International Journal on Artificial Intelligence Tools

Concepts of efficient simulation with spiking neural networks

IOANA D. GOGA
CONEURAL Centre for Cognitive and Neural Studies,
Cluj-Napoca,
Romania.
marianioana@yahoo.co.uk

COLM G. CONNOLLY
Dept. of Computer Science,
University College Dublin,
Dublin, Ireland.
Colm.Connolly@ucd.ie

RONAN G. REILLY
Dept. of Computer Science,
National University of Ireland, Maynooth,
Maynooth, Ireland.
Ronan.Reilly@may.ie

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Spiking neural networks represent a powerful tool to investigate how cognitive functions emerge from the properties of basic components functioning cooperatively. Significant efforts have been made in the last decade towards the creation of simulation environments adjusted to the specifics of computation with spiking neurons. In this article we address efficiency issues in the simulation of pulsed neural networks on single processor systems. Both literature and novel algorithms are investigated with respect to their time and memory effeciency. The techniques and concepts presented make feasible the simulation of large networks or high neural activity rates.

Keywords: Event-driven simulation; pulsed neural networks; spike response model; high neural activity rates.

1. Introduction

Computer simulations of the nervous system play an increasingly prominent role in understanding the way neurons process information. Spiking neural networks received special attention after experimental evidence has accumulated to suggest that biological neurons use the timing of the spikes to encode and compute information[1,39,33]. Synaptic modification as a function of the timing between single spike events (i.e., spike-timing dependent learning) has been studied in mod-

els of temporal pattern recognition[38], temporal sequence learning[34], coincidence detection[14], directional selectivity[28]. It was shown that computations with spiking neurons are of great interest for scene segmentation[20], contour integration[43], movement control[28], binding through synchronisation[37,44].

Previous work on the efficient simulation of spike-processing neural networks has indicated that available simulators that reach high performance for rate-coding networks are not appropriate for networks of spiking neurons[21,46]. Efficient modelling of large scale spike-processing networks requires the design of simulation environments dedicated to the specifics of computing with temporal patterns. Promising work has been done by creating dedicated hardware for spike-processing networks[35,10] or mapping the simulations onto parallel computers [12,6,16].

In this article we describe a series of algorithms for time-efficient simulation of spiking neural networks on single processor architectures. By spiking or pulsed neural networks, we understand networks of simplified neural models (i.e., integrate-and-fire, spike response model). The reason why we focus on the efficient design of serial algorithms is twofold. Firstly, the increased costs for acquisition and maintenance of high performance parallel computers justify their use for a reduced number of applications. That is why most commonly, fundamental research on cognitive modelling of brain phenomena is developed on single processor systems. Secondly, even though parallelisation is possible, the most efficient way to implement it is to decompose a large network into loosely-coupled subnetworks that are mapped onto different processors in such a way that minimises the inter-processor communication[16,4]. This way, one is still faced with the problem of simulating large number of neurons in a serial manner.

The remainder of this article is organized as follows. In the second section, we present the formalism of a simplified neural model, as it was implemented in our work. The third section is dedicated to a discussion of general design considerations for discrete-time simulation of pulsed neural networks. In section four, several event-driven algorithms are investigated with respect to their effectiveness in simulating large number of highly inter-connected neurons. We present two novel algorithms, the quick-sort pool strategy and the circular priority-queue architecture, and we compare their performances with state-of-the-art literature algorithms.

2. Spike Response Model

The nonlinear dynamics of a spiking neuron can be accurately captured by a single variable model, which has a huge computational advantage over comprehensive mathematical models, such as the Hodgkin-Huxley equations. In the formalism known as the simplified Spike Response Method (SRM_0)[15], the dynamics of the neuron are encoded in two sets of kernels, representing the effects on a unit of its own spikes (η) and those of the other neurons (ϵ). The membrane potential V_i of

the neuron i is computed at each time moment t as follows:

$$V_i(t) = \eta(t - t_i) + \sum_{j \in \Gamma_i} w_{ij} \sum_{t_j^{(f)} \in F_j} \epsilon(t - t_j^{(f)}) \tag{1}$$

where Γ_i denotes the set of neurons presynaptic to i, F_j is the set of all firing times of neuron j and the w_{ij} account for synaptic strengths between cells.

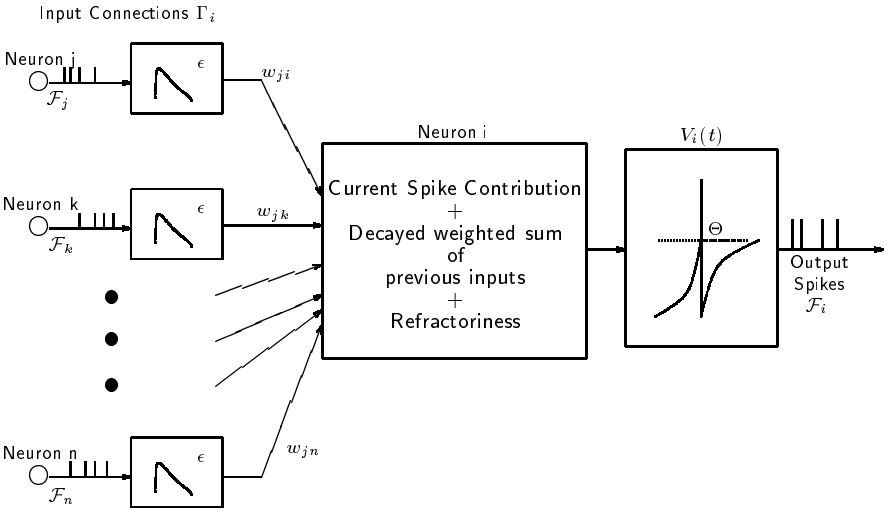

Fig. 1. Representation of spikes integration and generation in the spike response model. See in text for the meaning of notations.

If the sum of all excitatory and inhibitory contributions reaches a threshold value, the presynaptic neuron generates an output spike at time t_j^f, which travels along the axon, and reaches the postsynaptic neuron after a delay d. The postsynaptic response kernel ϵ evolves as a function of the difference $s = t - t_j^{(f)} - d$:

$$\epsilon(s) = exp(-\frac{s}{\tau_m})\,\mathcal{H}(s) \tag{2}$$

with

$$\mathcal{H}(s) = \begin{cases} 1, \text{ if } s \geq 0 \\ 0, \text{ otherwise,} \end{cases} \tag{3}$$

where $\mathcal{H}$ is the Heaviside step function and τ_m represents the neural time constant. After emitting the spike a node enters a refractory period described by the kernel η which, in our simulation, is simplified and depends only on the difference between current time and the last spike time $u = t - t_i^{(f)}$:

$$\eta(u) = -exp[-u^m + n]\,\mathcal{H}'(u)\,\Theta \tag{4}$$

with

$$\mathcal{H}'(u) = \begin{cases} \infty, \text{ if } 0 \leq u < 1 \\ 1, \quad \text{otherwise.} \end{cases} \tag{5}$$

Here, θ is the threshold and m and n are constants which give the decay rate of the refractory period.

