SpikeNET: An Event-driven Simulation Package for Modeling Large Networks of Spiking Neurons

Arnaud Delorme and Simon J. Thorpe

CERCO, Faculté de Médecine, 133 route de Narbonne, 31062 Toulouse Cedex, France

arno@salk.edu, thorpe@cerco.ups-tlse.fr

Abstract: Many biological neural network models face the problem of scalability because of the limited computational power of today's computers. Thus, it is difficult to assess the efficiency of these models to solve complex problems such as image processing. Here, we describe how this problem can be tackled using event-driven computation. Only the neurons that emit a discharge are processed and, as long as the average spike discharge rate is low, millions of neurons and billions of connections can be modeled. We describe the underlying computation and implementation of such a mechanism in SpikeNET, our neural network simulation package. The type of model one can build is not only biologically compliant, it is also computationally efficient as 400 000 synaptic weights can be propagated per second on a standard desktop computer. In addition, for large networks, we can set very small time steps (less than 0.01 ms) without significantly increasing the computation time. As an example, this method is applied to solve complex cognitive tasks such as face recognition in natural images.

Keywords: event-driven, integrate-and-fire, parallel processing, neural network, biological simulations, large simulations, real time processing.

1. Introduction

There are currently a large number of different software packages that can be used for simulating neural networks. Many have been designed for simulating networks of artificial neurons and make no attempt to model the detailed biophysics of neurons. The underlying units have no structure, and their outputs typically consist of a single continuous value (often in the range 0 to 1 or from -1 to +1). While such systems have been widely used, and have applications in a wide range of engineering and financial areas, few would regard them as being useful as tools for the computational neuroscientist.

At the other end of the spectrum, sophisticated programs exist such as GENESIS and NEURON, which are good for performing detailed biophysical simulations that take into account factors like the dendritic structure and complex channel kinetics [1, 2], but the level of detail makes it difficult to simulate very large networks.

In this article, we describe SpikeNET, an object oriented neural network simulation package written in C++ code that lies between these two extremes. SpikeNET is available for public download under a GNU public license [32]. It is sufficiently biologically realistic to examine the role of temporal properties such as synchronous or asynchronous spiking in neurons, and yet sufficiently simple to allow real-time simulation of large-scale networks of neurons. SpikeNET uses event-driven computation, a computationally efficient approach to propagate spike lists in neural networks [3-6]. We have already briefly presented our SpikeNET package elsewhere [4] and in this paper we intend to describe in details the underlying computation. Note that this paper describes the version of SpikeNet that is available for public download. A second version, developed in collaboration with the company SpikeNet Technology (www.spikenet-technology.com), has different objectives. In that case, the underlying computational architecture has been considerably revised to allow specific image processing applications to be run efficiently. Although the two versions share a number of fundamental features, the precise details of the implementation vary considerably.

We first detail the implementation of spike propagation in SpikeNET and analyze the implications for biological modeling. We then focus on SpikeNET performance for simulating large-scale model of retinotopically organized groups of neurons and compare SpikeNET's capacities with other event-driven approaches.

2. Methods: SpikeNET

The development of SpikeNET started in our laboratory in 1994 and was initially presented in 1996 [7] in an application to process natural images using large networks of integrate-and-fire neurons. In SpikeNET, spikes propagate in an event-driven way: at each time step, only activated neurons are processed. As with some other event-driven approaches spikes are not propagated immediately but buffered in lists before being propagated [5, 6]. Neurons are organized in retinotopical homogenous maps, which allow the definition of a common basis for the neurons receptive field (RF) that can be shifted depending on the neuron location. These two features form the basis of the computational power of SpikeNET compared to other neural simulator packages.

2.1. Neuronal units

The basic objects in SpikeNET are two-dimensional arrays – that we will call neuronal maps - of relatively simple leaky integrate-and-fire (IF) neurons. Each unit is characterized by a small number of parameters: a membrane potential, a threshold, and (in some cases) a membrane time constant. Using IF units to model neurons' behavior is relevant both from the biological and the computational perspective.

