

Maintaining causality in discrete time neuronal network simulations

Abigail Morrison and Markus Diesmann

Brain Science Institute, RIKEN, Wako, Japan

Summary. When designing a discrete time simulation tool for neuronal networks, conceptual difficulties are often encountered in defining the interaction between the continuous dynamics of the neurons and the point events (spikes) they exchange. These problems increase significantly when the tool is designed to be distributed over many computers. In this chapter we bring together the methods that have been developed over the last years to handle these difficulties. We describe a framework in which the temporal order of events within a simulation remains consistent. It is applicable to networks of neurons with arbitrary subthreshold dynamics, both with and without delays, exchanging point events either constrained to a discrete time grid or in continuous time, and is compatible with distributed computing.

10.1 Introduction

Neuronal network simulations are crucial for the advancement of computational neuroscience, as the non-linear activity dynamics is only partially accessible by purely analytical methods and experimental techniques are still severely limited in their ability to observe and manipulate large numbers of neurons. The brain is an unusual physical system, as it consists of elements (neurons) which can best be described by a set of differential equations, yet the interaction between these elements is mediated by point-like events (action potentials or spikes). It is, moreover, a very complex system - for example, each neuron in the cortex receives in the order of 10^4 connections from other neurons, both within its immediate area and from more remote parts of the brain. Simulating networks with this degree of complexity naturally suggests the use of distributed computing techniques. However, the meshing of continuous-time dynamics and discrete-time communication makes it notoriously difficult to define a consistent and sufficiently general framework for the integration of the dynamics.

There are two classical approaches to simulation: time-driven and event-driven. In the former, a computational time step h is defined. One iteration of a simulation involves each neuron advancing its dynamics over one time

step. If its conditions for generating an action potential are met, a spike is delivered to each of the neurons to which it projects. After all neurons have been updated, the next iteration begins. In the latter approach, an event queue manages the order in which spikes are delivered. Each neuron is only updated when it receives an event. If its conditions for generating an action potential are met, the new event is inserted into the queue. This algorithm can be defined very simply and can be very efficient if the neuronal dynamics is invertible - for example, if the arrival of a spike causes an immediate jump in the membrane potential which then decays exponentially. In this case, the neuron can only fire at the arrival of an incoming event, so the behavior of the neuron between the arrival of one event and the next is not relevant for the correct integration of the network. For neuron models with non-invertible dynamics, such as those where the maximum excursion of the membrane potential occurs some time after the arrival of a spike, it is much harder to define an event-driven algorithm. More sophisticated mechanisms are needed: for example, neurons might place provisional events in the queue if they are close to their firing conditions, but may have to revise their predicted spike times upon the arrival of further events (Lytton & Hines, 2005; Brette, 2006). In the following, we concentrate on the time-driven approach as defined above, which can incorporate any kind of subthreshold dynamics without changes being made to the updating and spike delivery algorithm, and has been shown to have good performance in simulating large-scale neuronal networks and to scale excellently when distributed (Morrison et al., 2005).

Here we present a framework which defines the interactions between the neurons without damaging causality, i.e. such that the order in which neurons are updated does not affect the outcome of a simulation. The framework is suitable for distributed computing. In Sec. 10.2 we cover the basics of point event interaction between continuous-time neuronal elements. We first discuss the historically important concept of neuronal networks with no propagation delay and describe an updating scheme ensuring that the simulation results are independent of the order in which neurons are updated (Sec. 10.2.1). We then demonstrate how this scheme needs to be adapted to incorporate delays which are multiples of the computational time step h (Sec. 10.2.2). Such networks have traditionally constrained spike times to the discrete time grid. However, for networks with propagation delays greater or equal to the computational time step, this constraint can be relaxed. In Sec. 10.3 we show how the scheme can be extended to permit neurons to generate and receive off-grid point events. Finally, we discuss how the propagation delays between neurons can be exploited to optimize communication efficiency between machines in a distributed environment (Sec. 10.4).