Stochastic behaviour was accounted for in the model by adding noise in the generation of the axonal delays. In this implementation, the transmission delay is proportional with the Euclidian distance D between the presynaptic and the postsynaptic node. Thus, the value of the delay d is not fixed, but chosen stochastically from a gaussian distribution $p(D)$ with mean $\tilde{D}$.

3. General design considerations for efficient discrete-time simulation

To compute a network of simplified neural models, a continuous-time period is divided in intervals of constant duration T, referred to as time bins or time steps[23]. Within a time slice the network state is computed, by integrating for all neurons the inputs they receive and their internal state variables. The output spikes generated, become input to postsynaptic neurons during the following time slice. For the design of an efficient simulation of pulsed neural networks, one has to answer three related questions: (i) *how* to compute the internal state of a neuron, (ii)*when* to integrate the activity of each neuron in the network; (iii) *which* spikes should be delivered. Let us consider each of these questions in turn.

Firstly, the question of *how* to integrate, addresses effciency issues at neural level (see section 3.1). Compared to the detailed modelling of neural behavior[4,19], less computational effort is generally required for the integration of activity of simplified neural models (see [26]). Nevertheless, optimization at this level becomes essential when the activity of large number of neurons must be integrated. A second crucial design issue consists in answering the question of *when* to integrate each neuron's activity, in such a way to minimize the number of neurons computed at each time clock (section 3.2). The third question concerns the optimization of inter-neural communication management, by reducing the overall number of spikes which are delivered at any time moment. We will discuss separetely the management of external stimulation (section 3.3) from inter-neural communication (section 4.1).

3.1. *Optimisations at the neural level*

Simulation of a simplified neural model requires the consideration of a number of general design issues.

3.1.1. *Connectivity scheme*

Neurons in the brain are not fully connected[5]. Accordingly, biologically plausible architectures are made of spiking neurons sparsely connected in a non-random fash-

ion. One way to store the connectivity of a neuron during a simulation, is to use a list. If the list contains addresses of the target neurons it is referred to as *sender-oriented*. Otherwise, if it stores the addresses of the input neurons, it implements a *receiver-oriented* connectivity scheme. Previous simulation work suggested that a sender-oriented scheme is more appropriate for spike-processing networks[22,35]. That is, because spiking neurons have a discrete output and the sender-oriented scheme assures that only active connections are computed.

It is generally considered, that the receiver-oriented scheme is more advantageous for continuous output functions that cause constant activity on synapses. In ref. 28 we compared the performances of the two connectivity schemes for spiking neurons simulations. Our results suggest that the receiver-oriented scheme is less advantageous even in the presence of high frequencies of spike communication on synapses.

3.1.2. *Integration method*

Another aspect to be considered at the neural level concerns the integration of the membrane potential (see Figure 1). The computational effort at this level derives primarily from the integration of the synaptic inputs. The dynamics of neural activity in simplified models such as SRM, merge stochastic and deterministic components. The first component is associated with the random nature of the sequences of spikes and the second expresses the evolution of the neural state between two successive afferent spikes[30]. Instead of computing at each time step, the weighted sum of the inputs, the integration method can exploit the deterministic evolution of the postsynaptic potential. The sum of past presynaptic spikes is stored and decayed every time the contribution of a new spike has to be added. Besides reducing the simulation time, this method also has better memory management properties[8,28].

3.1.3. *Small internal potentials*

An immediate improvement of the simulation speed can be obtained by neglecting at integration, small values of postsynaptic potentials. Since the postsynaptic potential value decays exponentially in time, many postsynaptic potential values approach zero. One possibility is to tag the postsynaptic values as invalid, if they have been received too long ago, and not compute them anymore[35].

3.1.4. *Numeric precision*

Finally, in order to reduce the time spent per neuron integration, instead of using floating-point arithmetic, fast-fixed point arithmetic may be used without any degradation in network performance[35,36]. Another way of accelerating the calculations is by looking up numerical values in a table, rather than calculating them[4,32]. For a discussion in more detail on the requisite precision for computation with spiking neurons see ref. 23.

3.2. *Advancement of the simulation time*

When simulating neural networks on digital computers time proceeds in discrete, basic time units. Two kinds of discrete simulation can be distinguished with respect to the way time is simulated. In a continuous time simulation, time is advanced in steps of constant size and integration of all units is performed at each time step[11]. The time-driven protocol represents almost a standard for detailed neural modelling, because it ensures the time resolution needed for the integration of the differential equations[4].

An alternative to the continuous approach is to use a multi—order variable time step integration method. In this case, the increase in performance can be substantial for problems in which all states vary slowly for long periods of time. For instance, fast spikes are integrated using a fine—grained clock, but during the interval between spikes a larger time step can be used. Neuron[19] is a simulator which implements such an approach (i.e. the CVODE algorithm) as an alternative to the fixed time step method.

3.2.1. *Event-driven strategy*

Drawing inspiration from the techniques used in the simulation of discrete systems[11], modelling of spiking neurons can discretise the computation of neural state at different types of events occurrence. An event-driven simulator for spiking neural networks was first proposed in 46, based on the management of a queue of scheduled events. In 46, a specific event is generated and inserted in the queue for each modification of every element in the network (i.e., a neuron spike, end of refractory period, end of post-synaptic current pulse).

The idea of an event-list protocol was further adopted in 22, 35, 23 to represent in an efficient way the sparsely-coded signals generated in a network of spiking neurons. This strand of research takes a *neuron-oriented* approach to asynchronous simulation, which favours parallelisation by handling separately the neural operations from the inter-neural communications[23]. In this case, simulation is driven by the computation of different phases of information processing at neural level:

(i) **input phase**, when spikes are distributed and target neurons increment their internal potentials;
(ii) **decay or filter phase**, when the internal potentials are decayed;
(iii) **output phase**, when neurons compute their membrane potentials and emit spikes if necessary;
(iv) **learning**, when parameters are adapted.

More recently, an *event-oriented* approach to asynchronous simulation emerged as a natural choice for the simulation of large number of highly interconnected neurons[30,29,32,8]. The core of an event-oriented algorithm consists of processing events generated during simulation. Events can be defined at different levels of grain:

(i) **spike reception**, which determines the integration at the neural level of the new input with the membrane potential;
(ii) **spike generation**, which requires the delivery of delayed spikes on output connections;
(iii) **external stimulation**, which requires specific handling of the signals;
(iv) **communication messages** that are used for both intra-neural and inter-neural communication[8].

