Accelerating Event Based Simulation for Multi-synapse Spiking Neural Networks

Michiel D'Haene*, Benjamin Schrauwen**, and Dirk Stroobandt

Ghent University, Department of Electronics and Information Systems, Ghent, Belgium
{Michiel.DHaene, Benjamin.Schrauwen, Dirk Stroobandt}@ugent.be
http://www.elis.ugent.be/SNN

Abstract. The simulation of large spiking neural networks (SNN) is still a very time consuming task. Therefore most simulations are limited to rather unrealistic small or medium sized networks (typically hundreds of neurons). In this paper, some methods for the fast simulation of large SNN are discussed. Our results equally amongst others show that event based simulation is an efficient way of simulating SNN, although not all neuron models are suited for an event based approach. We compare some models and discuss several techniques for accelerating the simulation of more complex models. Finally we present an algorithm that is able to handle multi-synapse models efficiently.

1 Introduction

Despite the ever increasing computational power in computer systems, there is still a huge demand for more computational power, especially in the field of spiking neural networks (SNN). Spiking neurons are biologically inspired neurons that communicate by using spikes. Because here the timing of the spikes is considered, SNN are able to handle temporal problems more efficiently (for example speech recognition [18]) and have more computational power than artificial neural networks [9] which use the average firing rate of neurons as inputs. Furthermore, they communicate through discrete spikes instead of analog values which significantly reduces the communication costs between neurons. This makes them particularly better suited for hardware implementations [13][14].

The behaviour of a spiking neuron can be represented by an internal membrane potential which is influenced by incoming spikes. When the potential of the membrane reaches a certain threshold 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, except when a spike is communicated between neurons.

An obvious way of implementing SNN in hardware is a one to one placement of the neurons into physical components. This approach benefits from the inherent parallel nature of spiking neural networks and allows extremely fast simulations (orders of

* Michiel D'Haene is sponsored by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

** Benjamin Schrauwen is funded by FWO Flanders project G.0317.05

S. Kollias et al. (Eds.): ICANN 2006, Part I, LNCS 4131, pp. 760–769, 2006.

magnitude faster than realtime). Unfortunately it has to deal with some important drawbacks. First, the size of the networks is limited by the amount of hardware available on the chip. In current FPGA's[1], one can implement a couple of thousand simplified spiking neurons in a direct manner [15]. By combining several chips, one should be able to create larger networks, however this is a very cost inefficient method. Another important drawback of this approach lies in the low activity, which is for a typical spiking neural network less than 1% ([5]). This means that generally most of the synapses and neurons are inactive (i.e. potential on its restpotential) and thus occupy costly space in hardware. Direct hardware implementations are thus very space inefficient.

Therefore, in practice, large SNN are simulated using more conventional architectures. There are essentially two ways of simulating SNN: time-step driven 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 size of each time-step, which also affects the simulation time. Although this is a very simple approach, the asynchronous nature of the spikes requires small time-steps ($\leq$ 1 ms) in order to achieve accurate simulation results [16].

Usually we are only interested in the external behaviour of a neuron (i.e. emitted spikes) due to incoming spikes instead of 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 will fire.

In the next section, we compare time-step driven simulation with event based simulation using two accurate simulation environments. In Section 3 we briefly discuss different models of spiking neurons and show why some of them can be simulated efficiently in an event driven manner while very biological realistic models are much more difficult to simulate. We show the results of some accelaration techniques for more complex models. Finally, because existing simulators do not support multi-synapse models efficiently, we present an algorithm that handle these models efficiently.

2 Event Based Simulation Versus Time-Step Driven Simulation

In order to compare event based simulation with time-step driven simulation, we created a random network consisting of 500 neurons with an interconnection fraction of 0.1 (i.e. each neuron has on average 50 input connections). Input spikes are generated through one input neuron. The neuron model used is developed by Olaf Booij (personal communication). It is a special case of a leaky integrate and fire neuron with exponential synapses where $\tau_m = 2\tau_s$ (this will be explained in Section 3.1).

For the time-step driven simulator, we used CSIM[2] which is a well known open source C++ simulator, optimized for speed by calculating only the active synapses. The resolution (time-step) was set to 0.1 ms.

