International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

A GPU Optimized Technique for Scalable Spiking Neural Network Simulation

Sreenivasa N., S. Balaji

ABSTRACT--- Simulation studies, in general, heavily rely upon the internal variables of the system / entity in the studies. In case of simulation study of the Spiking Neural Networks (SNNs), the major internal system variables are membrane potentials of the neurons and their respective synaptic inputs which demand to be updated at a sub-millisecond resolution. It would be very apt here to note that this requires thousands of updates to simulate one second of an activity per neuron and this factor makes it imperative to have a highly scalable model to derive some inferences from the simulation. Conventionally, high performance CPUs with high degree of multi-threading were leveraged to conduct simulations and derive inferences. With the advances in the hardware, the degree of parallelism has also increased, especially the GPUs have opened a multitude of avenues to perform SNN simulations at scale. In our pervious works [1, 2, 3], we have demonstrated how GPUs can be leveraged to achieve scalability and performance by using hybrid CPU-GPU approach which have improved the performance as compared to multi-threading on high performance CPUs. In this work, we have focused on hyper parameter tuning of some of the key parameters such as delay insensitivity, time step grouping and the active synapse grouping to achieve greater simulation speed of scalable spiking neural networks.

Keywords: SNN, SNN Optimization, delay, time step grouping.

1. INTRODUCTION

Of all the long-term scientific investigations, the SNNs have gained a lot of traction, especially the point-neuron based SNNs. One of the challenges of any simulation study is to get as close as possible to the real systems that are being simulated. Point-neuron based models have been widely used to simulate the spikes generated by the communication between the neurons and synapses. Several researchers [2-5] have presented their work which considerably had better execution speeds compared to others.

Simulation of SNN is unique given the fact that hundreds of thousands of updates need to be simulated to generate one second of activity. The neuro-temporal dynamics and their corresponding synaptic events that are interruptive in nature further adds to the complexity of the simulation study and makes it challenging to optimize. As discussed in a subsequent section, the membrane potential is one of the most important internal variables which is modeled and solved using numerical methods by integrating over the sub-millisecond time step. These spikes are interruptive as they are characterized by the neuron membrane potential which is inherently discontinuous or discrete like the synaptic input whose change is characterized by the pre-synaptic spike as shown in Figure 1. The arrival of the aforementioned events alters the state machine of the neurons and makes them non-Integra table and hence the analytical techniques or any integral based model should be in alignment with the interruption and resemble. Intuitively speaking, this demands the forward-Euler first order numerical integration methods to be employed to simulate such events which makes it domineer to operate on the tiny time intervals.

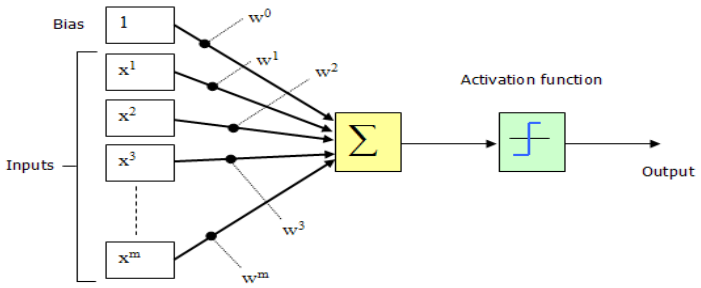

Figure 1: SNN Architecture

Furthermore, the requirement of such large number of updates demands the attention and the number of calculations that are necessary to be considered to improve the performance of the SNN simulation. Often, in many large SNNs, it can be noted that the number of synapses outnumbers the neurons which further increases the complexity of such a system. Though many researchers have leveraged the advanced hardware like the GPUs in the simulation of SNNs as well, some of the bottlenecks in the performance continue to persist. In this work, we propose few performance improvement strategies with respect to some key performance indexes.

2. METHODS

2.1 Prologue

In a general sense, in the realm of the GPUs, the entities/functions that perform data transformation is called a "kernel".