The main challenge in the design of current pulsed networks simulation is represented by the efficient management of huge pools of events generated during inter-neural communication. Original and state-of-the-art algorithms that address this issue are presented in section 4.

3.2.2. *Performance study of the event-driven vs. time-driven strategy*

Let us illustrate the major advantage that the event-driven strategy has over the time-driven approach. This is, time scaling with the decrease in the network activity. Each strategy has been separetly implemented for the simulation of a spiking self-organizing feature map. The learning rule was adapted from Ruf and Schmitt (1998) and is given by the formula:

$$\Delta w_{ij} = \eta \left(\epsilon_{ij} - w_{ij}\right) \frac{T_{out} - T_i}{T_{out}}, \quad \text{any } i \in N_c \tag{6}$$

where ϵ_{ij} is the postsynaptic potential from presynaptic unit j and is given by Eq. 2 with $s = T_i - T_j - d$, η is the learning rate, T_i and T_j are the times of the first spikes of neurons i, respectively j and N_c is a spatial neighborhood of the winner unit. A similar rule to Eq. 6 is applied for learning in lateral excitatory and inhibitory synapses. The synaptic efficacy of a lateral connection is modified depending on the activity of the connected neurons and upon the arrival time of the presynaptic spike. For a detailed description of the self-organizing process we refer the reader to ref. 28.

Self-organisation of a feature map is usually a time consuming process, due to the specifics of the learning procedure, which employs large number of training cycles and input patterns in order to ensure the accurate mapping of inputs[24]. Given an untrained spiking map, the neural activity starts out spreading over a large part of the network (i.e., full responsiveness of the map). However, in several hundred cycles, self-organzation of lateral and afferent synapses leads to a specialization of the network response, which converges to a stable activity bubble including a reduced number of firing units.

Because the event-driven algorithm computes in each time slice only the active fraction of neurons, it highly benefits from the localisation of the network activity. By contrast, the continuous time procedure computes the activity for the entire network, independently of the network activity (see Figure 2). The improvement

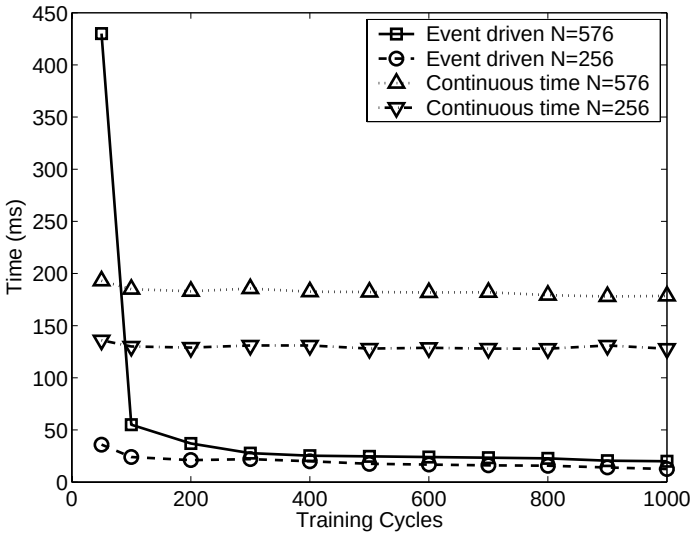

Fig. 2. Comparison between a straightforward event-driven strategy and a continuous, time-driven algorithm for a self-organization process. Computational times per training cycle are shown for the event and time driven algorithms, for two network sizes N = 256, 576 units. The self-organization task consists in the development of directional selectivity in a feature map of spiking neurons with Mexican-Hat shaped lateral connectivity. Note that the frequency of neural activity decreases from 100Hz during first 100 cycles, to about 30Hz at the end of training.

in performances is up to 10 times if the event-driven strategy is used for the self-organizing process described.

3.3. *Management of external stimulation*

It is common for an event-driven simulation of spiking neural networks, to manage the external spikes in a similar manner to the lateral spikes. A shortcoming of this strategy occurs when the network is embedded in a very large population of neurons that provide external background activity, meant for instance, to emulate the inputs coming from other areas of the brain (see refs. 2 and 45). In this case, creating an event for each external spike may excessively increase the size of the event list and overload the computational effort of the simulation. Hence, alternative strategies may be used to process the external stimulation.

Reutimann et al.[32] proposed an event-driven strategy which can efficiently manage the impact of large populations of external units on a neural network. In this case, the external spikes are not explicitly treated as synaptic events. Instead, authors assume that if the synaptic background activity is irregular, the impact of the external neurons on each simulated neuron of the network can be approximated by a continuous random current injection with appropriate statistical properties (see ref. 3). The state variables of the model are simulated starting from a determined value at the preceding update time, along all the possible trajectories given by the

different realisations of the input currents. When a relevant event occurs, the evolution is reduced to a single trajectory which is the one that has actually been followed by the neuron. The advantage is that it is not necessary to integrate iteratively the state variables during the interval between two relevant events.

Most generally, a distinctive treatment of the external stimulation can improve the performances depending on the particularities of the model studied, that is, the coding scheme and the frequency of the external input. In ref. 28, we implemented a distinct management of the external stimulation, by exploiting the specifics of the latency coding scheme used[39,27]. Temporal coding by relative latencies considers that input patterns are represented by the time advances of the afferent spikes relative to an arbitrary reference time T_{int} (see ref. 27). We used the time interval $[0, T_{\text{int}}]$ to accumulate the effects of input spikes and deliver them once to the hidden layers, instead of creating a distinct event for each external spike.

4. Novel techniques for efficient simulation

Today there is significant experimental evidence describing the existence in the cortex of bursts of spikes with a high frequency of oscillation[41,25], which may play a role in pattern recall in the mamalian hippocampus [31,17], or in processing of auditory auditory stimuli[18]. However, efficient simulation of high activity patterns has never been considered a criteria in the design of simulation environments. It has been argued that event-driven strategies are efficient solely for the simulation of spiking networks with a low neural activity[35,30]. Here is the reason.

The most expensive operation in the event-driven simulation is the insertion of events into the spike list SL sorted by delivery time, with a complexity of $O(\log_2(\text{length(SL)}))$. As long as the insertion operation depends on the length of a list that can grow tremendously (i.e., in the range of millions of events for a network of 1000 units), the event-driven algorithm cannot qualify as an appropriate simulation framework for high neural activity patterns. For an illustration of this issue, see again Figure 2, and note the left upper part of the graphics showing the event-driven algorithm's performances. These time values have been recorded during the initial phases of the self-organisation process, while the network activity reaches a spiking frequency of 100 Hz. Note that in this case, the supplimentary load introduced by the event-driven algorithm for the management of spike structures, reduces its perfomances bellow those of the continuous time algorithm.