Leaky integrate-and-fire neurons [8] lay between abstract and more detailed biological models. For biological modeling, until recently, only conductance based Hodgkin-Huxley (HH) models were used [9]. The recent regain in interest for IF neurons is due to their simplicity, the fact that they facilitate mathematical analysis of populations of spiking neurons [10], and are also computationally more efficient than HH neurons [11]. Moreover, as a biological modeling tool, IF neurons with minor modifications can model very accurately single compartmental HH neurons [12, 13]. Recently, Jaffe and Carnevale [14] also provided evidence that the dendritic architecture of neurons tends to normalize synaptic integration, so that the single compartment simplification of neurons is not unreasonable.

We will now give a more rigorous definition of IF neurons and describe their implementation in SpikeNET. The membrane potential evolves below the firing threshold θ according to:

$$C\frac{dV}{dt} = -g_l(V - V_l) + I_{syn}(t) \tag{1}$$

where the g_l and V_l parameters are the conductance and the reversal potential of the voltage dependant leak current. I_{syn} is the synaptic current due to the action of other neurons of the network and C the capacitance of the membrane. Whenever the membrane potential V reaches the threshold θ a spike is emitted and V is instantaneously reset to the resting membrane potential.

Under quiet conditions, leak currents make the membrane potential of real neurons converge toward an equilibrium state. Updating the membrane potentials of all neurons at each time step would result in a dramatic increase in computation time – especially when processing millions of neurons - and we would lose the efficiency gained by the event-driven propagation of spikes. An alternate solution is to update membrane potentials each time the neurons are stimulated by a presynaptic spike. In the absence of stimulation (and assuming no sub-threshold dynamics), membrane potential decays do not need to be calculated because they cannot lead to a discharge. Moreover, knowing the membrane time constant $\tau=C/g_l$, the time lag between two incoming spikes allow us to calculate the decrease of the membrane potential V between current time t and time of the last update t_{last}:

$$V(t) = V(t_{last})e^{\frac{-(t-t_{last})}{\tau}} \tag{2}$$

For a given neuron, the synaptic current can be seen as a weighted sum in equation (3) or can incorporate changes in membrane conductance for higher biological plausibility in equation (4):

$$I_{syn}(t) = \sum_{i \in A} W_i f(t - t_{spike}(i)) \tag{3}$$

2

or

$$I_{syn}(t) = -\sum_{i \in A} \overline{g}_i (V - V_i) f(t - t_{spike}(i)) \tag{4}$$

where W_i and $t_{spike}(i)$ are respectively the connection strength with the input neuron and its date of discharge i (A being the set of afferent neurons). V_i and $\overline{g}_i$ are the reversal potential and conductance of the synapse of input neuron i (this formula covers both excitatory and inhibitory synapses). The implementation of f in SpikeNET in its simplest form is a step function being always 0 except at the moment the discharge occurs where it has value 1 during one time step (if there is no current leak in the output neuron, the result in equation (3) is independent of the time step value). For more biological plausibility, f can implement a simple (or double) exponential decay function or more complex non-linear behaviors.

Most neurons are only affected by incoming spikes from their afferents. However, for certain "input" cells, for example for retinal ganglion cell, we determined spike timing by a direct calculation that depends on the stimulus. Electrophysiological studies have shown that X-cell behavior (i.e. parvocellular inputs to the visual system) can be approximated using a simple delay integrate-and-fire neuron where the input current is proportional to the activation value at this location [15]. In the context of retinal ganglion cells, we can perform a local contrast extraction ("Mexican-hat" convolution) on the image, and this value can be used to calculate the latency of the unit's spike - the earliest latencies correspond to those cells for which the value of the convolution is highest, whereas longer latencies correspond to lower activation levels. To discretize this process in time, one has then to sort these activation values to determine the order into which the neurons will discharge. This sorting procedure represents a lot of computing time for large input arrays so we used a pseudo-sorting algorithm by regular sampling of the activation values and using a sampling rate high enough to obtain the exact order.