10.2 Networks with discrete spike times

In the following, we will assume that communication between neurons is mediated by synapses. When a neuron spikes, all of its outgoing synapses send a discrete event to their respective post-synaptic neurons. The event is parameterized with a weight w , which is interpreted by the post-synaptic neuron with respect to the post-synaptic dynamics it implements. For example, the post-synaptic neuron may interpret w as the size of an instantaneous jump in its membrane potential, or as the maximum amplitude of a post-synaptic current implemented as an α -function. In Sec. 10.2.2, the event is further parameterized by an integer delay k which expresses the propagation delay d between the neurons in units of the computational time step h , i.e. $d = k \cdot h$.

10.2.1 Networks without propagation delays

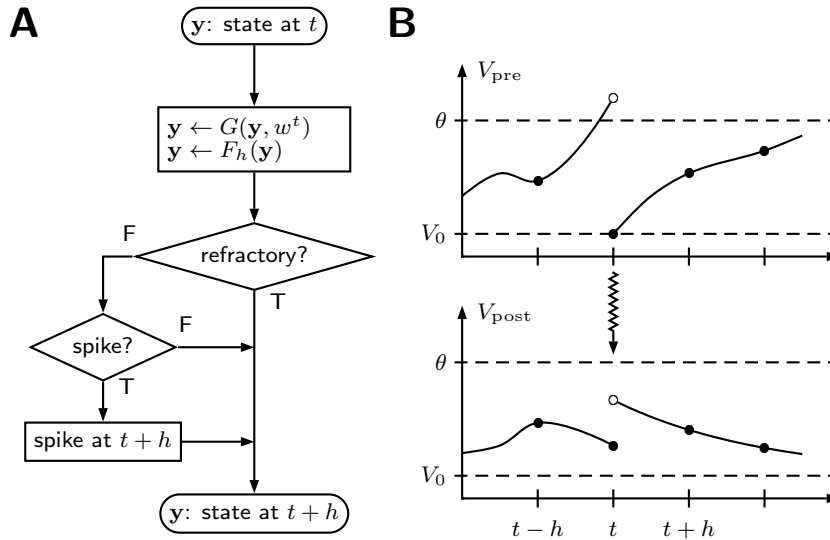


Fig. 10.1. Schematic of a discrete time simulation with no propagation delay. (A) Neuron update algorithm. The flowchart depicts the order of operations required to propagate the state y of an individual neuron by one time step h . Operator G modifies the state according to the incoming events and operator F_h performs the subthreshold dynamics. (B) Spike transmission and its effect on the postsynaptic neuron. The membrane potential V_{pre} of neuron *pre* crosses the threshold θ in the time step $(t-h, t]$, so a spike is emitted at time t and the membrane potential is reset to V_0 . The spike arrives at neuron *post* with no delay (zig-zag arrow). Filled circles denote the values of the membrane potential which can be reported by the neuron at the end of a time step. Intermediate (non-observable) values of the membrane potential are shown as unfilled circles.

Consider the following situation: neuron i and neuron j have a strong reciprocal inhibitory connection, such that a spike causes an instantaneous reduction in the membrane potential of the post-synaptic neuron. Each neuron is receiving enough input to drive it to spike at time t . If neuron i is updated to time t first, the spike is instantaneously delivered to neuron j . When j is updated, the strong inhibition prevents the membrane potential from passing the threshold, and so it does not itself generate a spike. Conversely, if neuron j is updated first, neuron j spikes at time t and neuron i is inhibited.

The order dependence in the above example is extremely undesirable. However, with a small conceptual adjustment the simulation can be made internally consistent. The convention is to define that the generation of a spike may only be influenced by spikes which precede it. This is depicted in the flowchart in Fig. 10.1A. When the neuron is updated from t to $t + h$, it first modifies its state according to the new spikes from its upstream neurons which fired at time t (operator G), for example incrementing the membrane potential or post-synaptic current. Then the subthreshold dynamics is performed to propagate the modified neuron state, including the new events, to $t + h$ (operator F_h). At this point the spiking criteria are applied; if they are fulfilled the neuron emits a spike. Thus, the effects of the spike on the neuron state are consistent with receiving a spike at t , as can be seen in the membrane potential of neuron *post* in Fig. 10.1B, but the earliest time the neuron can emit a spike as a result of receiving that spike is $t + h$. This is the equivalent of considering spikes to have an infinitesimal ϵ -delay, and has the effect of making simulations consistent, in that the order of updates does not affect the outcome. In our previous example of two neurons with mutual inhibition, both neurons would fire at time t . Assuming no refractory period, the effect of the mutual inhibition would result in a hyperpolarization of both neurons at time $t + h$.

