt} = -g_L(V(t) - E_L) + I(t)$$

If the I(t) is low, the potential V(t) manages to be within the threshold potential V_{th} and thus spike is not produced as the necessary condition is not satisfied. To solve the differential of the LIF model, the fourth-order Runge-Kutta method was implemented [12] in their works has processed this differential equation for both event driven (offline) and time-driven (online) techniques. The key performance indices for these methods are size of lookup table and size of integration step, respectively, as the execution time in former case would be directly proportional to the size of lookup table. In our study, we have used Hogdkin-Huxley (H&H) Model [14] [15] to simulate the SNN. The H&H model focuses on 3 channels: (i) sodium channel denoted by *Na* (ii) potassium channel denoted by *K* and (iii) *l* denotes the leakage channel along with the resistance *R*. Let us denote the input current as *I*, which is the sum of the impulses, namely, excitatory impulse which is denoted by I_E, I_I being the inhibitory impulse and I_{offset} being the current offset which is constant. The current which is externally injected to the system is denoted by I_{inj} and the model can be defined by the state equations:

$$\frac{dV}{dt} = \frac{1}{C}\left[-gNam^3h\,(V - E_{Na}) - gkn^4(V - E_k) - g_l(V - E_l) - g_e\,(V - E_E) - g_i(V - E_i) + I_{offset} + I_{inj}\right]$$

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(V)m$$

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n$$

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(V)h$$

$$\frac{dg_e}{dt} = -\frac{g_e}{\tau_{synE}}$$

$$\frac{dg_i}{dt} = -\frac{g_i}{\tau_{synI}}$$

where g_e and g_i, denote excitatory and inhibitory synapses conductivity and g_{Na}, g_K, g_l, ion channels conductance, E_{Na}, E_k , E_l , E_e , E_i ion channels reverse potentials. The definitions of the functions $\alpha_m, \beta_m, \alpha_n, \beta_n, \alpha_h, \beta_m$ are provided in [14]. The necessary condition for an action potential to be produced is that the potential membrane should quickly cross the threshold ($dV / dt \geq V_{th}$).

3. METHODOLOGY

Before we get into the methodology, we will investigate the components involved in the simulation of large-scale SNNs. The major components are outlined in Figure 3 (as described below):

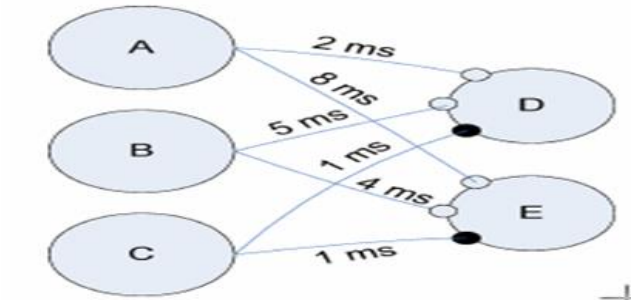

Figure 3: A Simple Illustration of Cortical Networks

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 [3]. In Izhikevich model [3], the neurons are characterized using the dimensionless constants, the membrane potential and recovery variable.

Let

v be the potential membrane neuron

u is the potential recovery

a, b, c, d be the constants of the dimensionless,

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

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

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

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

Figure 4 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 Figure 3.

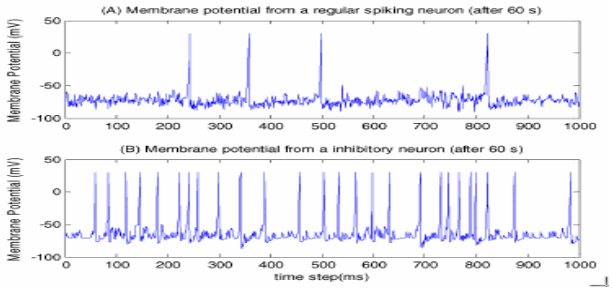

Figure 4: Neuronal Response

Let us consider Figure 4 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 [7][8] 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 in pre synaptic and post synaptic sideways that characterizes the sign and degree of synaptic alterations. This mechanism drives the synaptic connections to contend with the others to gain domination over the firing of post-synaptic neurons.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Authors of work [6][7] suggest that this kind of contention characterizes the stability of firing patterns which can be leveraged for simulating large scale SNNs.

4. THE GPU ARCHITECTURE

Figure 5 demonstrates a generic opinion of the GPU CUDA architecture from the NVIDIA [16]. A GPU, at the hardware level is composed of multiple Streaming Multiprocessors (SMs). As mentioned earlier, GPUs intrinsically are very efficient when it comes to floating-point calculations which are 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.