Manuscript published on 30 September 2019
* Correspondence Author
Sreenivasa N*, Research Scholar-Jain University, Dept. of Computer Science & Engineering., Nitte Meenakshi Institute of Technology, P.O. Box 6429, Yelahanka Bengaluru-560064, India, (E-mail meetcna@yahoo.co.in)
S. Balaji, Centre for Incubation, Innovation, Research and Consultancy, Jyothy Institute of Technology, Tataguni, Off Kanakapura Road, Bengaluru-560082, India, (E-mail drsbalaji@gmail.com)

The kernels are responsible to process each data element over all the GPU threads in a parallel fashion. Many such kernels have been used by many researchers in simulation of the SNNs to perform computationally intensive tasks such as membrane potential updates and transformation of neuronal spikes into corresponding synaptic inputs. This creates a scenario which demands the network dynamics to be synchronized in the time domain and the caller (host) must wait until all the kernels associated with the entire specific independent tasks complete their respective execution in order to make it to the next time sub-millisecond time interval [6].

To achieve greater accuracy, it is necessary to simulate the large SNNs for a considerable amount of time which means that we will have to repeatedly run the kernels for a large number of times on the neuronal data set. This leads to operational overhead of managing the life cycle of the kernels which generally is termed as "kernel launch overhead". Every launch of kernel performs only one task of processing spike for a given time interval (time-step). Considering the operational complexity and the overhead of maintaining and managing such kernels, it is only wise to either maximize the throughput of the kernel or lower the launch overhead. Since launching the kernels is repeated over the smaller time intervals, we group these time intervals. We combine the time intervals and group them to improve the efficiency and the throughput of the kernels. Synapses will not have the spikes arriving in a synchronous manner and this is another challenge which the researchers face while simulating SNNs; the threads in the kernels launched at times may not have any computational task to perform and thus being end up underutilized which is detrimental to the accuracy and speed of the simulation experiments. The number of threads that are occupied, that is, the number of threads which are executing a computational task at any given instance of time is called the occupancy of the kernel. The higher the number of threads which are executing a computational task, the higher is the occupancy of the kernel [9].

It is essential to improve the occupancy of the kernels employed to improve the overall performance of the system. Since the spikes arrive asynchronously at any synapse, we will group the synapses that are "active", that is, have the spikes coming into them and name them Active Synapse Group (ASG) so that we can design and launch the kernels in a way that the occupancy is optimal. Reza Haghighi and Chien Chern Cheah [7] introduced the concept of ASG in their work on GeNN. Inspired by their work we have tried to combine the concept of ASG and the time grouping on the GPU to improve the performance. The optimization methods using time interval grouping algorithms are discussed in Figure 2.

(A) Regular Time step centered Network Update

N ← simulation run time (seconds) / Time intervals (seconds)
for n ← 1 to N **do**
    Launch the Kernel
        Update Network State
    Return to Host
end for

(B) Time Interval Grouping for Network Updates

N ← simulation run time (seconds) / Time intervals (seconds)

D ← least network delay (seconds) / Time intervals (seconds)

for n ← 1 to $\frac{N}{D}$ do

    Launch Kernel
    for d ← 1 to D **do**
        Update Network State
    end for
Return to Host
end for

Figure 2: Optimization Using Time Interval Grouping (A) Regular Method without Grouping (B) Time Interval Method with Grouping.

2.2 Time Interval Grouping

As previously discussed, there is always an overhead in managing the kernels which we referred to as kernel launch overheads. This overhead generally causes a delay in any GPU ecosystem and this is proportionate to the number of times the kernel is launched, that is, the number of times the spike is updated in the SNN leading to a considerably important computational cost. In an effort to solve this high computational cost, we have leveraged a feature which several SNNs models have had in the past where a delay has been introduced in the spike arrival from the presynaptic neuron. This delay is termed as axonal delay which we have employed in grouping the time intervals [8,9].