In this section we investigate different concepts and strategies, proposed by the authors or from literature, aimed to improve the management of the events generated during the simulation. A novel algorithm is described for the efficient handling of the simulation of networks with high neural activity frequencies.

4.1. *Multiple spike concept*

Given a network with N neurons and a network activity per time slice a, a straightforward implementation of an asynchronous algorithm generates for aN firing neu-

rons with S synapses per neuron a maximum number of $aN \cdot S$ action potentials in each time slice. Instead of creating a distinct event for each spike, the algorithm may accumulate presynaptic spikes and deliver them together. We defined a data structure, referred to as *multiple spike*, to store the synaptic weights m on which spikes arrive at the same moment in time t to the target unit i. In 35 a similar concept, was proposed, referred to as weight caching. To implement this concept without a significant computational effort, spikes are cummulated for each unit, over time periods comparable to the average transmission delay $\tilde{D}$ in the network.

Implementation of the multiple spike concept leads to fewer spikes to insert in the event list and fewer to distribute. The overall computational load per time slice reduces by $1/m$. In the most favourable scenario, m can equal S, in which case the simulation time scales very well with the increase in the network activity. In the worst case, when m represents just a low percent of S, the method cannot improve significantly the algorithm's performances. The value of m is affected by the numerical precision used in the generation of the external stimulation and internal spikes. A low numerical precision in the generation of time events favours the aggregation of spikes and decreases the length of the event list.

4.1.1. *Performance study*

We compared the performances of several variants of a basic event-driven algorithm, for the self-organization process described above.

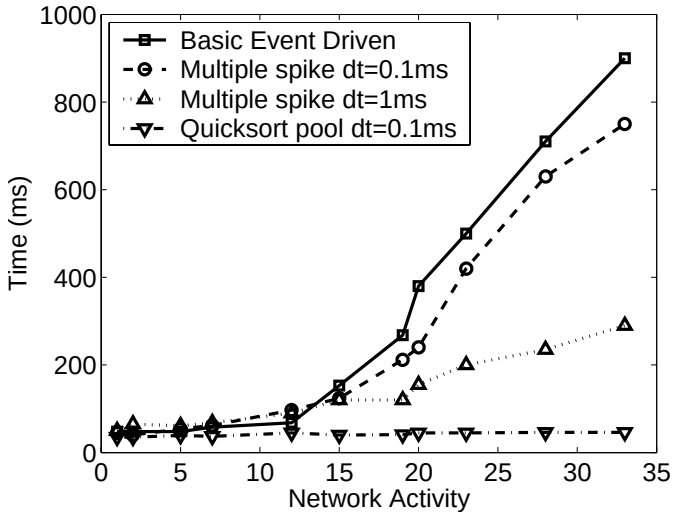

Fig. 3. Computational effort in CPU time to integrate 1000 units vs. levels of network activity, when different event handling methods are applied. The network activity is measured as the average number of spikes in one ms divided by the total number of neurons for N=576.

To compare the algorithms we evaluated the computational effort, measured in CPU time, to compute 1000 network units. One neuron computation involves: integrate its activity, generate output spikes, and insert new spikes in the event-list. Graphic 1 in Figure 3 shows the performance of a straightforward event-driven implementation that maintains a chronologically ordered spike list. It represents the baseline for the comparison of the algorithms implemented. Graphics 2 and 3 show the performances of an event-driven approach that implements the multiple spike concept at different numerical precisions $dt = 1$ms and $dt = 0.1$ms. When the multiple spikes strategy is applied to a series of events generated with a high time resolution (i.e., $dt = 0.1$ms) the probability of spikes accumulating is low, leading to an average improvement of 20%. By decreasing the numerical precision (graphic 3 in Figure 3) the method becomes twice as efficient (i.e., up 50% reduction of the computational effort). Graphic 4 illustrates the scaling of a novel algorithm Quick-sort pool, described in section 4.2.2.

The benefits of implementing the multiple spike concept are more evident with the increase in the network size and activity. For instance, a network of 3000 units, with a connectivity rate of 8%, and an activity rate of 20% generates approximately 2 million spikes in about 15 ms of simulation time. By aggregating spikes at a time resolution of 1 ms, only about 230 000 units are integrated. This leads to a reduction of almost one order of magnitude in the number of unit computations that would be done if each spike were to be delivered separately.

4.2. *Efficient management of the event-list*

The multiple spike concept was implemented by the authors to prevent the excessive growth of the event-list. The method has certain advantages, but it is problem dependent and it does not offer a general solution for the main question of the event-list management. In the remainder of this section, we shall examine the performances of four event-driven algorithms, two from literature and two novel algorithms proposed by authors. These will be evaluated based on their effectiveness in managing large dynamic data structures, and on their trade-offs between algorithmic complexity and implementation constraints.

4.2.1. *Layered delay-queues architecture*

In 30 an ingenious solution is proposed by splitting the global event-list in several FIFO queues each associated with a fixed axonal delay value (e.g., up to 16 different delays have been tested). All neurons' synapses are organised in matrix-structured layers, each layer corresponding to one spike transmission delay value. When a new event is generated it is directed to that queue corresponding to its transmission delay. Because in the same queue, all spikes share the same transmission delay, the spike generated first will be the oldest (top of the queue) and the most recent spikes will be the last in the queue. Accordingly, the data structure implemented for spikes management, needs no sorting and the insertion is done with O(1) complexity.

The approach described by Mattia and Del Giudice in 30 performs a significant reduction of the computational load incurred upon introduction of dynamic synaptic efficacies in the network. The algorithm preserves a linear increase of execution time per neuron with the network size, compared with the N^2 scaling of the computational complexity per neuron in the case of a basic priority queue policy. The major drawback of the algorithm is the limitation of the synaptic delays to a fixed number of values. Although formal neural models perform simplifications of the biological neuron description in several respects, most of the models account for noise effects on neural response, by introducing noisy thresholds, noisy integration or noisy delays[15,27]. The transmission delays represents a new set of parameters that have no counterpart in traditional neural network models, and which can be used to read a temporal latency code[13,40] or to enhance the flexibility of spiking neurons by being subject of learning algorithms[15,42]. By discarding the noise effects on synaptic transmission, the approach proposed in 30 is suited only for a limited set of applications.