2.2. Neuronal projection in SpikeNET

Instead of neurons integrating activity within their receptive fields as in standard simulator packages, event-driven propagation of activity means that it is the spiking neurons that update their target neurons. Thus, we need a mechanism to convert these synaptic receptive fields into efferent projections from the presynaptic neurons. In SpikeNET, connections are usually homogeneous, which means that inside a neuronal array map neurons share the same synaptic weights while still processing separate zones of their input space. Then, transforming input synapses into output ones between neuronal array maps of equal sizes this is achieved by mirroring the convolution (figure 1).

The fact that SpikeNET uses homogeneous projections can also be understood from a biological point of view. For example in the mammalian primary visual cortex, it is generally assumed that all the neurons that detect the same orientation at different locations of the visual field have similar properties. Thus, in a computer implementation, it is possible to use the same synaptic pattern of connectivity for all these neurons. On the other hand, if each neuron in a neuronal map has its own receptive field and synaptic weights – in the non-homogeneous receptive field case - we must reconstruct lists of output synapses for each neuron individually. Because homogeneous implementations allow to simulate many more neurons and connections than non-homogeneous ones, we will mainly focus on homogeneous projections between two neuronal maps of the same size. Note that concerning neuronal map size, SpikeNET can actually implement projection between neuronal maps of different size but that in this case, the mechanism that converts RFs into efferent synaptic projections becomes considerably more complicated.
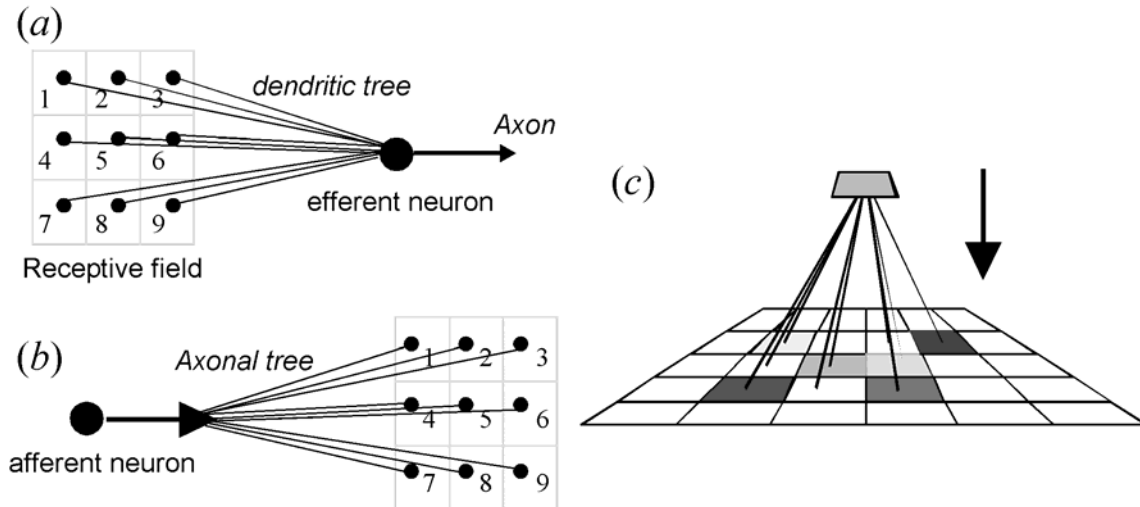
Figure 1. (*a*) Illustration of a homogeneous receptive field for an output neuron. (*b*) Transformation of a RF into a projection field for an input neuron using event-driven propagation. The convolution of synaptic weight must be mirrored. (*c*) Illustration of a spike propagation, the gray levels of the output neurons representing different synaptic weights (taken as random in this case).

We also want to emphasize that, for computational purposes, all zero weights were removed from the projection field. The computation cannot be seen as matrix multiplications but rather as an application of several one-dimensional arrays, each one representing consecutive values in the pattern of synaptic weights along the x-axis. This structure was used (*i*) to speed up computation for non-square receptive fields and (*ii*) to facilitate future development using highly efficient vectored-array computation such as OPENGL, and multimedia instruction sets such as SSE or ALTIVEC.