In this method, the least axonal delay is used to modify the network in the time grouped calculations. Figure 2 (A) shows a conventional time interval grouping based update in which the kernel launch at each update updates each network. Figure 2(B) shows the time interval grouping based update wherein a single kernel can be reused multiple times to update the network. It should be noted that we do not perform unnecessary hyper-parameter tuning to optimize beyond its elastic limit since the SNN dynamic will get affected resulting in highly inaccurate results. Considering the minimum axonal delay as the baseline, we can say that for any slice of the simulation which is shorter than this delay, only the spikes generated before that time can affect the neuron and hence by grouping the time intervals shorter than or at least equal to the least axonal delay we can avoid any SNN dynamics from being affected. Hence, all the spiking activity that can affect the different neurons afterward the delay can be composed at the end of the next time interval grouping. As previously discussed, in a GPU based ecosystem, a kernel is launched multiple times in parallel and it must be noted that we do not have much control on the order, these parallelized threads will execute. I.e., if we want to update 'n' neurons from n1,n2,…,n10 we would not know which of these neurons are updated first.

Since these updates do not alter the dynamics of a SNN , we can afford to ignore the need for a synchronized computation method while all the network activity is logged and recorded which is referred to as "Return to Host" in Figure 2 , reason being all the computation on the host is executed in series without the SNN dynamics being affected[10,11]. It now becomes trivial to comprehend that the kernel launch numbers are minimized proportionally equal to the time interval grouping.

Also, by employing this technique we can increase the throughput of the kernel and since computing the update needs to be synchronized after the task, the overall task time will have an upper bound of the time taken by the slowest thread of the kernel. This means that the threads on which the execution involves the reset of the membrane potential and its propagation are the slowest ones. However, such events of spiking for a neuron are not dense in the time domain meaning that for the time interval-based kernel launches, the majority of would have to wait for these sparse phenomena for only a few neurons which would have caused a spike. However, in contrast to these conventional techniques, the grouping of the time intervals results in launching the kernel simultaneously on a large number of such action potential which leads to higher occupancy of the kernel threads and reduces the launch overhead [13].

2.3 Active Synapse Groups

More often than not, in a SNN model, the synapses outnumber the neurons by a large magnitude. Generally, synapses contribute either to conductance of the post-synaptic neurons or the instantaneous current injections. It is trivial to mention that the computation happens only when a spike reaches the synapse after the axonal delay. Monitoring the state machine of all the synaptic connections in the network is simple and straight forward way of determining when to propagate the spike generated from the presynaptic neuron as shown in Figure 3. This causes many threads to be launched which perform no computation owing to the in-activity of the presynaptic neurons, rendering the computation of the simulation, ineffective [15-18].

(A) Naive Synapse Updating

S ⟵ # Synapses

Launch Kernel

for s ⟵ 1 to S **do**

if s spiked **then**

Update upcoming synaptic inputs to Post-Synaptic Neurons

End if

end for

Return to Host

(B) Active Synapse Grouping

N ← # Neurons

ActiveSynapses ← [ ]

Launch Kernel

for n ← 1 to N **do**

if Neuron(n) spiked **then**

ActiveSynapse [] ← Add(n)

end if

end for

Return to Host

Launch Kernel

$\boldsymbol{for}\ s_{active} \in ActiveSynapses\ \boldsymbol{do}$

PostSynapticNeuron [] ← Add(s_{active})

end for

Return to Host.

Figure 3: The synaptic updates Optimizations (A) Conventional Method (B) ASG (Active Synapses Grouping)

In the aforementioned approach, the kernel loops through the neurons to check for any spike emissions. In case of a spike being emitted the emitting connections are flagged active and added to the active group collection. After the successful completion of the first loop, another kernel is launched to update all the neurons that have been set to active in the previous step thus reducing the number of synaptic update computations [19,20].