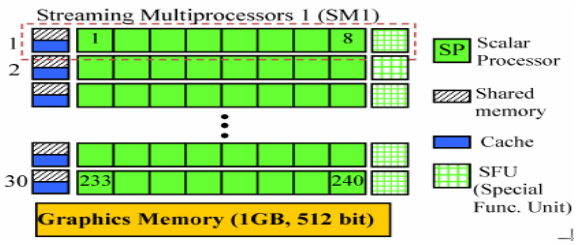

Figure 5: Simplified CUDA Architecture

In our research work, we practise a single NVIDIA GeForce GTX 1060 GPU that contains of 1286 CUDA cores and 10 SMs each operating at 1.27GHz. Every SM with hardware thread schedule which schedules a group of threads for processing. If one of threads in this group has set of instructions that makes an 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 are 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 is stalling condition and to manage the trade-off it is imperative to launch a big amount 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 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. To achieve optimal scaling performance, we use different strategies to ensure minimized memory consumption. This includes sparse connections and by incorporating address event demonstrations format for loading firing data. Other techniques which involve compression can also be used to eliminate redundancy for memory optimization.

(4) **Multi Thread Attributes:** The CUDA GPUs by design selects a warp of 32 threads which are executed by single instructions register. It is important to note here that if all the threads execute the same instruction then the warp 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 instruction 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 afore-mentioned KPIs are proportionately dependent on each other and for a holistic optimization we need to optimize all the KPIs.

5. 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 to achieve maximum performance.

5.1 Parallelism

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

5.1.1 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 elements 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.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

5.1.2 Synaptic [14]-[17] Layer Parallelism

Like 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 connection 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 approved by sequentially the degree of parallelism gets limited. The amount of parallelism achieved in this context is measured by the 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.

5.1.3 Neuronal-Synaptic Layer Parallelism

This is a combination of the afore-mentioned parallelism strategies. The neuronal approach is used whenever a neuron fires and the information must 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 [20] [21].

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.

5.2. 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 limits the performance of the parallelism that can be achieved. The impact of the divergence is determined by the branching condition, that is, 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.

5.3. Representation of Parameters

One of the strategies employed to map the SNN pipelines on to the GPUs is to represent the network parameters in a sparse manner. Consider the SNN with N neuron, M Synaptic connections and the axonal delay *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 Figure 6.

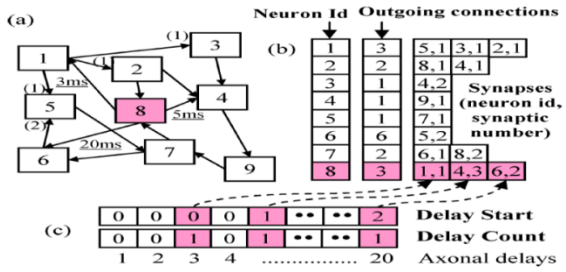

Figure. 6: (a) A Sample Network (b) Neuron Ids and Post-Synaptic Connections (c) Delay Information for Neuron with Id 8.

From Figure 6, it can be noticed that every neuron with a unique id and post synaptic connection is 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 with index *i* specifies the neurons with an axonal delay of *i* ms. Also, in Delay, the i^{th} element is the main synaptic connections with a delay of *i* ms.

5.4 Event Queue

In SNN simulation experiments, generally circular queue data structure is used [10] [17] to store the firing information from the neurons. The subsequent sets of synaptic events which result 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 tables and firing event driven table.

The former indicates the number excitatory neuron that have fired up with 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 less 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. Figure 7 shows a representation of the improved AER format.

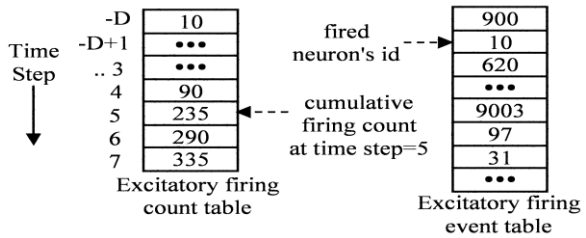

Figure 7: Improved AER Format

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

5.5. GPU Simulations

Figure 8 gives an overview of how SNNs are simulated on the GPU.