4.2.2. *Quick-sort pool algorithm*

The solution to the efficient management of events-list structure proposed by one of the authors in 28 eliminates the insertion overhead while preserving the essentials of neural behaviour (i.e., noise effects). Instead of performing the ordered insertion, a new spike-event is simply added to an unordered pool of spikes, an operation of complexity O(1). Since the events have to be processed in chronological order, at constant time intervals, the simulation engine stops from processing spikes, and chronologically sorts those events from the pool which will be processed in the next interval. The time window during which processing of events takes place continuously is referred to as T_{window}. It is similar to the *safe window* concept used in parallel simulations to guarantee the temporal correctness of the algorithm[11].

The selection of spikes to be processed in the next time window is performed using a *quick-sort* algorithm. Most importantly, the sorting algorithm is run only over a small percentage q of the elements in the pool, namely those whose time stamp falls within the next processing interval (see Figure 4). This is realized by setting the first pivot point of the quick sort procedure to $t + T_{\text{window}}$. After the first iteration of the sorting algorithm the pool is shuffled and only the elements with a time stamp less than $t + T_{\text{window}}$ are sorted

The algorithm efficiency depends on the choice of the T_{window} value. On one hand, a small value, no larger than the minimum synaptic transmission delay is necessary to ensure the chronological processing of the events. On the other hand, a large time window prevents the sorting algorithm from failing to find any events in the pool within the corresponding interval, and it is also desirable because it decreases the number of sortings required.

The quick-sort pool algorithm was developed by the authors as a solution for the management of high neural activity frequencies and it was tested in a number of

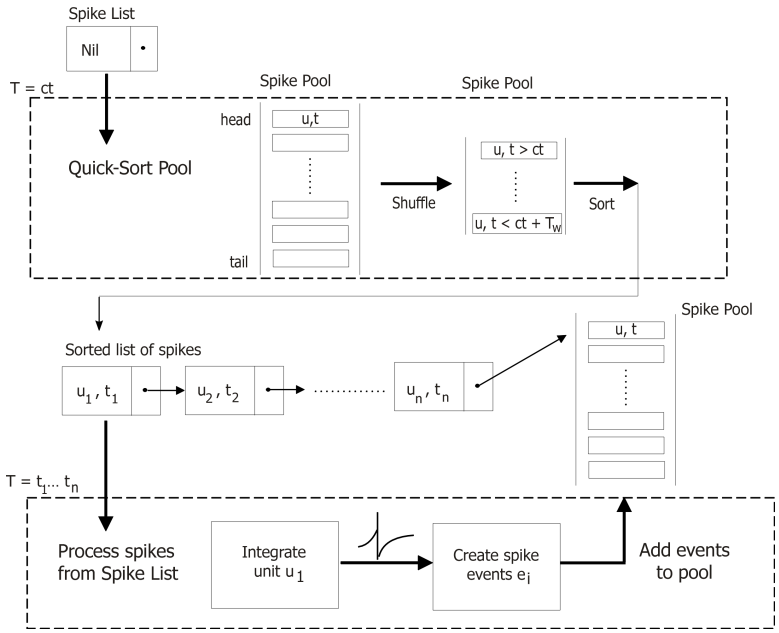

Fig. 4. The event-driven simulation engine based on the quick-sorting strategy of an unordered pool. Whenever the spike list becomes empty the quick sort algorithm is run upon a fraction of the elements in the spikes pool, namely those with a time stamp between the current time t and $t + T_{\text{window}}$. Sorted spikes are stored in the list and are processed by the simulation engine. Any new events are simply added at the end of the pool.

simulations described in 28, 9. In Figure 3 are shown the performances of the quick-sort algorithm for the self-organization task. Graphic 4 corresponds to a combination of the multiple spike concept with the quick-sort pool. For $T_{\text{window}} = 2$ ms and a time resolution of $dt = 0.1$ ms, the algorithm outperforms the rest of strategies and scales very well with the increase in the network activity.

To illustrate the differences between the basic event-driven implementation (graph 1) and the improved quick-sort pool algorithm (graph 4) we estimate the number of operations required in the implementation of the performance critical parts (i.e., loops). For algorithm 1 we have:

$$NO_1 \approx N_p \cdot (a + 1) + N_p \cdot \log(N_p) \tag{7}$$

with $N_p = \frac{T}{\sigma t} \cdot N \cdot S \cdot a$. N is the number of neurons, S the number of average synapses per neuron, a the network activity, T the time interval simulated, σt the time resolution in the generation of events. The overall critical computational effort in Eq. 7 results from: integration of neural activity for the units that receive spikes (i.e., the first term of summation); operations for new spikes distribution (i.e., the second term); the ordered insertion of spikes in the event list (i.e., last term). For

algorithm 4 we have:

$$NO_4 \approx \frac{N_p}{m} \cdot (a+1) + \frac{T}{T_{\text{window}}} \cdot \frac{q\,N_p}{m} \cdot \log(\frac{q\,N_p}{m}) \tag{8}$$

with $m \in [1, S]$ the fraction of spikes aggregated, and $q \in [0.05, 0.25]$ the fraction of the pool which is sorted at every T_{window} interval. The speed gain by implementing the latter algorithm can be up to 20 times, as long as the computational effort for the pool sorting remains low and independent of the pool size.

4.2.3. *Comparative performance study*

We compared the layered delay-queues algorithm[30] with the quick-sort pool algorithm with respect to the computational effort, in CPU time, spent for the integration of a given number of units. The comparison was facilitated by the fact that both algorithms have been used for the simulation of spike-driven learning processes of similar complexities. In 30 the layered delay-queues algorithm was applied to a learning task with integrate-and-fire neurons, while the spiking frequencies were maintained constant: $F_e = 2$ Hz for excitatory neurons and $F_i = 4$ Hz for inhibitory neurons.

The quick-sorting strategy was implemented for the self-organisation of a feature map of spiking neurons, with plastic excitatory and inhibitory lateral synapses (for a detailed description see ref. 28). The quick-sorting algorithm was tested for networks with the same size and connectivity rate (i.e., 10%) as reported in 30, with the main difference that frequency of neural activity in our simulations was of 100Hz. All our simulations were run on a 400 MHz PC running Linux.

Results reported for the layered delay-queues architecture are time per neuron needed to complete the simulation of one second of neural time in networks with different sizes. In order to compare the algorithms, we computed the CPU time required for the simulation of all operations entailed by the firing of the same number of neurons. We choose this measure mainly because the algorithms work at significantly different frequencies, which makes other computational measures inappropriate for comparison. Results are shown in Figure 5.

When examining the performances of these algorithms one should consider the significant difference in the frequencies of neural activity at which they work. The merit of quick-sorting algorithm is that it manages to keep the simulation time approximately twice as long as the layered-delays algorithm in the presence of a 30 times higher frequency of activity. Most importantly, it does this while preserving the noise effects in synaptic transmission.