10.2.2 Networks with propagation delays

Minimal delay

It is particularly simple to alter the algorithm described in Sec. 10.2.1 to one in which all propagation delays in the network are equal to the computation time step h . In fact, all it amounts to is changing the order of the two operators F_h and G , see Fig. 10.1 and Fig. 10.2. If neuron *pre* spikes at time t , this spike is delivered immediately to neuron *post* (Fig. 10.2B). When neuron *post* is being updated from t to $t + h$, first the subthreshold dynamics are performed to propagate the neuron by a step of h (operator F_h), then the neuron state is modified to include the new events visible at $t + h$, including the spike sent by neuron *pre*. Note that for this case and the case where no propagation delay is assumed, the infrastructure of the simulation is the same. A spike produced at time t is delivered immediately to its target, but due to the different order

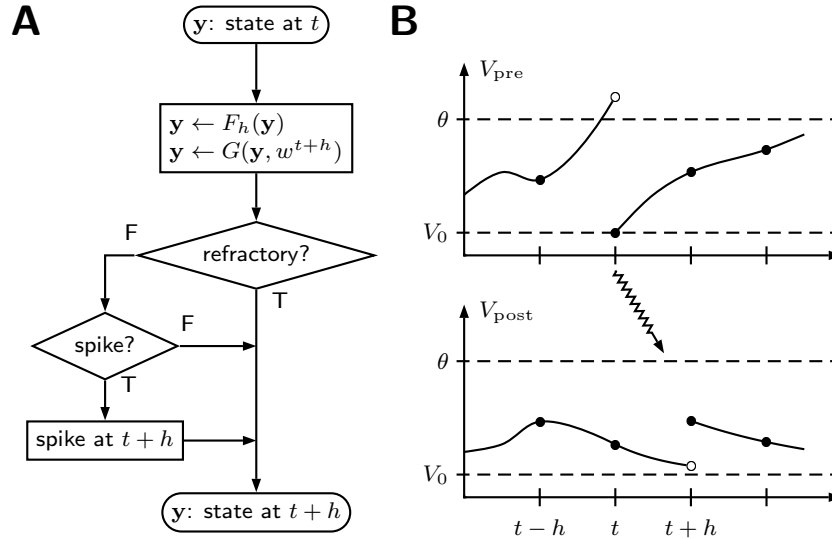


Fig. 10.2. Schematic of a discrete time simulation with propagation delays. (A) Neuron update algorithm. The flowchart depicts the order of operations required to propagate the state \mathbf{y} of an individual neuron by one time step h . Operator F_h performs the subthreshold dynamics and operator G modifies the state according to the incoming events. Note that the order of these two operations is the reverse of the order shown in Fig. 10.1. (B) Spike transmission and its effect on the postsynaptic neuron. As in Fig. 10.1B, neuron *pre* emits a spike at time t . The spike arrives at neuron *post* with a minimal delay of h (zig-zag arrow). Filled circles denote the values of the membrane potential which can be reported by the neuron at the end of a time step. Intermediate (non-observable) values of the membrane potential are shown as unfilled circles.

of operations, the effect of the spike is instantaneous in the first case, but delayed by h in the second.

The data structure used to store the pending events can be very simple. If all the synapses have the same dynamics, varying only in amplitude, then each neuron only requires one buffer to store incoming events. Examples of neuron models which only require one buffer are those in which synaptic interactions cause an instantaneous increment to the membrane potential, or induce exponential post-synaptic currents. Other neuron models may have more than one synaptic dynamics, such as a longer time constants for inhibitory than for excitatory interactions. Clearly, in this case one buffer per time constant would be required. However, for the sake of simplicity, we will focus on neuron models with only one synaptic dynamics.