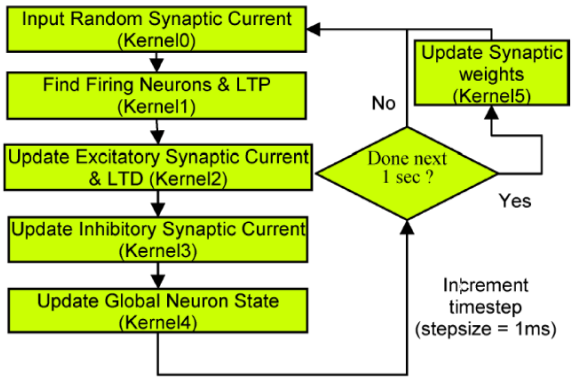

Figure 8: Overview of SNN Simulation on GPU

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 for the concurrent execution. We have chosen the block sizes to be in range of 30-120 for experimentation purposes 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 neuron is accidentally 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 Figure 8.

6. RESULTS

In our study and experiments, the key performance parameters like accuracy and scalability of the techniques and models have been evaluated which leverages the hybrid co-processing (CPU-GPU) model where time-driven events are processed in CPU and the event driven events are processed in GPU (to leverage the parallelism offered by GPU). As discussed earlier, the time driven simulation on CPU achieves high performance at a high precision. However, the shortcoming of such a simulation is scaling it to a large-scale neural network. Figure 9 shows the firing rate using Hogdkin-Huxley model.

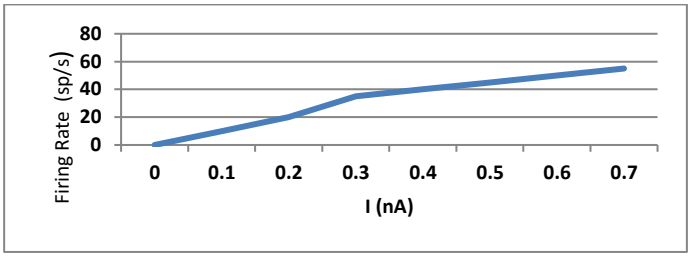

Figure 9: Firing Rate on GPU

The results show that the firing rate on 100, 1000 and 10,000 neurons and we can see the constant behaviour when it fires with I(nA).

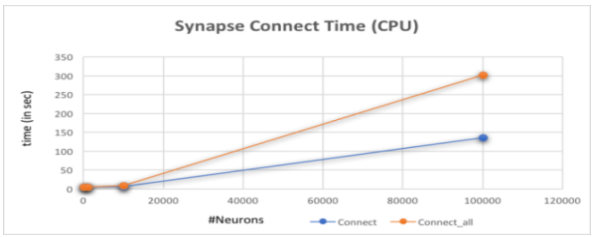

Figure 10: Synapse Connection Time (CPU)

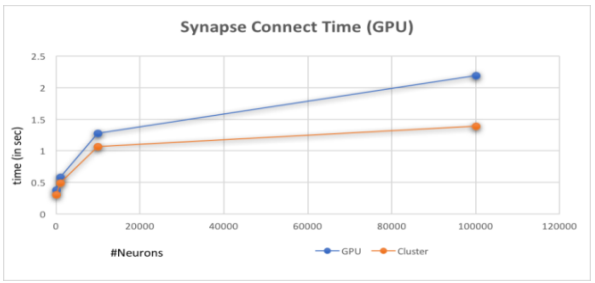

Figure 11: Synapse Connection Time (GPU)

Figure 10 shows the Synapse connection time for 100, 1000, 10000 and 100,000 neurons on an Intel Core i7-5960X CPU @ 3.00GHz. We can see that the time grows exponentially. The blue line is an indicator of the conditional connection of synapse, that is, the synapses are connected based on a condition whereas the orange line indicated unconditional connection of all synapses.

Similarly, Figure 11 shows the synapse connection time on a GPU. However, blue line indicates a standalone GPU and orange line indicate a GPU cluster and we can see that the performance stabilizes as we scale the number of neurons.

As stated, the time-driven (TD) method faces the challenge of scaling shows an exponential behaviour whereas the proposed co-processing (CPU-GPU) does address this challenge fairly well in terms of scaling and timing performance.

6.1 Memory

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

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Table1: Memory Calculations

SNNs Component	Requirement of Memory in Words
information ofNeuron	$5N + \frac{MN}{32}$
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.