The event-based simulator used is ESSpiNN, a simulator developed at our research group. The core of this simulator is based on MVASpike, a general event based C++

[1] Field Programmable Gate Array.
[2] http://www.lsm.tugraz.at/csim

simulator for SNN from Olivier Rochel [12]. We have adapted this simulator to our goals, and extended it in order to allow more general neuron models (e.g. SRM_0) to be simulated. Besides a broad range of neuron models, it can simulate different types of (delayed) connections such as STDP, dynamic synapses, etc. The time resolution for the neuron model used in this example is only limited by the resolution of the double precision floating point numbers, which results in very accurate simulations compared to time-step driven methods.

In Table 1 we measured the execution time of both simulators for different numbers of input spikes (on the same platform, i.e. AMD Athlon 64 3400+). We can see that the average speedup of the event driven simulator is 60 times for a relatively high neuron activity (on average 100 spikes per neuron per second). For 100 and 1000 input spikes, we used a simulation time of 1s, which means that the incoming spike activity is a factor 10 higher for the second case. When we look at CSIM, we can see that despite the 10 times lower activity, its execution speed is only a factor 3.75 faster. This is due to its time-step driven nature: time-step based simulation scales in the first place with the simulated time, while event-based simulation scales mainly with the number of spikes.

Table 1. Comparison between CSIM and ESSpiNN for a network of 500 neurons, interconnection fraction of 0.1 and 1 input neuron. The neuron model is Booij's integrate and fire neuron where $\tau_m = 2\tau_s$.

Number of spikes	Simulated time	Execution time CSIM	Execution time ESSpiNN
100	1 sec	14.6 sec	0.10 sec
1 000	1 sec	55.0 sec	0.80 sec
10 000	10 sec	451 sec	7.13 sec
100 000	100 sec	/	72.8 sec

3 Event Simulation of Different Spiking Neuron Models

In event based simulation, a new calculation will be performed only when an event occurs. An event-simulator has to keep track of all events in the system. This can be done with a queue that keeps all generated events in chronological order (the event-queue). The simulator takes the event with the smallest time stamp from this event queue, processes it and adds new events to the queue if necessary. Then it takes the next event with the smallest time stamp, etc.

An obvious requirement for event simulation of SNN is that incoming and outgoing pulses can be considered as discrete events. However, not all neural models fulfill this condition. Some models are very easy to simulate in an event driven fashion, while others are much harder. Below, we discuss some important models and their applicability to event simulation. Then we give some results measured on our event simulator for different neuron models.

3.1 Some Common Neuron Models

Hodgkin Huxley model. The model of Hodgkin and Huxley [7] is a very good imitation of a certain type of biological neuron (Fig. 1a). It consists of a collection of

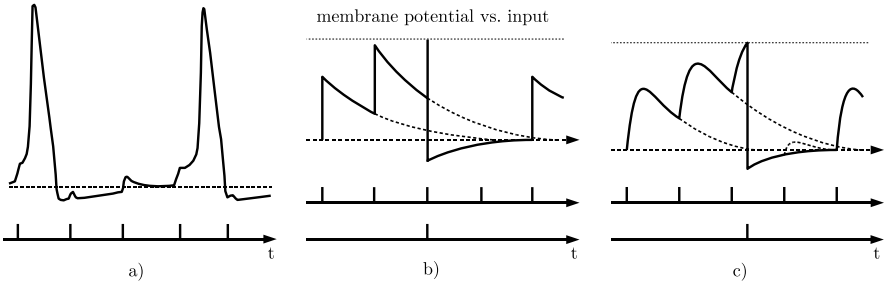

Fig. 1. Some popular spiking neural models a) example of a Hodgkin-Huxley-model, b) the Leaky IF-neuron and c) an approximation with SRM ($0^t h$ order) model

differential equations that model the internal processing. It is an important model for the detailed study of the biological behaviour of neurons that deals with ion channels, different types of synapses and geometry of individual neurons. This model is in essence a continuous model, which makes it very difficult to simulate in an event driven fashion without making certain concessions [8]. However it is an important reference model for more simple models.

(Leaky) Integrate and Fire model (IF model). IF models are one of the best known examples of the so-called threshold model neurons, which means that a pulse is generated as soon as the membrane potential reaches a certain threshold. Pulses are stereotypical events which are completely characterized by their firing time stamp. After firing, the neuron optionally has a refractory period during which it stays inactive for a certain amount of time. In Fig. 1b, an example of a Leaky IF model is shown (without a synapse model). Because firing happens only when an incoming spike arrives, the event based simulation of this model is simple. However, this property also drastically limits the expression power of this model [10].