2.4 Delay Insensitivity

Ideally, we need a simulator which is inert to the axonal delays as it opens a lot of research avenues particularly in the models that leverage the axonal delay. In [22,23] authors have demonstrated that SNN models can be used to characterize a wide range of entities like auditory processing, sound localization etc. which leverage the concept of axonal delays. To achieve the delay insensitivity every neuron will make use of a circular buffer which stores the data of the synaptic updates coming into the synapses. The network neurons will repeat on these buffers to get the synaptic modifications. At every time interval, the next neuron shifts ahead by one position and reads the synaptic input corresponding to the current interval [29,32].

Since these buffers are circular, the time complexity is O(1) meaning it takes a constant time to add to buffer, remove from buffer and update and help to conserve the memory. Now, for optimizing this, if we define the length to be equal to the maximum delay then every neuron that is currently querying for the updates in the current time interval can find all the inputs needed at the corresponding delay of that synapse [32-35]. It is trivial to note here that there is no computational overhead and hence the speed of the simulation is not affected even after adding the delay parameters in the transmission of the synapses. However, for the required speed improvement we must trade the space complexity for the sake of time complexity.

3. RESULTS AND DISCUSSION

We have simulated about 10,000 LIF (Leaky Integrate and Fire) neurons with (2500 inhibitory and 7500 excitatory) approximately 2 percent random connectivity. The network neurons are supplied with a 200pA input current to excite them. Effect of the various optimizations on GPU based SNN is as shown in Figure 4.

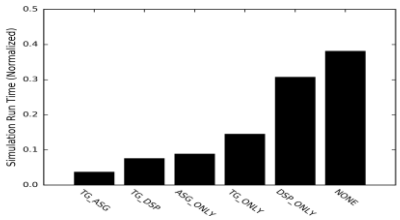

Figure 4: Effect of the Various Optimizations on GPU Based SNN

Vogels-Abbott benchmark is used for benchmarking the simulation time and the results are compared with scenarios without any optimizations, that is, the ones which use conventional algorithms as explained in Figure 2(A) and Figure 3 (A) which are based on brute force approach. This condition is then compared with the ASG and Dynamic Synapse Parallelism (DSP) technique. DSP is a technique where the individual threads in a GPU can launch more kernels, the feature available on the recent NVIDIA GPUs, which was developed by [24,25].

It becomes trivial form Figure 4 that ASG proves to be faster than the DSP and the conventional brute force approach under the optimized environment. Time grouping also shows a considerable speed up in both the scenarios, that is, with and without the optimizations. We have used identical networks for simulation and found that TG has more speed up than others because of a) minimized launch overhead and b) increased kernel throughput.

3.1 Multithreaded CPU Performance

In [28,29], the researchers have presented the benchmark in a multithreaded environment in Auryn, which is the most preferred simulator in multithreaded ecosystem and thus we see a plausible simulator for comparison. Multithreaded CPU vs single GPU graph is as shown in Figure 5.

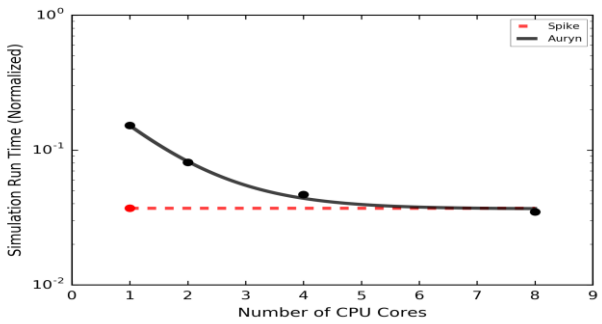

Figure 5: Multithreaded CPU vs Single GPU

From Figure 5, it can be noted that, to come closer to the performance of a single GPU based simulator, the multithreaded simulator needs at least 8 cores of CPU. We can now say with conviction that GPU based simulations lead any benchmark, any performance parameter. Since we can produce custom kernels for a specific simulation we can afford to set some parameters locally to the kernel and hence achieve better performance and also it opens up new avenues to perform more hyper-parameter tuning based research experiments.