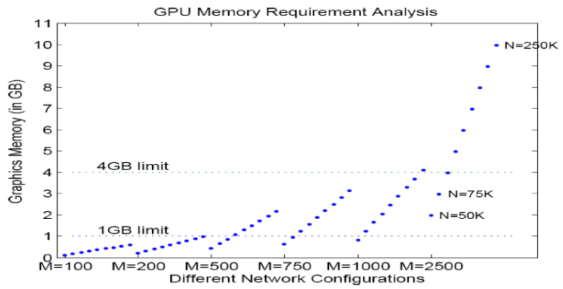

Figure. 12: Memory Requirements for varied SNN Configurations

6.2 Scalability

As shown in Figure 12, we can see the performance for various value of N and M = 200. Also, Figure 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.

6.3 Distributed Synchrony

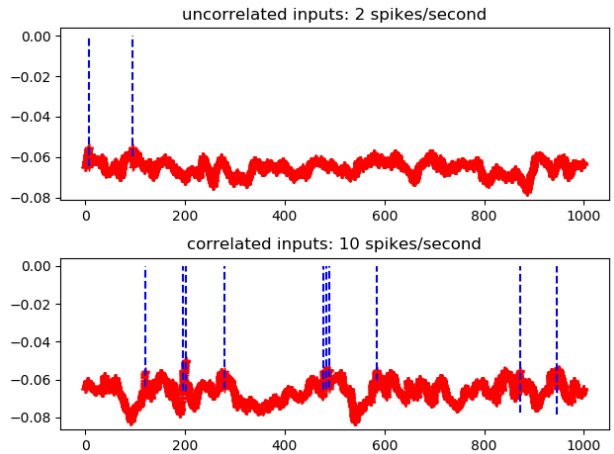

Figure 13: Distributed Synchrony

10000 independent E/I Poisson inputs are injected into a leaky integrate-and-fire neuron. Synchronous events, following an independent Poisson process at 40 Hz, are considered, where 10 E Poisson spikes are randomly shifted to be synchronous at those events. The output firing rate is then significantly higher, showing that the spike timing of less than 1% of the excitatory synapses has an important impact on the post-synaptic firing is as shown in Figure 13.

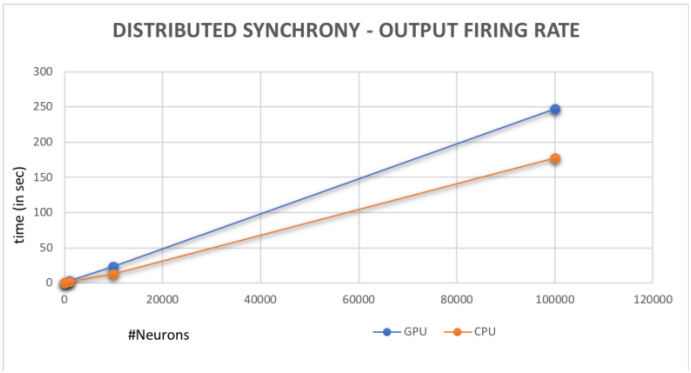

Figure 14: Distributed Synchrony Comparison

It can be seen from Figure 14 that the output firing rate is significantly higher in GPU (standalone) when compared to the CPU.

7. CONCLUSION

From the description of the afore-mentioned work, we can infer that it is imperative to integrate heterogeneous simulation techniques to develop a scalable, high-performing simulator. The next step in the study will be to perform hyper-parameter tuning to study the effect on the spike propagation and the queue management mechanism. The hyper-parameter tuning techniques have opened several 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 CPUs simulation. Though the performance is very limited using memory-bandwidth, we can use GPU cluster in future to simulate even millions of neurons.

REFERENCES

1. Sreenivasa.N S.Balaji (2019) “Hybrid CPU-GPU Co-Processing Scheme for Simulating Spiking Neural Networks” International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8, Issue-4S2 March, 2019 pp. 409-412.
2. Sreenivasa.N S.Balaji(2019) ”Hybrid CPU-GPU Model Based Simulation of Spiking Neural Networks Using a Look-up Table” International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-6S, April 2019 pp 328-333
3. Liwan,Yuling luo, Shuxiang song, Jim harkin, Junxiu liu “Efficient neuron architecture for FPGA-based spiking neural networks” Signals and systems conference(ISSC), IEEE, 2016
4. Govind Narsimhan,Subhrajit Roy, Xuanyou Fong, Kaushik Roy, Chip-Hong Chang, Arindam Basu “A low-voltage, low power STDP synapse implementation using domain-wall magnets for spiking neural networks” ISCAS, IEEE, 2016
5. Francisco Naveros, Niceto R. Luque, Jesús A. Garrido, Richard R. Carrillo, Mancia Anguita, and Eduardo Ro:“A Spiking Neural Simulator Integrating Event-Driven and Time-Driven Computation Schemes Using Parallel CPU-GPU Co-Processing: A Case Study” IEEE transactions on neural networks and learning systems, vol. 26, no. 7, July 2015.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