4.2.4. *Calendar queues*

Another algorithm which deserves special attention is the priority queue proposed in 8 for the management of communication in a message-based event driven approach. The neural model on this approach is made up of several blocks (i.e., synapses,

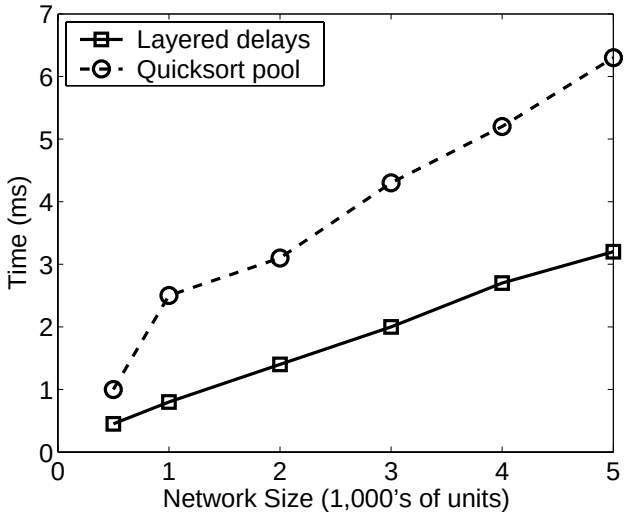

Fig. 5. Execution times per neuron vs. size of the network, computed for the simulation of the same number of firing units by two algorithms. Note that the average firing rate in the layered delay-queues simulation is 2.5 Hz, whereas the times reported for the quick sorting algorithm were measured for a neural activity of 100 Hz.

threshold, burst) and the communication between neurons and blocks within a single neuron is achieved by message passing. In order to reduce the time required for message insertion in the priority queue, the authors use a calendar queues technique, based on a look-up table (LUT). During simulation, messages are aggregated as multiples of a basic time step of $100\mu s$. All events with the same time stamp are then added to the same list. Using a LUT with 10^6 entries and a time step of $100\mu s$ means that events can be scheduled no further into the future than $100s$. For such a queue, insertion of events is $O(1)$.

The algorithm is also accompanied by a dynamic memory allocator for the new events created, which reduces by about 10% the average memory allocated per object. The memory required only for the storage of the look-up table containing pointers to the first message in each list is approximately of 4MB. For a network of 5000 units with 200 synapses per neuron about 50MB are required. The simulation time scales linearly with the increase in the network activity and 200 million events are handled in about 700 seconds.

4.2.5. *Circular priority-queue*

The circular priority-queue algorithm represents a combination of the multiple spike concept with the calendar queue techniques described in refs. 7 and 8. The circular priority-queue consists of a two dimensional matrix. Spikes are stored at the intersection of a row representing the time step of the simulation with the column corresponding to the number of the target neuron in the network. The circular

priority-queue incorporates the idea of *multiple spike* by grouping all spikes pending delivery to a neuron on a particular time step and delivering them as one event. In the example depicted in Figure 6, there are 50 events to be delivered to the neuron with number 2 at time step 2. When a dequeue operation is performed spikes are removed from the queue cell and passed for integration to the target unit.

The queue is circular, by wrapping around every T time steps. A pointer to the head of the queue sets the current time step being processed and moving the head pointer from one row to the next advances time. When adding spikes to the queue special care must be taken when the queue wraps around, in order to ensure that the tail of the queue is always behind the head.

Once a new spike is emitted, the *indirection matrix* is used to determine the storage location of the event in the spike queue. A row in the indirection matrix holds a permutation of the numbers of neurons in the simulation. The row in the spike queue where a new event is stored is given by its time stamp. The number of the column is given by the value stored in the indirection matrix at the intersection between the row given by the time stamp (i.e., 4) and the number of the target neuron (i.e., 5) (see Figure 6). Once all spikes for a given time step have been processed the corresponding row in the indirection matrix is shuffled.

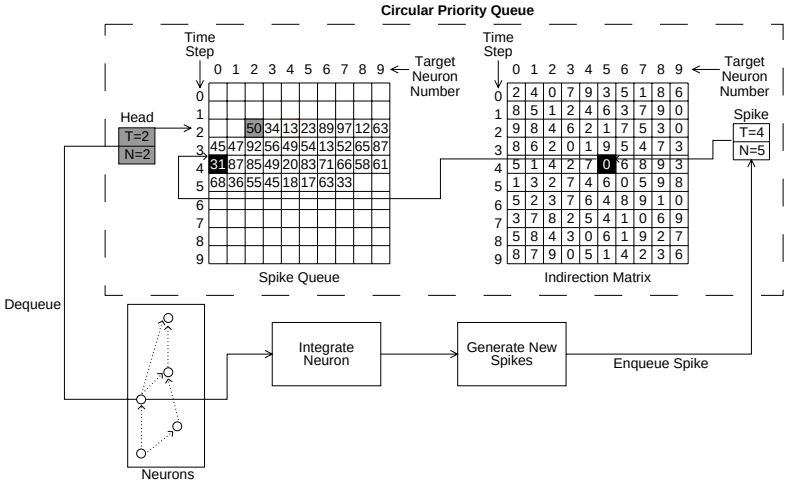

Fig. 6. A circular priority queue incorporating the multiple spikes paradigm. The diagram depicts the queue for a network of 10 neurons. The length of the queue is 10, indicating the number of time steps T that the queue can store before it must wrap around. A value in the spike queue indicates the number of spikes pending delivery to a neuron. A new spike is stored in the spike queue in the column indicated by the indirection matrix and in the row corresponding to its delivery time.

The advantage of this approach is that, regardless of the number of events stored, insertion time is always $O(1)$. The same is not true for spike removal from the queue, which depends on the number of spikes stored. The higher the queue occupancy rate, the faster spike removal will be. For occupancy rates greater than 10^3 events the

complexity of spike removal is $O(1)$. The major disadvantage of such an approach is the significant memory load introduced by the storage of the circular queue data structures. The memory load increases linearly with the number of neurons in the network and with the length of the queue (i.e., time steps before it wraps around).

4.2.6. *Comparative performance study*

We compared the performances of calendar queue algorithm[8] with the quick sort-algorithm and the circular priority-queue implementation. In 8, the event processed in the simulation is a message defined as a data packet containing the type of the message *on*, *off* and the time of delivery. The results reported are simulation times versus total number of messages processed. In the quick-sort and the circular priority-queue, processing of one spike-event would employ (with approximation) processing of the following messages: a synapse receiving the *on* message delivers an *on* message to the threshold block, which integrates the unit activity and if the unit spikes, sends an *on* message to the burst generator block, which in turn broadcasts an *on* message to all output synapses. We assumed a correspondence of 2 messages per one spike-event and we measured for the quick-sort and circular priority-queue algorithms the CPU times required for the simulation of same number of events as reported in 8.