Spike Response Model (SRM). The SRM [4] can be seen as a generalization of the Leaky IF model. One of the differences however is that the membrane potential is expressed as a function of the time passed since the last fire event, instead of a function of itself. A special case of the SRM is the 0^{th} order SRM or SRM_0. All incoming pulses now have the same shape, independent of the time since the last incoming event and the state of the membrane. A simple example is shown in Fig. 1c. The model shown is an exponential Leaky IF model with exponential synapses. For most applications, the SRM_0 is still sufficient and has been used for example for the analysis of the computational power of neurons [10], studies of collective phenomena in local coupled networks [4] and in the Liquid State Machine (LSM) [6].

There exist a number of event based simulators specially built for a specific type of IF neurons (most without synapse model) and networks. These simulators allow fast simulations but –given the simplicity of the neuron model– they have limited applicabilities and are often built with a specific application in mind. A totally different approach is to try to create a biological very realistic simulator [8]. However the aim of our work is not to create a biological very realistic simulator, but to allow efficient emulations of huge

networks of powerful spiking neurons. The LIF with a synapse model is a good starting point for such a simulator, but it involves some difficulties when we try to simulate it in an event-based manner. In the next subsection, we briefly explain these difficulties and show some techniques and optimisations to solve them efficiently.

3.2 A Simple $\mathbf{SRM_0}$ Neuron

To show the problems of simulating a more general IF model, we consider the exponential Leaky IF model with exponential synapses (Fig. 1c). The postsynaptic potential $u_i(t)$ of neuron i with n inputs at time t can be written as [4]

$$u_i(t) = \sum_f \eta_i\left(t - t_i^{(f)}\right) + \sum_j w_{ij} \sum_f \epsilon_{i,j}\left(t - t_i^{(f)}\right) \tag{1}$$

with kernels (after convolution of $\epsilon_0(s)$)

$$\eta_i(s) = -(\theta - u_{rest}) \exp\left(-\frac{s}{\tau_{m,i}}\right) \mathcal{H}(s) \tag{2}$$

$$\epsilon_{i,j}(s) = \frac{1}{\max_{i,j}} \left[\exp\left(-\frac{s}{\tau_{m,i}}\right) - \exp\left(-\frac{s}{\tau_{s,i,j}}\right)\right] \mathcal{H}(s) \tag{3}$$

with θ the threshold, u_{rest} the restpotential and τ_m and $\tau_{s,j}$ the decay constants of the membrane and the synapse. $\max_{i,j}$ is a rescaling constant to assure that a weight of 1 generates a pulse of the same height. To schedule the next update-time of a neuron, the event-simulator must be able to predict the next fire-time stamp of a neuron, i.e. the time stamp when ϵ_0 reaches the threshold θ.

Therefore equation (1) with $u_i(t) = \theta$ must be solved for t. However for non-natural values of τ_m and $\tau_{s,j}$ this can not be solved analytically. There exists different solutions for this problem which we divide roughly into tree classes: using a restricted model which can be solved very efficiently, using look-up tables, or using iterative techniques to approximate the fire-time stamp.

A good example of the first solution is the neuron model developed by Olaf Booij (personal communication). He assumes that each input synapse has the same τ_s, and that $\tau_m = 2\tau_s$. Now, the solution for t is reduced to a quadratic equation which can be solved analytically very quickly.

The use of lookup tables is a well known method to estimate the fire-time stamp which offers high speed with (depending on the size of the table) sufficient accuracy [1]. Some researchers even replaced all calculations with precalculated lookup tables [2]. An important drawback of lookup tables however is the memory size that grows more than exponentially with the desired accuracy. Also, separate lookup tables are required for each τ_s. Therefore, most researchers limit their models to a single synapse model.

An example of the last type of solution is developed by Makino [11]. His simulator is able to estimate the fire time stamp of an arbitrary continuous membrane function by dividing the function into linear envelopes where each envelope containes at most one threshold crossing. Inside each envelope, a Newton-Rhapson based method is used to estimate the fire-time stamp.

It is clear that most techniques are developed to be used with one or a very restricted number of synapse constants. When extending these techniques to neurons with several synaptic time constants (multi-synapse neurons, in the most common case, each input has its own synaptic time constant), they have to deal with the calculation of each synapse for each update of their state, which quickly decreases their efficienty. Therefore, we developed an efficient technique that can handle several synaptic time constants, which we will describe briefly below.

