A Framework for parallel Event Driven Simulation of Large Spiking Neural Networks

Michiel D'Haene

Supervisor(s): Dirk Stroobandt

Abstract— **The simulation of spiking neural networks (SNNs) is still a very time consuming task. For realtime applications, this limits simulations to rather unrealistic and computationally weak small or medium sized networks. SNNs consist of neurons that communicate using discrete spikes which makes them suitable for an event-driven approach. We can increase the simulation speed even further by exploiting the parallel nature of SNNs in hardware implementations. However, when using parallel processing units, one has to ensure that events are processed in the correct time order which introduces the need for synchronization mechanisms. Although they are well known, they have yet to be studied for SNNs. We present a framework for Parallel and Distributed Event Simulation (PDES) for SNNs. It allows us to test different synchronization mechanisms using networks with programmable delays and any number of processor units. This will allow us to select and optimize the best synchronization mechanisms.**

Keywords—**Spiking Neural Network, Event Simulation, Synchronization**

I. Introduction

SPIKING neural networks (SNNs) consists of biologically inspired neurons that communicate by using spikes. Because the exact timing of the spikes is considered, SNNs are able to handle temporal problems more efficiently (for example speech recognition [10]) and have more computational power than artificial neural networks [5] which model the average firing rate of neurons as analog inputs. Furthermore, they communicate through discrete spikes instead of analog values which significantly reduces the communication cost. This makes them more suitable for e.g. hardware implementations [8].

The behaviour of a spiking neuron can be represented by an internal membrane potential function which is influenced by incoming spikes. When the potential of the membrane reaches a certain value, the membrane potential will be reset to a lower value and a spike is emitted. It is important to note that each neuron operates independently from the other, except when a spike is transmitted between neurons.

An obvious way of implementing SNNs is a direct placement of the neurons into hardware, which benefits of the inherent parallel nature of spiking neural networks and allows extremely fast simulations (orders of magnitude faster than realtime) [9]. However due to the typical low activity of a SNN, these implementations are very space inefficient and are limited in size by the amount of hardware available on the chip. Therefore, in practice, large SNNs are simulated using more conventional architectures.

There are essentially two ways of simulating SNNs on conventional architectures: time-step based and event based simulation. The first one divides the simulation into fixed time-steps. At each time-step, the complete network is evaluated and the new state of each neuron is calculated. The precision of the simulation depends on the time-step size, which also affects the simulation time. Despite its simplicity, the asynchronous nature of the spikes requires small time-steps ($\leq$1ms) in order to achieve accurate simulation results.

Usually we are only interested in the external behaviour of a neuron (spike generation) due to incoming spikes, and not in the internal membrane changes. Event based simulation takes advantage of this. Instead of evaluating the whole network on regular time intervals, the membrane potential of each neuron is evaluated only when necessary, i.e. when a neuron receives a spike, or when it generates a firing event. Because the frequency of spike generation is typical much lower than the 1ms interval of time-step driven simulations, this approach results in more efficient simulations. Also the accuracy is much higher, in many cases analytically correct results can be obtained. Our tests show that a speedup of 60 or more compared to advanced time-step driven simulators (for example CSIM) is possible [2].

An event-simulator has to keep track of all events in the system and execute events in the correct time order. This can be done with a queue that keeps all generated events in chronological order. The simulator takes out the event with the smallest timestamp from this event queue, processes it and adds new events to the queue if necessary. Then it takes the next event with the lowest timestamp, etc.

One disadvantage of the event-driven approach is that incoming and outgoing pulses must be considered as discrete events. However, not all neural models fulfill this condition. Therefore some biologically realistic models like the Hodgkin-Huxley model can not be simulated in an event-driven manner. Another difficulty is that if a neuron will fire, the simulator has to schedule the fire-timestamp as a new event in the queue which often requires some iterative calculations. As long as this timestamp is not reached, new incoming spikes can change this timestamp causing a recalculation of the fire-timestamp. It can turn out to be very computationally expensive for more complex models. Some techniques to handle this problem are discussed in [2].

II. Parallel event based simulation

The event driven principle described above is obviously a sequential process. Events are taken one by one from the event-queue and processed. After each execution, newly generated events are inserted into the queue before processing the next event (Fig. 1a). Although this allows much faster simulations than time-step based simulation, it is still far too slow to be interesting for many applications. For example, realtime simulations on modern architectures are limited to networks of order of magnitude 10.000 (simple) neurons [1].

M. D'Haene is with the Electronics and Information Systems Department, Ghent University (UGent), Gent, Belgium. E-mail: Michiel.DHaene@UGent.be .

An obvious way to accelerate this sequential process is the use of more processing units in parallel or through pipelining as can be seen in Fig. 1b and c. This is for example used in the SPINN Emulation Engine (SEE) [4] or in the SpikeFORCE project [7]. Unfortunately the intense memory interaction of event simulation creates an important memory bottleneck. Also the inherent parallelism of SNNs remains unused: SNNs are composed of a huge number of neurons that communicate only through spikes. Otherwise they are independent and can thus be calculated in parallel.