4. CONCLUSION

A range of simulation models have been proposed each focusing on some key performance indices and a combination of them, that is, speed, hardware, model definition etc. In this study, we have focused on parameters like time interval grouping and conditional synaptic grouping (active and in-active). Our results show a significant speed gain by leveraging GPU when compared with the multithreaded simulation environment. We have noticed a significantly large improvement in simulation time on GPU when compared to multithreaded systems and we, therefore, are convinced to state that GPUs as a de-facto platform for simulation of SNNs of this nature.

REFERENCES

1. Sreenivasa N., S.Balaji (2019) "Empirical Analysis of Spiking Neural Networks Using CPU, GPU and GPU Clusters" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8, Issue-5, June 2019 PP. 1942-1950 Impact factor: 5.15 Unpaid Scopus Indexed Journal
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 Impact factor: 5.15 Scopus Indexed Journal
3. 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. Impact factor: 5.15 Scopus Indexed Journal
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. D. F. Goodman and R. Brette, "The Brian simulator," Frontiers Neuroscience., vol. 3, no. 2, pp. 192–197, 2009.
12. 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.
13. 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.
14. E. Kandel, J. Schwartz, and T. Jessell, Principles of Neural Science, 4th ed. Amsterdam, Netherlands: Elsevier, 2000.
15. W. Gerstner and W. Kistler, Spiking Neuron Models. Cambridge, U.K.: Cambridge Univ. Press, 2002.
16. 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.
17. Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spikingneurons. J. Comput. Neurosci. 8, 183–208.
18. Eguchi, A., Isbister, J. B., Ahmad, N., and Stringer,S. (2018). The emergence of polychronization and feature binding in a spiking neural network model of the primate ventral visual system. Psychol. Rev. 125, 545–571.
19. Erfanian Saeedi, N., Blamey, P. J., Burkitt, A. N., and Grayden, D. B. (2016). Learning pitch with STDP: A computational model of place and temporal pitch perception using spiking neural networks. PLoS Comput. Biol. 12, e1004860.
20. Goodman, D. F. M. and Brette, R. (2010). Learning to localise sounds with spiking neural tworks. In Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1 (USA: Curran Associates Inc.), NIPS'10, 784–792.

21. Kasap, B. and van Opstal, A. J. (2018). Dynamic parallelism for synaptic updating in PUaccelerated spiking neural network simulations. Neurocomputing 302, 55–65.
22. Linssen, C., Lepperød, M. E., Mitchell, J., Pronold, J., Eppler, J. M., Keup, C., et al. (2018). Nest 2.16.0. doi:10.5281/zenodo.1400175.
23. Paugam-Moisy, H., Martinez, R., and Bengio, S. (2008). Delay learning and polychronization for reservoir computing. Neurocomputing 71, 1143– 1158.
24. Stimberg, M., Goodman, D. F. M., Benichoux, V., and Brette, R. (2014). Equation-oriented specification of neural models for simulations. Front. Neuroinform. 8, 6.
25. Vitay, J., Dinkelbach, H. U¨ ., and Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. Front. Neuroinform. 9, 19.
26. Vogels, T. P. and Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. J. Neurosci. 25, 10786–10795.
27. Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework foraccelerated brain simulations. Sci. Rep. 6, 18854.
28. Zenke, F. and Gerstner, W. (2014). Limits to highspeed simulations of spiking neural networks using general-purpose computers. Front.
29. S Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson International Edition, 2009.
30. H. Paugam-Moisy, S.Bohte,"Computing with Spiking Neuron Networks", Handbook of Natural Computing, Springer, Heidelberg, 2009.
31. X.J. Wang, "Neurophysiological and Computational Principles of Cortical Rhythms in Cognition", Physiol. Rev. vol. 90, pp. 1195-1268, 2010.
32. E.M.Izhikevich, Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting, The MIT press, 2007.
33. 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.
34. E. M. Izhikevich, "Polychronization: Computation with spikes,"Neural Computation, vol. 18, no. 2, pp. 245-282, February 2006.
35. S. Song, and L.F. Abbott, "Cortical Development and Remapping through Spike Timing-Dependent Plasticity", Neuron, Volume 32(2), 339 – 350.