Depending on the implementation, either a one-element or a two-element buffer is sufficient to maintain causality in the system. If the global scheduling algorithm iterates through all the neurons twice - once to advance the

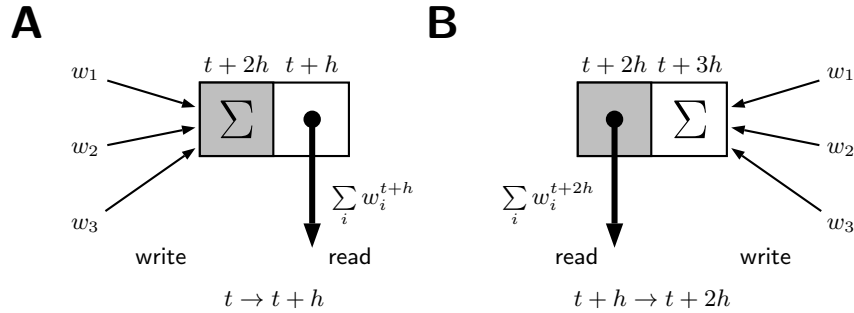


Fig. 10.3. A two-element buffer, suitable for use in networks with a delay of h . (A) In the time step $(t, t+h]$, the gray side of the buffer is the ‘write’ side, and it sums the weights of events generated in this time step which are to become visible in the next step. The white side of the buffer is the ‘read’ side, containing the summed weights of all the events received in the previous time step, $\sum_i w_i^{t+h}$. Once the neuron has read out the buffer, the ‘read’ side is emptied. (B) After all neurons have been updated, the neuron buffers are toggled. Now the empty white side receives new events, and the gray side is read by the neuron as it updates from $t+h$ to $t+2h$.

dynamics, the second time to apply the spiking criteria and deliver any generated events - then a one-element buffer is sufficient, as the ‘read’ and ‘write’ phases are cleanly separated. However, for reasons of cache effectivity it may be preferable to iterate through all the neurons only once - i.e. for each neuron, advance its dynamics, apply its spiking criteria and deliver the new events if it spikes. In this case, the ‘read’ and ‘write’ phases are no longer cleanly separated, and a two-element buffer is required.

A two-element buffer is depicted in Fig. 10.3. One side of the buffer can be considered as the ‘read’ side, the other as the ‘write’ side. When neuron i is modifying its state to incorporate the new events (operator G in Fig. 10.2), it collects the summed weights becoming visible at $t+h$ from the ‘read’ side. The act of reading clears that side of the buffer. If any neurons which project to i emit a spike at $t+h$, the weight of this event is added to the ‘write’ side. After all the neurons have been updated, all their buffers are toggled so that the empty ‘read’ sides are now ‘write’ sides, and the ‘write’ sides, containing those events which become visible at $t+2h$, are now ‘read’ sides. Thus, the order in which the neurons are updated does not affect the outcome, as events generated in one time step are always cleanly separated from those generated in the next.

Exactly the same structure can be used for networks with no propagation delay, except the assignation of times to buffer elements is shifted by h : in Fig. 10.3A, the left side receives events for the time step $t+h$ while the summed weight of events becoming visible in time step t is read out of the right side.

General delay

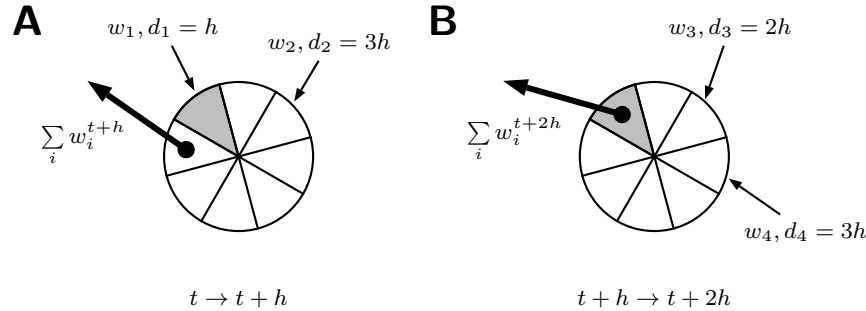


Fig. 10.4. A random access ring buffer, suitable for use in networks with delays which are integer multiples of h . The weights w_i of incoming events are written to the segments corresponding to their delays d_i , such that a delay $d = k \cdot h$ corresponds to the segment k along from the current read position. (A) Time step $(t, t + h]$: the read position of the buffer is the segment containing the summed weights of all the events which are to become visible at $t + h$. (B) Time step $(t + h, t + 2h]$: The read position has moved to the next segment (gray), containing the summed weights of all the events which are to become visible at $t + 2h$.