2.3. Propagation of spikes and Parallel applications

We will now describe the dynamics of spike propagation in the network, which is the core of the algorithm. We also show how the algorithm can be extended for parallel applications using asynchronous communications between several processors, thus preventing most synchronization problems. Similar though not identical approaches have been implemented elsewhere using the Parallel Virtual Machine environment in PGENESIS and NEOSIM [16, 17].

The computation is very simple and only involves transfers of lists of spike. As described previously, neurons are organized in topologically organized two-dimensional arrays – neuronal maps. These maps are organized in a circular list at the beginning of each time step as shown in figure 2 and table 1. Then, every map is processed sequentially checking whether spikes were triggered in its afferent maps at the previous time step. If so, input spikes are propagated in the output maps and output neuron membrane potentials that have been modified are checked to see whether the threshold has been exceeded. Whenever a map finishes its processing, it is removed from the circular list. The time step propagation finishes when the circular list collapses. Figure 2 and table 1 illustrates the propagation in a simple model when one of the neuronal maps is processed on a remote computer.

The propagation is map centered as the processing maps just pick up the information they need – spikes locations – in the other maps. Input maps and network maps are purely passive: they simply store spikes lists to be provided to other processing maps. If the information is not available (i.e. for a network map that has not receive yet its list of spikes from the network) then the processing for this map is skipped but the map remains in the circular list to be processed next turn. In the case of a parallel implementation, accesses from the network are asynchronous to this loop, which means that it could occur at any time and would just suspend the current computation without affecting it.

```
(a) Main loop

For each time step {

Organize all neuronal map pointers circularly (include only maps which have afferent maps projecting onto them).
      Pointer = pointer to the first map in the loop
      Reset all the afferent map processing flags for each map j :
            map[j]->map_already_processed[from 1 to number_of_afferent_maps(j)] = false

N = number of maps circularly organized
While (N>0) {                                                      // test if the circular list is empty
      For i = 1 to number_of_afferent_maps(Pointer)  {             // scan afferent maps to Pointer
            If (available( Pointer->afferent_map[i]->spike_list) and
            (not( Pointer->map_already_processed[i])) ) {          // test if the afferent spike list is available
                  Pointer->process_afferent_spike_list( Pointer->afferent_map[i] ) †† // process the spike list
                  Pointer->map_already_processed[i] = true         // update afferent map flag

            If (all(istrue(Pointer->map_already_processed[1 to end]))) {  // test all afferent map flags
            Remove map pointed by Pointer from the circular map list
                  Pointer = next map pointer in the circular list
                        N = N – 1                                  // decreases the number of pointer in
            }                                                      // the circular list
            }
      }
            Pointer = next map pointer in the circular list        // go to the next map in the circular list
}
}

(b) If several processors, the following code might interrupt processing in (a) asynchronously

If network interruption from remote map j {
Store incoming spike-list in local map j (local map j on the local processor is only a spike list repository map and does not perform any processing)
Make available the spike list for other maps to process (efferent maps)
}
```

Table 1. (*a*) Pseudo-algorithm depicting the algorithm implemented in SpikeNET to process lists of spike. At line ††, note that while the incoming spike lists are being processed, if neurons discharge, they might be added to list of spike of the map that is being processed (in cases where inhibition is present the process is a little bit more complicated). This list of spike will then be processed by other maps at the next time step. The processing of input neuronal maps is not described but simply involves making available their pre-computed list of spikes at each time step. In case of a parallel implementation, the list of spikes might also be dispatched to the network as soon as processing for this map is terminated. (*b*) Asynchronous processing of lists of spike provided by other processors on the network.