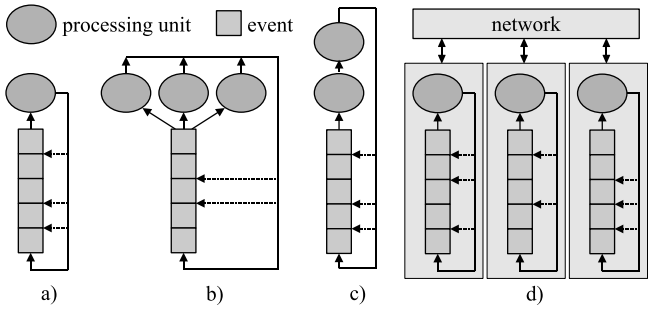

Fig. 1. a) Single queue with 1 processing element, b) Single queue with multiple processing units, c) Single queue with pipelining and d) Multiple queues.

Parallel discrete event simulation (PDES) refers to the execution of a single discrete event simulation program on concurrent processing units, often called logical processes (LPs). Each processing unit has its own event queue and processing element (Fig. 1d). Communication between the processing units occurs through messages. In asynchronous applications like SNNs, the probability that events coincide is very small. Therefore we have to concurrently process events with a different timestamp. An important requirement of event based simulation however is that if two events influence each other, they have to be executed in the correct chronological order (to avoid causality errors). As can be seen in Fig. 2, it is hard to determine which events can be executed in parallel because the mutual influence of two events depends on the input and the internal state of the neurons, which is very unpredictable [3]. To guarantee that events are executed

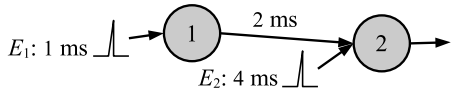

Fig. 2. Data dependency: event E_1 will only influence E_2 when it causes neuron 1 to fire, which is dependent of the state of neuron 1.

in correct order, a synchronization mechanism must be used that keep the different subsimulators synchronized. There exist several techniques that deal with these synchronization problems. They can be roughly divided in conservative, optimistic and adaptive synchronization. Each synchronization method has its advantages and disadvantages. The little research that has already be done with PDES for SNNs uses conservative methods, mainly for its simplicity. However there are some indications that other methods might perform better, especially when the number of LPs grows. Still these techniques have yet to be studied for SNNs.

III. The framework

To allow us to investigate different synchronization mechanisms under several circumstances, we have built a general framework for PDES of SNNs. We used SystemC as the platform for the concurrent simulation of several LPs. It allows us to control the simulated execution time of each LP, independently from the actual execution time. We also wrote a communication class which allows LPs to communicate with each other with a programmable propagation delay of messages between each LP.

Another issue when distributing the network between several LPs is the partitioning of the network. This can be done manually, random or using hMETIS, a well known hypergraph partitioning algorithm. Different synchronization mechanisms can be easily implemented within the framework using the well chosen functions that control the simulation process. The base simulator is based on MVASpike, a general event based simulator for SNNs by Olivier Rochel [6].

An important aspect is hardware-friendly design: in a later stadium, the simulator will be implemented in digital hardware (FPGA) in order to benefit optimally of the inherent parallelism of SNNs.

IV. Conclusions and future work

In this paper, we have shown that event driven simulation can efficiently simulate a broad class of SNNs. However, the inherent parallel nature of SNNs remains still unused. Therefore, further speedup can be achieved with a parallel approach. Unfortunately this involves difficult synchronisation issues. We have built a general framework for testing synchonization mechanisms under different circumstances. It will teach us which techniques are best suited for our applications, i.e. fast simulation of large-scale neural networks.

References

[1] M. D'Haene. Parallelle event-gebaseerde simulatietechnieken en hun toepassing binnen gepulste neurale netwerken. Technical report, Universiteit Gent, 2005.

[2] M. D'Haene, B. Schrauwen, and D. Stroobandt. Accelerating event based simulation for multi-synapse spiking neural networks. In *Proceedings of the 16th International Conference on Artificial Neural Networks*, volume 4131, pages 760–769, Athens, 9 2006. Springer Berlin / Heidelberg.

[3] R.M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[4] H.H. Hellmich, M. Geike, P. Griep, P. Mahr, M. Rafanelli, and H. Klar. Emulation engine for spiking neurons and adaptive synaptic weights. In *IJCNN*, pages 3261–3266, 2005.

[5] W. Maass. Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1):1–40, 1996.

[6] O. Rochel and D. Martinez. An event-driven framework for the simulation of networks of spiking neurons. In *Proceedings of the 11th European Symposium on Artifical Neural Networks, ESANN 2003*, pages 295–300, 2003.

[7] E. Ros, E.M. Ortigosa, R. Agís, R. Carrillo, A. Prieto, and M. Arnold. Spiking neurons computing platform. In *Proceedings of the 8th International Work-Conference on Artificial neural Networks, IWANN 2005*, pages 471–478, 2005.

[8] B. Schrauwen. Embedded spiking neural networks. In *Doctoraatssymposium Faculteit Toegepaste Wetenschappen*, pages on CD–ROM. Universiteit Gent, Gent, December 2002.

[9] Benjamin Schrauwen and Jan Van Campenhout. Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic. In *Proceedings of the European Symposium on Neural Networks*, pages 623–628, 2006.

[10] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.