3.3 Efficient Processing of Multi-synapse Neurons

An obvious optimisation for the simulation of multi-synapse models that has also been used in e.g. CSIM is to separate inactive synapses from active synapses and to consider only the active synapses of a neuron in the computations. Due to the typical low activity of spiking neural networks, this results already in a significant speedup (a factor 2).

Another optimisation follows from the observation that a neuron generally has to receive several spikes before it will emit a spike itself. However, estimating if a the membrane potential will reach the threshold, is a time consuming process itself when using several synaptic constants.

Therefore, we developed a simple but fast membrane-value estimation function. Whenever the neuron receives a spike, we consider the (updated) maximum influence of the current synapse on the membrane potential by adding it to the total approximated membrane maximum (Fig. 2). When a spike enters, we only update the synapse potential of the input that receives the spike. When there is a possibility for the membrane potential to reach its threshold (i.e. the approximated membrane maximum reaches the threshold), we start updating the total membrane potential by calculating the exact potential of each synapse at the current time stamp.

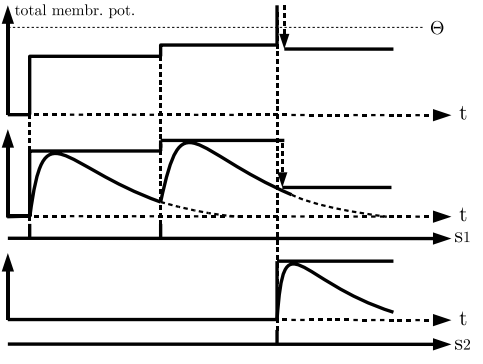

Fig. 2. Maximum approximation of the membrane potential. On time 3, the approximated maximum reaches the threshold. After updating synapse 2, we see already that the threshold will not be reached.

In many cases, after updating just a couple of synapses, the approximated membrane potential will drop below the threshold, so we can stop updating and wait for the next

spike to enter. By updating the synapses with the oldest update-time first, we have the highest likelihood that only a couple of updates are sufficient to see that the threshold will not be reached because the maximum drops below the threshold. Therefore, we keep the synapses ordered by their update-time. We implemented this in an efficient way by placing them in a circular doubly linked list. Due to the algorithm, when the oldest synapse needs to be updated (to the current time stamp), only the start-pointer of the list needs to be updated, which involves no sorting actions (Fig. 3). As soon as the influence of a synapse on the membrane potential becomes negligable, we remove the synapse from the active synapse list.

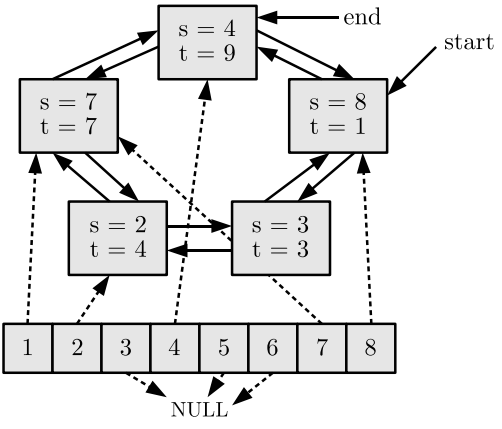

Fig. 3. A circular doubly linked list is used to keep synapses ordered by their update time, thus eliminating a sorting action. Also an index structure allows fast random access of synapses.

When the threshold is still reached after updating all synapses, we apply a Newton-Rhapson (NR) method to estimate the exact fire-time stamp. Measurements show that we have a high accuracy after only a few NR iteration steps. A drawback of our approach however is that for very high activity (spike-frequency $\sim$kHz, which is very unrealistic), the maximum membrane approximation becomes almost useless and introduces an overhead to the simulation.

3.4 Results

In order to be able to compare the simulation speed of our model with other models, we used Booij's restrictions to the multi-synapse neurons (i.e. each synapse $\tau_{s,i}$ was set to $\tau_{m,i}/2$) and compared the execution-times for several implementations. We used multi-synapse neurons with a separate synapse constant for each input and neglected of course the fact that the synapses can be actually combined to one synapse (because they all have the same time constant, and can thus be calculated more efficiently).