   Though it might seem complex at a first glance, this algorithm is actually simple: neuronal maps at each time steps only pick up lists of spikes from their afferents and propagate them. We only organized neuronal maps into a circular list that collapses at each time step in the case of a parallel implementation to allow that processors skip nodes (i.e. input spike lists) that are not ready yet. Also note that intra-map connections are processed exactly the same way as external map connections: because of the object oriented approach, each map processes the different lists of spikes in the same way regardless of their origin.
   Neuronal processing inside a map and construction of lists of spike for each map is performed in a standard fashion. For two neurons i and j, i projecting onto j: when neuron i fires, it is added to the list of spikes for the map it belongs to. At the next time step, the weight of the synapse between neuron i and neuron j is added to the target neuron's potential, and SpikeNET tests to see whether the potential of neuron j has exceeded its threshold. If so, the neuron's potential is reset (by subtracting the threshold) and neuron j is added to the list of neurons that have fired during the current time step for the map it belongs to. Note once more that unlike standard event-driven approaches [3, 5, 6], the propagation of spikes is not fully asynchronous and require at least one time step to occur. We will discuss in section 4 why this approach is more relevant when processing neuronal maps as SpikeNET does.
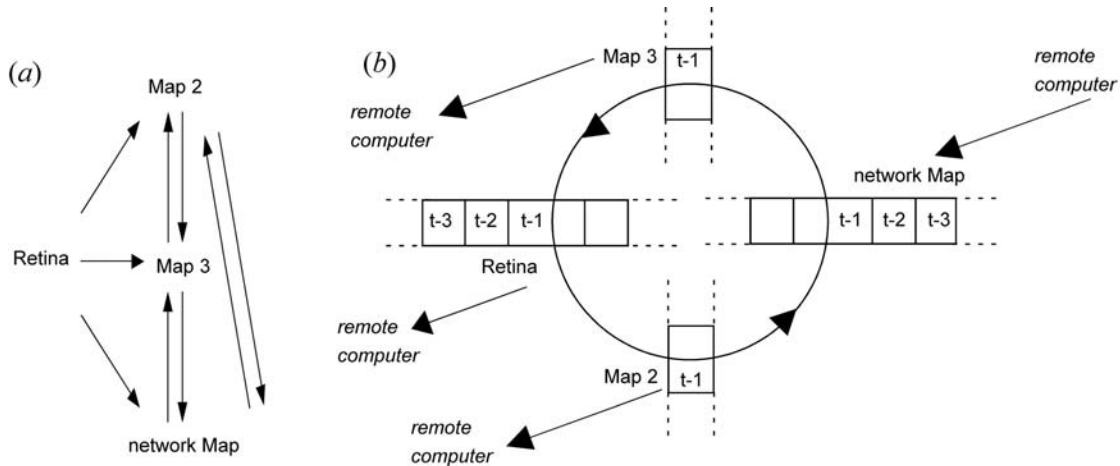
Figure 2. Illustration of SpikeNET algorithm. (*a*) Example of a simple model with 3 local maps "Retina", "Map1" and "Map2" and one remote map named "network Map" processed by another computer. Arrows indicate direction of projections for the neurons in each map. (*b*) Flow of information in a map centered view. At each time step *t*, all the maps are organized in a circular list. Each time box for each map represents a list of spikes (for simplicity we use *dt*=1). At each processing time step, only local "Map2" and "Map3" perform computations. A remote computer that sends its results at each time step provides the list of spikes for the "network Map". This information is received in an asynchronous way by the local processor, meaning that there is no waiting delay associated with the communication. The circular list shrinks as each map completes its computation and is removed from the circular list (network maps do not really do any computation on the local computer but have been inserted in the circular list for better readability). Finally the circular list collapses when all the maps have finished processing their input spikes. At this point the algorithm proceeds to the next time step.

2.4. Learning algorithms

Implementing learning algorithms imposes constraints in SpikeNET: some learning mechanisms might update synaptic weights of a neuron depending on the history of input synaptic spikes to this neuron. For instance, recent neurophysiological studies have shown that synapse plasticity depends on the relative timing between the presynaptic and the postsynaptic spikes [18]. Thus we need to keep the reference of every synapse that has been activated: a convenient way to do that is to keep the reference of the pre-synaptic spikes. By keeping previous presynaptic times of discharge, SpikeNET can easily reconstruct the history of each synapse and update synaptic parameters depending on this history. We have implemented this mechanism in a simulation for the emergence of orientation selectivity in the primary visual cortex [19].