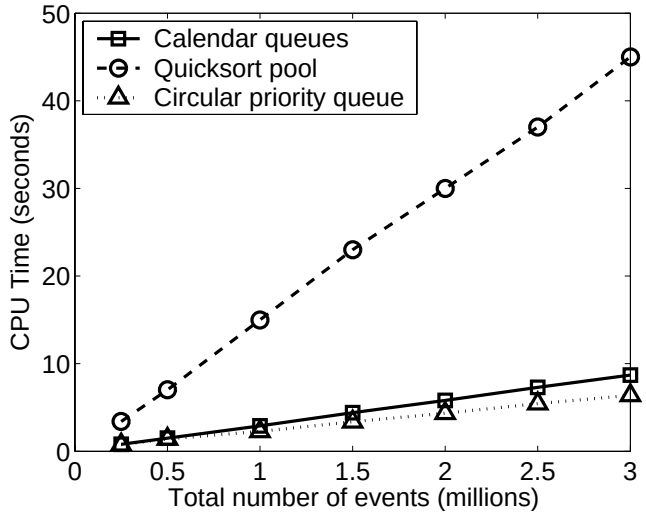

Fig. 7. Simulation time versus total number of events/messages processed, for three different algorithms.

Results indicate that the algorithms based on the calendar queues approach outperform the quick-sort method. The O(1) complexity obtained through the use of a look-up table turn these algorithms on a benchmark for evaluation of event-driven

strategies. The quick-sort method performs well for high neural activity patterns within medium scale networks, while with the increase of the total number of events generated, the additional effort for the sorting of the pool increases and slows down the simulation. The circular priority queue and the calendar queues algorithms perform similarly, with the latter being slightly slower.

5. Conclusions

This article addressed efficient simulation of networks of simplified neural models, such as spike response neurons. Simulation of large numbers of formal neural models is faced with specific problems, mainly due to inter-neuron communication, which requires adequate solutions. We described several strategies by which the implementation can be optimised with respect to three issues.

Concerning *how*the algorithm should integrate neural activity, it is important to exploit the deterministic evolution in formal models of the neural state between two consecutive spikes. Theoretically, *when* a unit's activity should be integrated allows two possibilities: at every time step, in a synchronous manner, or asynchronously, triggered by the reception of an incoming spike. Practically, it was shown that for spiking neural networks the event-driven approach brings, even in its most simple implementation, a significant improvement in the simulation performances. Asynchronous simulations that discretise the computation of neural state at spike-event occurrence, tends to become the de-facto standard for the simulation of large networks of formal neurons and becomes appealing for computationally expensive detailed neural modelling. Accordingly, it is crucial to find the most efficient strategies for dealing with very large numbers of events, generated either by large-scale networks, high activity patterns or massive external stimulation.

The most expensive operations of the event-driven strategy concern the management of the data structures that deliver spikes in chronological order. The problem arises when these data structures increase tremendously and the insertion/ removal time grows accordingly. One can try to prevent the excessive increase in the number of events to be processed, by using techniques such as the multiple spike. If this is not possible, than different strategies may be employed in order to reduce the complexity of large dynamical data structures management. We discussed several such techniques and we provided a coarse comparative evaluation of their performances. One may argue that the comparison of algorithms presented is neither very easy to achieve, nor very accurate, due to many differences in network architectures, learning procedures or results reported in each simulation. One has to consider that this paper is aimed at presenting a general review of the concepts and strategies existing in the literature for the efficient simulation of spiking neural networks. It is beyond the scope and the possibilities of our attempt to make a classification of the state-of-the-art algorithms. Evaluations offer a rough measure of the algorithms' performances, meant to outline each algorithm's advantages and shortcomings. The implementation of the appropriate combination of strategies is the choice of each

modeler and it is strongly dependent on the details of studied models.

6. Acknowledgments

The work of Ioana D. Goga (Marian) at University College Dublin has been supported through the Basic Research Programme of Enterprise Ireland.

References

1. M. Abeles, H. Bergman, E. Margalit, and E. Vaadia. Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of Neurophysiology*, 70(4), 1993.
2. D. J. Amit and N. Brunel. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cerebral Cortex*, 7(3):237–252, 1997.
3. D. J. Amit and M. V. Tsodyks. Quantitive study of attractor neural networks retrieving at low spike rates: I Substrate-spikes, rates and neuronal gain. *Network*, 2(3):259–274, 1991.
4. James M. Bower and David Beeman. *The book of GENESIS: Exploring realistic neuronal models with the GEneral NEural SImulation System*. Springer Verlag, New York, NY, 2nd edition, 1998.
5. V. Braitenberg and A. Schuz. *Cortex: Statistics and Geometry of Neuronal Connectivity*. Springer-Verlag, Berlin, 1998.
6. D. Brettle and E. Niebur. Detailed parallel simulation of a biological neuronal network. *Computational Science & Engineering*, 1(4):31–43, 1994.
7. Randy Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the AMC*, 31(10):1220–1227, Oct 1988.
8. Enric T. Claverol, Andrew D. Brown, and John E. Chad. Discrete simulation of large aggregates of neurons. *Neurocomputing*, 47:277–297, 2002.
9. Colm G. Connolly, Ioana Marian, and Ronan G. Reilly. Approaches to efficient simulation with spiking neural networks. In Howard Bowman and Christophe Labiouse, editors, *Connectionist models of cognition and perception II*, volume 15 of *Progress in neural processing*, pages 231–240. World Scientific, London, UK, 2004.
10. J G. Elias, and D .P.M. Northmore. Building silicon nervous systems with dendritic tree neuromorphs. In W. Maass and M. Bishop, editors, *Pulsed Neural Networks*,pages 3–53. MIT Press, 1999.
11. A. Ferscha and S. Tripathi. Parallel and distributed simulation of discrete event systems. Technical Report "CS-TR-3336", University of Maryland, College Park, MD, 1994.
12. Y. Fujimoto, N . Fukuda, and T. Akabane. Massively parallel architectures for large scale neural network simulations. IEEE Transactions on Neural Networks, 1992, vol. 3:6, pages 876888.
13. T. Gawne, T. Kjaer, and Richmond B. Latency: Another potential code for feature binding in striate cortex. *Journal of Neurophysiology*, 76(2):1356–1360, 1996.
14. W. Gerstner, R. Kempter, J.L. van Hemmen, and H. Wagner. A developmental learning rule for coincidence tuning in the barn owl auditory system. In *Computational Neuroscience: Trends in Research*, pages 665–669. Plenum Press, New York, 1997.
15. Wolfram Gerstner. Spiking neurons. In Wolfgang Maass and Christopher M. Bishop, editors, *Pulsed neural networks*, chapter 1, pages 3–53. The MIT Press, Cambridge, MA, 1999.