A system to simulate a network with a minimal propagation delay h can be converted into one encompassing many different delays, as long as they are all integer multiples of h , by replacing the simple two-element buffer with a ring buffer. A traditional ring buffer is an implementation of a queue. The data is represented in a contiguous series of segments. New elements are appended to one end of the series, and the oldest elements are popped off the other end. Thus the arc containing the data can be imagined as rotating around the ring as data is added and removed. In this way a queue can be implemented without continually having to allocate fresh memory. For our purposes, we need something more like a random access ring buffer, as shown in Fig. 10.4 (see also Morrison et al., 2005). Each segment of the ring corresponds to one time step. When the neuron is updating from t to $t + h$, it reads from the segment containing the summed weights of the events which become visible at $t + h$, and then clears this segment. Incoming events are sorted into the other segments depending on their delays: a spike with a delay of $k \cdot h$ would be sorted into the k th segment along from the current read position. After all the neurons have been updated, the read positions for all buffers is moved around one segment. The ring buffer needs to be appropriately sized if the correct order of events is to be maintained - it must be large enough to accommodate the largest propagation delay between neurons in the simulated system, $d_{\max} = k_{\max} \cdot h$, without ‘wrapping’. Therefore the optimal size for the buffer is $k_{\max} + 1$. Depending on implementation, d_{\max} may either be

specified before the creation of the network, or, more elegantly, determined dynamically whilst the network is created.

10.3 Networks with continuous spike times

In the systems discussed above, spike times were constrained to the discrete time grid. However, Hansel et al. (1998) showed that forcing spikes onto the grid can significantly distort the synchronization dynamics of certain networks. The integration error decreases only linearly with the computational step size, so a very small h is necessary to capture the dynamics accurately. An alternative solution is to interpolate the membrane potential between grid points and evaluate the effect of incoming spikes on the neuronal grid in continuous time (Hansel et al., 1998; Shelley & Tao, 2001). This concept was extended in Morrison et al. (2006) by combining it with exact integration of the subthreshold dynamics (see Rotter & Diesmann, 1999). Here we discuss how the scheme described in Sec. 10.2.2 can incorporate off-grid spike times.

Continuous spike times can easily be incorporated into discrete time simulations without having to implement a central queuing structure if the minimum propagation delay is greater than or equal to the computational step size h and an appropriate representation of time is used. In the networks discussed in Sec. 10.2.2, first the subthreshold dynamics is advanced by one time step from t to $t + h$, and then the spiking criteria are applied. If the neuron state passes the criteria (for example, by having a membrane potential above a threshold), the neuron emits a spike which is assigned to the time $t + h$. Now, let us assume that the actual spike time can be determined more precisely, either by interpolation of the membrane potential or by inverting the dynamics or by any other method, such that $t < t_{\text{spike}} \leq t + h$. If the propagation delay to the neuron's post-synaptic target is $k \cdot h$, the event should become visible at time $t_{\text{spike}} + k \cdot h$, which is in the update interval $(t + k \cdot h, t + (k + 1) \cdot h]$. An appropriate representation of the spike time therefore consists of an integer time stamp $t + h$ and a floating point offset $\delta = t_{\text{spike}} - t$. By definition, δ is in the interval $(0, h]$. This choice of representation allows the infrastructure described in Sec. 10.2.2 to be kept with only minimal changes. The propagation delay k can still be used to sort the event into the segment which will be read in the step $(t + k \cdot h, t + (k + 1) \cdot h]$, but the ring buffer is adapted to hold a vector of events in each segment instead of a single value. The weight w and offset δ of the event are appended to the vector. When the neuron performs this update step, the vector is first sorted in order of increasing δ . Note that the vector is just the simplest possible implementation and could be replaced by a more sophisticated data structure such as a calendar queue (Brown, 1988).

The subthreshold dynamics is then advanced from the beginning of the time step to the arrival time of the first event, at which point the neuron state is modified to take account of the first event. Then the dynamics is

advanced between the arrival time of the first event and the arrival time of the second event, at which point the neuron state is modified to take account of the second event, and so on until all the events for that update step have been processed. Finally the dynamics is advanced from the arrival time of the final event to the end of the time step. Thus all incoming events have been processed in the correct temporal order.