3. Results: performance of SpikeNET

We have implemented various applications using SpikeNET ranging from face detection and recognition in natural images [20, 21] to unsupervised learning in primary visual cortex [19] and motion processing in extrastriate visual areas [22]. SpikeNET has been mainly used to model feed-forward propagation in the visual system, but it is also well suited to simulate any network that can be organized into topological maps. Though not specifically designed for this purpose, SpikeNET can also be used to model randomly connected networks of neurons.

SpikeNET has been designed to be computationally efficient. One of its advantages comes from the efficient use of RAM. Since the number of parameters per neuron is kept low, each neuron can require as little as 16 bytes of memory, depending on the type of numerical precision required. More importantly, the use of shared synaptic weights in homogeneous projections means that one set of weights can be used for all the neurons in an array. The use of shared weights for neuronal maps is not biologically unrealistic: for instance in the visual system, many neurons have similar selectivity at

different positions in the visual field. As a result it is perfectly reasonable to simulate networks with tens of millions of neurons and billions of synapses on standard desktop computers.
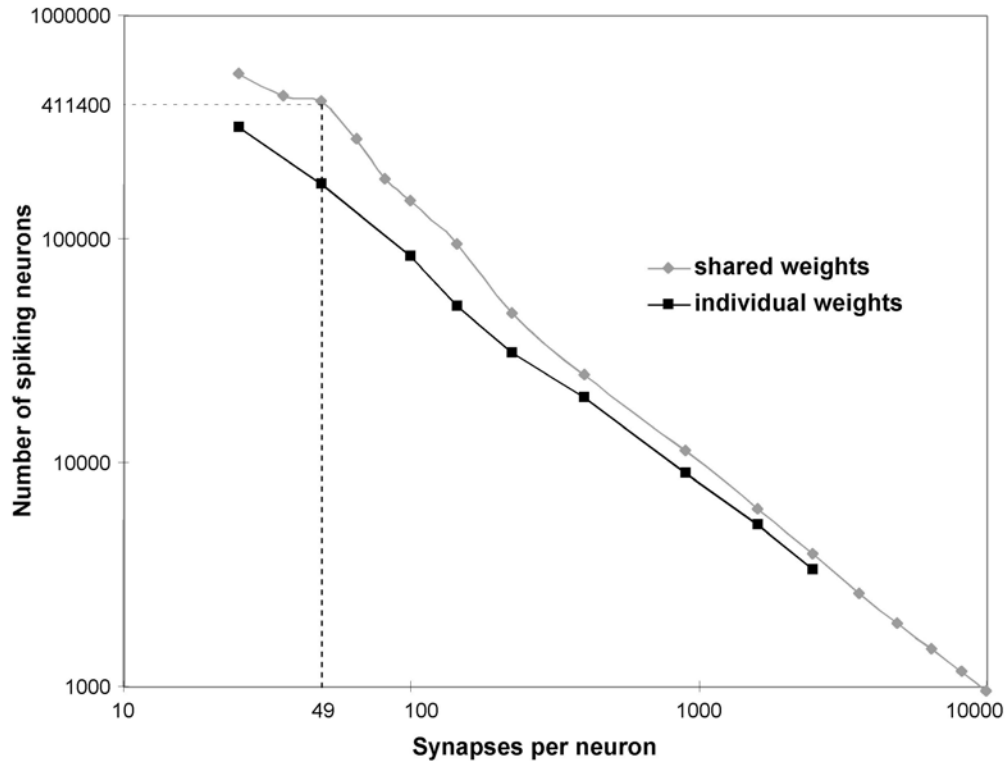

Figure 3. Result of a simulation in real time with a variable number of integrate-and-fire neurons and a variable number of synapses (dt = 1 ms). Neurons implement equation (3) with no decay and the function f simulating fast shunting inhibition as explained in the text. For simplicity, the model is only two layers of neuronal maps of the same size, the first layer emitting spikes and the second one integrating these spikes (for the neuron count, only the second layer is taken into account). We fixed the number of synaptic connection and determined the number of neurons that one can simulate in real time at 1Hz. We considered two conditions, one in which each neuron had a separate set of synaptic weights and another in which every neuron had the same set of synaptic weights but shifted depending on the position of the neuron in the neuronal map (for individual weights, we were not able to go above a certain number of synapses because of the memory limitations of our computer). In both cases the relation is roughly linear, which implies that the computation time only depends on the number of synapses updated. For shared weights, the computation is more efficient and reaches a maximum at 49 synapses per neuron because of the optimal use of cache memory by the processor.