16. N. Goddard, G. Hood, F. Howell, M.Hines, and E. de Schutter. NEOSIM: Portable large-scale plug and play modelling. *Neurocomputing*, 38–40:1657–1661, 2001.
17. B. Graham. Dynamics of storage and recall in cortical associative memory networks. *Proceedings of International School of Neural Networks E.R.Caianiello, 8th Course Computational Neuroscience:Cortical Dynamics*, 2004.
18. C. Haenschel, T. Baldeweg, R.J. Croft, M. Whittington, and J. Gruzelier. Gamma and beta frequency oscillations in response to novel auditory stimuli: A comparison of human electroencephalogram (EEG) data with in vitro models. *Proceedings of the National Academy of Sciences of the United States of America*, 97–3: 7645-50, 2000.
19. M. L. Hines and N. T. Carnevale. The neuron simulation environment. *Neural Computation*, 9(6):1179–1209, aug 1997.
20. D. Horn and I. Opher. Collective excitation phenomena and their applications. In Wolfgang Maass and Christopher M. Bishop, editors, *Pulsed neural networks*, chapter 11, pages 296–320. The MIT Press, Cambridge, MA, 1999.
21. A. Jahnke, U. Roth, and H. Klar. Towards efficient hardware for spike-processing neural networks. In *Proceedings of WCNN '95*, Washington, USA, 1995.
22. A. Jahnke, T.D. Schonauer, U. Roth, K. Mohraz, and H. Klar. Simulation of spiking neural networks on different hardware platforms. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks - ICANN-97*, Lecture Notes in Computer Science 1327, pages 1187–1192. Springer-Verlag, 1998.
23. Axel Jahnke, Ulrich Roth, and Tim Schönauer. *Pulsed Neural Networks*, chapter 9, pages 237–257. The MIT Press, Cambridge, MA, 1999.
24. Teuvo Kohonen. *Self-organizing maps*. Springer, Berlin, 1995.
25. R. Levy, W.D. Hutchison, A.M. Lozano, and J.O. Dostrovsky. High-frequency Synchronization of Neuronal Activity in the Subthalamic Nucleus of Parkinsonian Patients with Limb Tremor. *Journal of Neuroscience*, 2020: 7766-7775, 2000.
26. W. Maass. On the computational complexity of networks of spiking neurons. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, vol. 7, pages 183190, MIT Press, 1995.
27. Wolfgang Maass. *Computing with spiking neurons*, chapter 2. The MIT Press, Cambridge, MA, 1999.
28. I. Marian. A biologically inspired computational model of motor control development. Master's thesis, University College Dublin, Department of Computer Science, 2003.
29. Ioanna Marian and Ronan Reilly. Self-organization of neurons coding directional selectivity in the motor cortex. In Diarmuid O'Donoghue, editor, *Proceedings of AICS 2001*, pages 253–263, Maynooth, Co. Kildare, 2001.
30. Maurizio Mattia and Paola Del Giudice. Efficient event-driven simulations of large networks of spiking neurons and dynamical synapses. *Neural Computation*, 12(10):2305–2329, Oct 2000.
31. O. Paulsen and E.I. Moser. A model of hippocampal memory encoding and retrieval: GABAergic control of synaptic plasticity. *Trends in Neuroscience*, 21–7:273-278, 1998.
32. Jan Reutimann, Michele Giugliano, and Stefano Fusi. Event-driven simulation of spiking neurons embedded in very large networks. Submitted to Neural Computation, May 2002.
33. F. Rieke, D. Warland, R. R. van Steveninck, and W. Bialek. *Spikes: exploring the neural code*. Computational Neuroscience. MIT Press, Cambridge, Ma, London, 1997.
34. P.D. Roberts. Computational consequences of temporally asymmetric learning rules: I.differential hebbian learning. *Journal of Computational Neuroscience*, 7:235–246, 1999.
35. T. Schönauer, S. Atasoy, N. Mehrtash, and H. Klar. MASPINN: Novel concepts for a

neuro-accelerator for spiking neural networks. In *Proceedings of VIDYNN '98*, Stockholm, Sweden, Jun 1998.
36. L. Shastri. A computationally efficient abstraction of long-term potentiation. *Neurocomputing*, 44–46:33–41, 2002.
37. Wolf Singer. Putative functions of temporal correlations in neocortical processing. In Christof Koch and Joel L. Davis, editors, *Large-Scale neuronal theories of the brain*, Computational Neuroscience, chapter 10, pages 210–237. MIT Press, Cambridge, MA, 1994.
38. J.-W. Sohn, B.-T. Zhang, and B.-K. Kaang. Temporal pattern recognition using a spking neural network with delays. In *Proceedings of Internationl Joint Conference on Neural Network (IJCNN'99)*, volume 4, pages 2590–2593, 1999.
39. S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381:520–522, 1996.
40. Simon J. Thorpe and Jacques Gautrais. Rapid visual processing using spike asynchrony. In *Advances in neural information processing systems*, volume 9, pages 901–907. MIT Press, Cambridge, MA, 1997.
41. R.D. Traub, M.O. Cunningham, T. Gloveli, F.E.N. LeBeau, A. Bibbig, E.H. Buhl, and M.A. Whittington. GABA-enhanced collective behavior in neuronal axons underlies persistent gamma-frequency oscillations. *Proceedings of the National Academy of Sciences of the United States of America*,100-19:1047-11052, 2003.
42. Tal Tversky and Risto Miikkulainen. Modeling directional selectivity using self-organizing delay-adaptation maps. In James M. Bower, editor, *Computational Neuroscience: Trends in Research, 2002*. Elsevier, New York, 2002.
43. R. Van Rullen, A. Delorme, and S. J. Thorpe. Feed-forward contour integration in primary visual cortex based on asynchronous spike propagation. *Neurocomputing*, 38–40(1–4):1003–1009, 2001.
44. Christof von der Malsburg. The correlation theory of brain function. Internal Report 81-2, Max-Plank-Institute for Biophysical Chemistry, Göttingen, Germany, 1981.
45. X.-J. Wang. Synaptic basis of cortical persisitant activity: The importance of NMDA receptors to working memory. *Journal opf Neuroscience*, 19(21):9587–9603, 1999.
46. Lloyd Watts. Event-driven simulation of networks of spiking neurons. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in neural information processing systems*, pages 927–934. Morgan Kaufmann Publishers, 1994.