The scheme described above is very general and can be applied to any kind of subthreshold dynamics, allowing the processing and generation of spikes in continuous time within a discrete time algorithm. No global queuing of events is required as each neuron queues its events locally. In the case that the subthreshold dynamics is linear, this can be exploited such that not even local queuing is required. This involves a slightly more complicated mechanism for receiving spikes, see Morrison et al. (2006).

10.4 Distributed networks

Neuronal network simulations can consume a huge amount of memory, especially if biologically realistic levels of connectivity are assumed. In the cortex each neuron has of the order of 10^4 incoming synapses, and a connection probability of about 0.1 in its local area (Braitenberg & Schüz, 1991). Therefore, a network fulfilling both of these constraints must have at least 10^5 neurons. This is equivalent to about 1 mm^3 of cortical tissue, and represents a threshold network size for simulations, as beyond this point, the number of synapses increases only linearly with the number of neurons, rather than quadratically as is the case for smaller systems. Such networks contain 10^9 synapses, which, even using an extremely simple synapse representation, require several gigabytes of RAM. This state of affairs has naturally prompted much interest in distributed computing, for example (Hammarlund & Ekeberg, 1998; Harris et al., 2003; Morrison et al., 2005). However, distribution raises new issues about maintaining causality in the simulated system. If neuron *pre* projects to neuron *post* with a delay of $k \cdot h$, a spike produced by neuron *pre* at time t should be visible to neuron *post* at time $t + k \cdot h$, no matter whether the two neurons are located on the same machine.

In Morrison et al. (2005), it was demonstrated that it is more efficient to distribute a neuronal simulation by placing the synapses on the machines of their post-synaptic neurons than of their pre-synaptic neuron. This is equivalent to distributing a neuron's axon but keeping its dendrite local. That way, when a neuron fires, only its index must be sent across the computer network, rather than a weight and delay for every one of its post-synaptic targets. For neuronal networks with biologically realistic levels of connectivity, this can represent a difference of several orders of magnitude in the amount of information being communicated. One way of ensuring that spikes are always delivered on time is to communicate in each time step, after all the neurons have been updated but before the read positions of their buffers has been

incremented (see Sec. 10.2.2). However, this approach is sub-optimal with respect to communication efficiency. Communication between machines has an overhead, so it is more efficient to send one message of N bytes than N messages of one byte each. Fortunately, it is generally possible to communicate less often and still deliver the events correctly. For this, it is necessary to determine the minimum propagation delay between neurons in the simulated system, $d_{\min} = k_{\min} \cdot h$. As in the case of d_{\max} , described in Sec. 10.2.2, depending on implementation d_{\min} could either be specified before the creation of the neuronal network, or determined dynamically whilst the neurons are connected. By definition, a spike cannot have an effect on a post-synaptic neuron earlier than k_{\min} time steps after generation. Therefore, as long as the temporal order of spikes is preserved, it is possible to communicate in intervals of k_{\min} time steps. This can be a significant improvement, as the minimum delay can be considerably larger than the computational time step.

Preserving the temporal order of spikes has two parts: correct storage before communication, and correct delivery after communication. If spikes are constrained to the discrete time grid, for correct storage it is sufficient for each machine to store the indices of spiking neurons in a buffer, with tokens separating the indices of neurons which spiked in one time step from those which spiked in the previous or next step. Thus, at the end of k_{\min} time steps, the buffer contains k_{\min} blocks neuron indices separated by $k_{\min} - 1$ tokens. Then the machine sends this buffer of indices to all other machines, and receives buffers in turn. If the spikes are not constrained to the grid, as described in Sec. 10.3, in addition to the index of a spiking neuron, its spike offset δ must be buffered and communicated as well. For correct delivery, it is necessary to activate the synapses of the neurons registered in these buffers whilst taking the temporal order into consideration. The position of an index in the buffer represents the communication lag, k_{lag} , in delivering the information that a neuron has fired, i.e. $k_{\text{lag}} = k_{\min}$ for an index in the first block of data, $k_{\text{lag}} = k_{\min} - 1$ for an index in the second block of data and so on until $k_{\text{lag}} = 1$ for indices in the last block of data. Note that this assumes that the read positions of all the ring buffers had been incremented before exchange of spike data (see Sec. 10.2.2). If the order is exchanged, then the communication lag ranges from $k_{\min} - 1$ for the first block to 1 for the last block. If the communication lag is subtracted from the propagation delay encoded in a synapse, then the event will become visible to the postsynaptic neuron at exactly the same time as it would in a serial simulation. For example, consider a synapse from neuron *pre* to neuron *post* with a weight w and delay $k \cdot h$. In a serial simulation, if neuron *pre* emitted a spike at time t , w would be added to the ring buffer in the k th segment along from the current read position. In a distributed simulation, w would be added to neuron *post*'s ring buffer $k - k_{\text{lag}}$ segments along from the current read position, with k_{lag} determined by the position of the index of neuron *pre* in the received index buffer. A slightly more sophisticated version of this approach is discussed in Morrison et al. (2005).