3.1. Real time computation speed

The main advantage of SpikeNET is computation speed and its potential application to real time modeling. For simple integrate-and-fire neurons, SpikeNET can update roughly 20 million connections per second (figure 3), even when using a sensitivity parameter to modulate the effect of each synaptic input (using a standard 266 MHz Macintosh PowerPC 750 processor). This is sufficient to model a network of 400,000 neurons in real time, using a time step of 1 millisecond (assuming 49 connections per neuron, and an average firing rate of 1 spike per second, a value which is a reasonable estimate for the average firing rate of cortical neurons). Note that with a more conventional neural network simulation approach one has to recalculate every unit at every time step, and so the same computational power would only allow 20,000 connections to be calculated per millisecond, which with 49 connections per neuron would limit real-time simulation to around 400 neurons.

Typically, the complexity of the spike propagation algorithm is $\theta(N)$, N being the number of synapses updated per second. As in other event-driven implementations [5, 6], the complexity $\theta(N)$ does not depend on the time step value as long as there are enough synapses updated per time step ($N_{dt} \gg 1$). Increasing the time resolution from 1 ms to 0.1 ms has virtually no effect on computation time, since the number of spikes that are propagated does not change. For a large network of 1 million neurons discharging at 1Hz and connected to 10000 neurons each, 10^{10} connections are processed per second and one can thus get time steps as low as 10^{-8} s (assuming $N_{dt}=100$) without a significant impairment of computation time.

Performance is clearly optimal with shared weights, but even when each neuron has its own set of weights (which obviously increases RAM usage considerably), speed only drops by a factor of around 2. Adding decays using equation (2) to simulate the leaky nature of the synaptic integration process adds roughly 30-40% to the computation time.
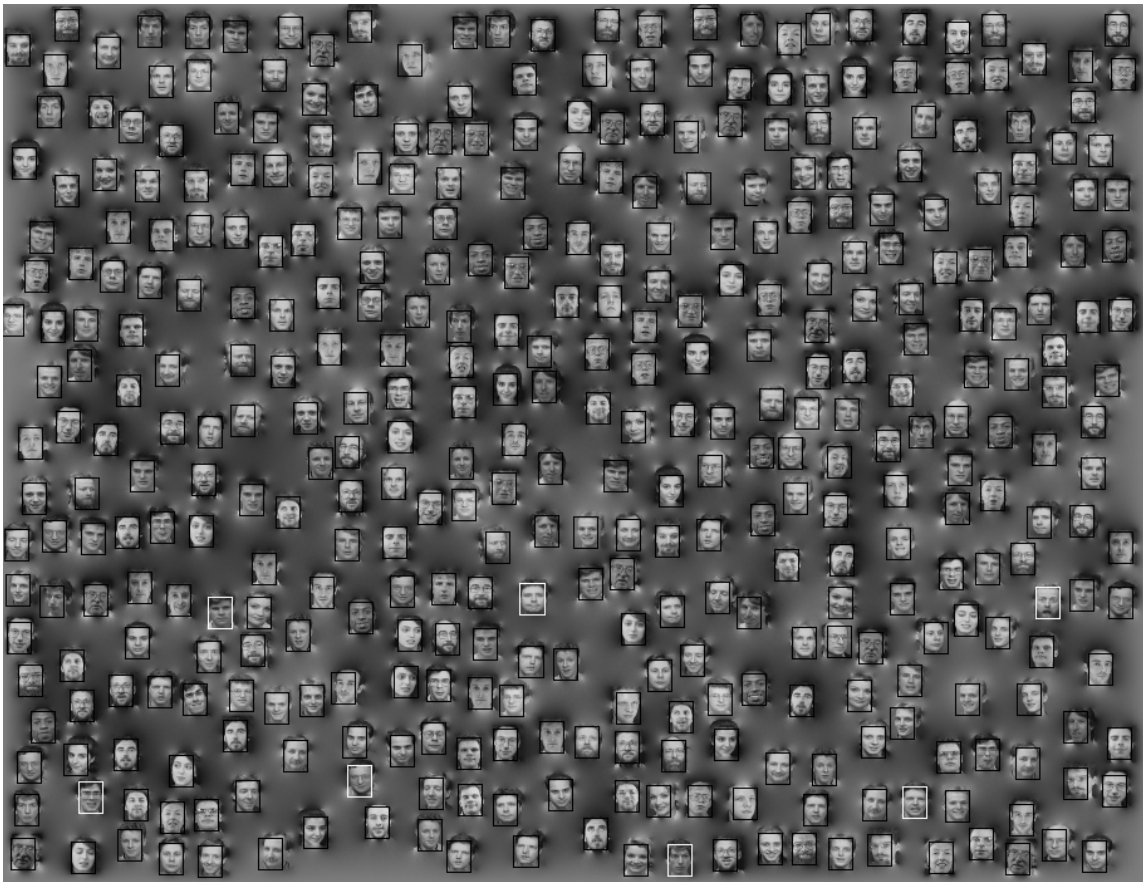