In Figure 4 (top), we compare the execution time of several implementations of the general LIF model for different network activities. The first optimization (calculating

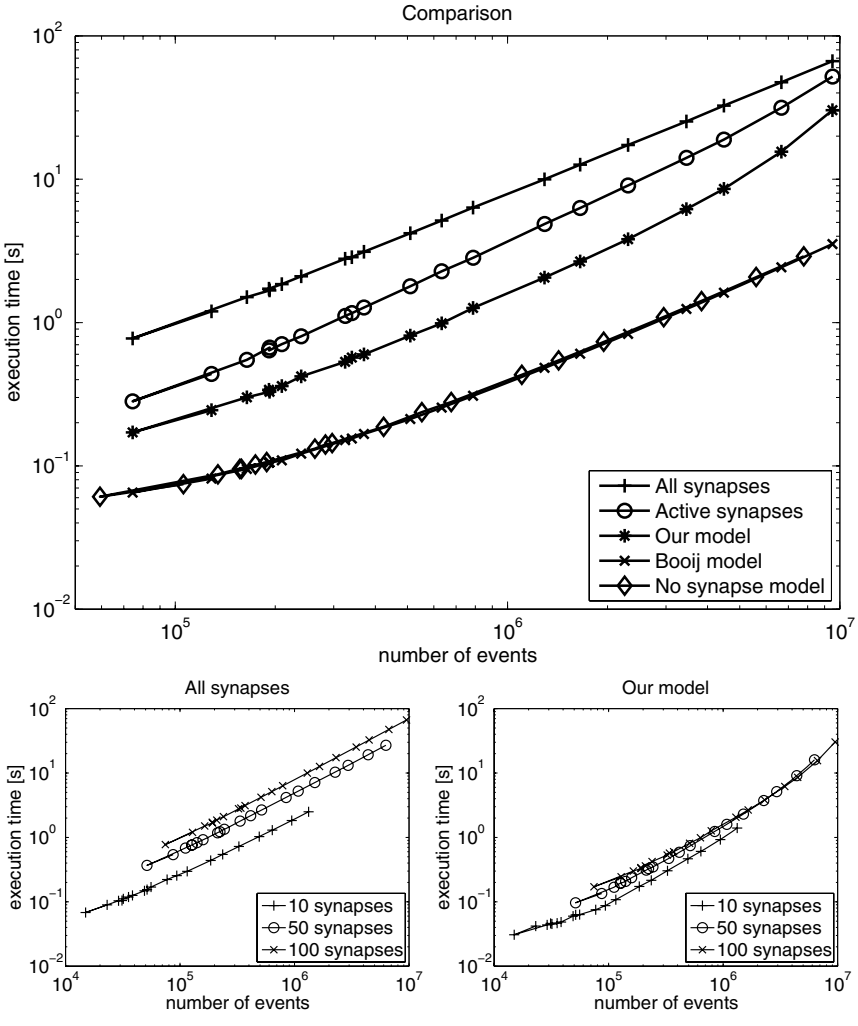

Fig. 4. Simulations performed on a AMD athlon 64 3400+ based system. We used a network of 1000 neurons and one input neuron. Neurons are randomly interconnected to each other with a spectral radius of 0.9 (this is the absolute value of the highest eigen-value of the connection matrix). The input neuron is randomly connected to the network with an interconnection fraction of 10% and fixed weights of 0.9. Each internal connection has a random delay between 0 and 10 ms and a random weight which is rescaled afterwards according to the spectral radius of the network [17]. Inputs are generated by a Poisson process within the simulated interval [0,10] seconds. The horizontal axis shows the total number of spikes (external and internal) processed by the simulator in the simulated interval ($\sim$ the network activity).

Figure top: comparison between Booij's model and several optimizations of our common LIF model with exponential synapses. All simulations use exact the same network with on average 100 synapse inputs per neuron.

Figure left: scaling of the not optimized common LIF model according to the average number of synapses per neuron. Figure right: the same but for the optimized model.

only active synapses) improves the performance of the simulator with approximately a factor 2, dependent on the activity of the network. The membrane-value estimation function further improves the performance with another factor 2. To show the efficienty of Booij's simulation model, we also plotted the execution time of a simple no-synapse LIF model. We see that the performance of both models is almost the same, although the computation power of Booij's model is much higher [10]!