6. Reza Haghighi and Chien Chern Cheah, “Optical Micromanipulation of Multiple Groups of Cells”, IEEE International Conference on Robotics and Automation (ICRA) Washington State Convention Center Seattle, Washington, 2015.
7. Youssef Chahibi, Sasitharan Balasubramaniam et.al., “Molecular Communication Modeling of Antibody-mediated Drug Delivery Systems”, IEEE Transactions On Biomedical Engineering, Vol. 62, No. 7, July 2015.
8. Jesus A. Garrido1, Richard R. Carrillo2, Niceto R. Luque1, and Eduardo Ros1 J. Cabestany, I. Rojas, and G. Joya (Eds.): “Event and Time Driven Hybrid Simulation of Spiking Neural Networks” IWANN 2011, Part I, LNCS 6691, pp. 554–561, 2011.
9. N. R. Luque, J. A. Garrido, R. R. Carrillo, O. J. D. Coenen and E. Ros, “Cerebellar input configuration toward object model abstraction in manipulation tasks,” IEEE Trans. Neural Networks, vol. 22, no. 8, pp. 1321–1328, Aug. 2011.
10. D. F. Goodman and R. Brette, “The Brian simulator,” Frontiers Neuroscience., vol. 3, no. 2, pp. 192–197, 2009.
11. Gibson Hu, Ying Guo, Rongxin Li, Adaptive Systems Team, Autonomous Systems Laboratory ICT Centre, CSIRO”, A Self-Organizing Nano-Particle Simulator and Its Applications”, 978-0-7695-3166-3/08 $25.00 © 2008 IEEE.
12. R. Brette et al., “Simulation of networks of spiking neurons: A review of tools and strategies,” J. Computer. Neuroscience., vol. 23, no. 3, pp. 349–398, 2007.
13. E. Kandel, J. Schwartz, and T. Jessell, Principles of Neural Science, 4th ed. Amsterdam, Netherlands: Elsevier, 2000.
14. W. Gerstner and W. Kistler, Spiking Neuron Models. Cambridge, U.K.: Cambridge Univ. Press, 2002.
15. [R.Brette et al., “Simulation of networks of spiking neurons: A review of tools and strategies,” J. Comput. Neurosci., vol. 23, no. 3, pp. 349–398,2007.
16. S Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson International Edition, 2009
17. H. Paugam-Moisy, S.Bohte,”Computing with Spiking Neuron Networks”, Handbook of Natural Computing, Springer, Heidelberg, 2009.
18. X.J. Wang, “Neurophysiological and Computational Principles of Cortical Rhythms in Cognition”, Physiol. Rev. vol. 90, pp. 1195-1268,2010
19. E.M.Izhikevich, Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting, The MIT press, 2007.
20. 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.
21. E. M. Izhikevich, "Polychronization: Computation with spikes,"Neural Computation, vol. 18, no. 2, pp. 245-282, February 2006.
22. S. Song, and L.F. Abbott, "Cortical Development and Remapping through Spike Timing-Dependent Plasticity", Neuron, Volume 32(2),339 – 350.
23. T.Simon, A.Delorme, R.Van Rullen, "Spike-based strategies for rapid processing", Neural Netwosrks 14: 715-25,2001.
24. A.Rajagopal, Dharmendra S. Modha: "Anatomy of a cortical simulator", SuperComputing ,2007.
25. E. M. Izhikevich, "Simple model of spiking neurons," Neural Networks, IEEE Transactions on, vol. 14, no. 6, pp. 1569-1572, 2003.
26. Kepecs etal., 2002, "Spike-timing-dependent plasticity: Common themes and divergent vistas", Biological Cybernetics. v87. 446-458.
27. Kayvon Fatahalian and Mike Houston, "A Closer Look at GPUs", Communications ofthe ACM. Vol. 51, No. 10, October 2008.
28. 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.
29. 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.
30. 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.
31. 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.
32. Ernst Niebur, Dean Brettle, "Efficient Simulation of Biological Neural Networks on Massively Parallel Supercomputers with Hypercube Architecture", NIPS 1993, pp: 904-910.
33. 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.
34. F.Bernhard, and R.Keriven, "Spiking neurons on GPUs", International Conference on Computational Science: Workshop on GPGPU, May 2006.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)