10.5 Conclusions and perspectives

We have shown how relatively simple methods and data structures can be used to simulate networks of spiking neurons in discrete time whilst reliably maintaining the temporal order of events. If spike times are constrained to the discrete time grid, the framework is applicable to networks with no propagation delay and to networks with arbitrary delays which are integer multiples of the computation step size h . If spike times are not constrained to the grid, the framework is only applicable to networks with propagation delays greater than or equal to the computational step size h . In this case, the constraint that delays must be an integer multiple of h could be relaxed, because floating point offsets and delays can always be recombined on the fly to produce integer delays and floating point offsets. All these networks can be implemented in a distributed environment in a symmetrical fashion, i.e. the architecture is peer-to-peer rather than master-slave.

These networks can be simulated efficiently because the delivery time of an event in the simulated system has been decoupled from the arrival time at the post-synaptic neuron and the temporal resolution of the simulation. This is the concept that underlies both the ring buffers and the minimum delay intervals for communication. However, if delivery and arrival times are decoupled, this can be problematic for synaptic processes which depend on the state of the post-synaptic neuron, for example spike-timing dependent plasticity (Markram et al., 1997; Bi & Poo, 1998). An algorithm has been developed which maintains the correct relationships if propagation delays are assumed to be predominantly dendritic (Morrison et al., 2006). However, if the propagation delays are predominantly axonal, the framework presented here is not sufficient and will have to be adapted.

Acknowledgements

We acknowledge constructive discussions with the members of the NEST initiative (www.nest-initiative.org). Partially funded by DIP F1.2, BMBF Grant 01GQ0420 to the Bernstein Center for Computational Neuroscience Freiburg and EU Grant 15879 (FACETS).

References

- Bi, G.-q., & Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.
- Braitenberg, V., & Schüz, A. (1991). *Anatomy of the Cortex: Statistics and Geometry*. Berlin, Heidelberg, New York: Springer-Verlag.

- Brette, R. (2006). Exact simulation of integrate-and-fire models with synaptic conductances. *Neural Comput.* 18(8).
- Brown, R. (1988). Calendar queues: a fast $O(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM* 31(10), 1220–1227.
- Hammarlund, P., & Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *J. Comput. Neurosci.* 5(4), 443–459.
- Hansel, D., Mato, G., Meunier, C., & Neltner, L. (1998). On numerical simulations of integrate-and-fire neural networks. *Neural Comput.* 10(2), 467–483.
- Harris, J., Baurick, J., Frye, J., King, J., Ballew, M., Goodman, P., & Drewes, R. (2003). A novel parallel hardware and software solution for a large-scale biologically realistic cortical simulation. Technical report, University of Nevada.
- Lytton, W. W., & Hines, M. L. (2005). Independent variable time-step integration of individual neurons for network simulations. *Neural Comput.* 17, 903–921.
- Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213–215.
- Morrison, A., Aertsen, A., & Diesmann, M. (2006). Spike-timing dependent plasticity in balanced random networks. *Neural Comput.* under review.
- Morrison, A., Mehring, C., Geisel, T., Aertsen, A., & Diesmann, M. (2005). Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Comput.* 17(8), 1776–1801.
- Morrison, A., Straube, S., Plesser, H. E., & Diesmann, M. (2006). Exact subthreshold integration with continuous spike times in discrete time neural network simulations. *Neural Comput.* in press.
- Rotter, S., & Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biol. Cybern.* 81(5/6), 381–402.
- Shelley, M. J., & Tao, L. (2001). Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks. *J. Comput. Neurosci.* 11(2), 111–119.