A second advantage of our algorithm is that the simulation of multi-synapse neurons becomes less dependent of the interconnection density of the network, i.e. the average number of input connections of each neuron. This is shown in Figures 4 (left) and (right). We can see that for the unoptimized version of the algorithm, the execution time is highly dependent of the number of inputs, because for each incoming spike, all synapses have to be updated. In the optimized version however, it appears that the influence of the average number of synapses per neuron has decreased significantly. We did not plot the results for Booij's model, but we found that the time to calculate a number of events does not depend on the number of inputs of each neuron, as was expected (because each input shares the same synapse).

4 Future Work

The event based principle discussed in this paper is a sequential process: events must be handled one by one in the correct time-order. Although this allows yet for much faster simulations compared to time-step based simulation, it is still much to slow to be interesting for many applications (e.g. realtime simulations on modern architectures are still limited to networks of order of magnitude 10.000 neurons [3]).

An obvious way to accelerate this sequential process is the use of more processing units in parallel or through pipelining. Unfortunately the intense memory interaction of event simulation creates an important memory bottleneck. Also the inherent parallelism of SNN remains unused.

Our ultimate goal is to implement an event based simulator in parallel hardware. Therefore, we are building a SystemC framework for parallel discrete event simulation of SNN. It will allow us to investigate and optimize different synchronization mechanisms for parallel event based simulation (discribed in [3]) in order to build an efficient parallel SNN emulator. An important aspect is a hardware-friendly design: in a later stadium, the simulator will be implemented in digital hardware (FPGA) in order to benefit optimally from the inherent parallelism of SNN.

5 Conclusions

In this paper, we have shown that event based simulation is a good candidate for efficient simulation of SNN, characterized by discrete pulses and very low activity. However, not all neuron models are suited for such an event driven approach. A good compromise between complexity and possibilities is offered by the SRM_0 model. We discussed several techniques to efficiently implement a simple SRM_0 with exponential membrane function and synapse model. Because these techniques do not support multi-synapse models

efficiently, we presented an algorithm that handles multi-synapse models much more efficient. It provides a significant speedup of the simulations and moreover it improves the scalability of the simulator with regard to the number of synapses.

References

1. R. Brette. Exact simulation of integrate-and-fire models with synaptic conductances. Submitted, 2005.
2. R. Carrillo, E. Ros, E. Ortigosa, B. Barbour, and R. Agis. Lookup table powered neural event-driven simulator. In *Proceedings of the 8th International Work-Conference on Artificial neural Networks, IWANN 2005*, pages 168–175, 2005.
3. M. D'Haene. Parallelle event-gebaseerde simulatietechnieken en hun toepassing binnen gepulste neurale netwerken. Technical report, Universiteit Gent, 2005.
4. W. Gerstner and W. M. Kistler. *Spiking Neuron Models*. Cambridge University Press, 2002.
5. C. Grassmann and J. Anlauf. Distributed, event driven simulation of spiking neural networks. In *Proceedings of the International ICSC /IFAC Symposium on Neural Computation*, pages 100–105, 1998.
6. A. Graves, D. Eck, N. Beringer, and J. Schmidhuber. Biologically plausible speech recognition with LSTM neural nets. In *Proceedings of Bio-ADIT*, pages 127–136, 2004.
7. A. L. Hodgkin and A. F. Huxley. A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. *J. Physiol. (London)*, 117:500–544, 1952.
8. C. Lobb, Z. Chao, R. Fujimoto, and S. Potter. Parallel event-driven neural network simulation using the Hodgkin-Huxley neuron model. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 2005.
9. W. Maass. Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1):1–40, 1996.
10. W. Maass. Computation with spiking neurons. In *The Handbook of Brain Theory and Neural Networks*. The MIT Press, 2001.
11. T. Makino. A discrete-event neural network simulator for general neural models. *Neural Computing & Applications*, 11:210–223, 2003.
12. 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.
13. B. Schrauwen. Embedded spiking neural networks. In *Doctoraatssymposium Faculteit Toegepaste Wetenschappen*, pages on CD–ROM. Universiteit Gent, Gent, December 2002.
14. B. Schrauwen and M. D'Haene. Compact digital hardware implementations of spiking neural networks. In J. Van Campenhout, editor, *Sixth FirW PhD Symposium*, page on CD, 1 2005.
15. B. Schrauwen and J. Van Campenhout. Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic. In *Proceedings of ESANN'06*, 2006. To be published.
16. W. R. Softky. Simple codes versus efficient codes. *Current opinion in neurobiology*, 5:239–247, 1995.
17. D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir-based techniques for speech recognition. 2006. Accepted for publication at WCCI'06.
18. 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.