Figure 4. Propagation of a set of 400 face images (belonging either to the learning base or one of the two test bases). We only present the global result of the network superimposed with the input image. The size of the image was 910x700 pixels, which requires a network containing roughly 32 million neurons and 245 billion connections. For a correct detection, a neuron selective to a particular face must discharge within a 4x4 region located at the center of the face. Black squares indicate correct recognition and white ones false recognition. Despite the size of the network, the simulation could be completed in about 30 minutes of CPU time on a modest desktop computer (Macintosh G3, 266 MHz). The network also shows high resistance to noise and image degradation. Adapted from [23].

Rather than entering abstract details about the variation of the performances of SpikeNET as a function of parameter modifications, we present the result of an application in figure 4 [23]. The simulation represents the propagation of a feed-forward wave of spikes in a visual system like architecture containing about 32 million neurons and 245 billion connections (for details about the

simulation see [23]). In comparison, other biological modeling software would have problems to simulate small groups of about 100 000 neurons on supercomputers.

3.2. Parallel application

Although running SpikeNET on a standard desktop machine is already reasonably quick, as described previously, the very nature of SpikeNET makes it an ideal candidate for implementation on parallel hardware. The factor that usually prevents large-scale use of parallel hardware is the amount of communications needed between processors. For many problems, one sees little speed up once the computation has been split between more than 4 or 8 processors. However, with SpikeNET, the only information that needs to be transferred between processors is the lists of spikes. The format used by SpikeNET means that the identity of each neuron which fired can be transmitted using only around 1-2 bytes, and so even a network with 10 million neurons firing at an average of one spike per second could be simulated in real time without saturating the bandwidth of a cluster of processors linked by conventional fast-Ethernet technology at 100 Mbytes per second. We implemented SpikeNET using a cluster of 2 Linux machines. We used the implementation for parallelization that was described in the previous paragraphs, using asynchronous UNIX interruption and TCP/IP to communicate between the processor. In preliminary simulations, we observed typically a 90% efficacy of parallelization using Fast-Ethernet (100% would be perfect parallelization). However higher levels of parallelization need to be assessed in more detail and we are working on developing broadcast techniques such as UDP to minimize the network load.

3.3. Accuracy of the algorithm

Concerning the accuracy of spike integration and membrane potential variation, SpikeNET's error is of the order of the time step value (dt) but it could be made lower with a few modifications. When a neuron fires between time t and $t+dt$, a local error on firing time of order dt is generated because firing occurs systematically at time $t+dt$, assuming no delay in the connection. First it leads to an error on membrane potential at time $t+dt$ and another one on the EPSP delivery date. Hansel [11] proposed that one should interpolate membrane potential variations between time t and dt in order to calculate a better approximation for the date of discharge of neurons. The global error is then of order dt^2. In SpikeNET, the reset of membrane potential of neurons can easily be modified to fit this better approximation.

4. Discussion

We showed how SpikeNET was able to simulate efficiently networks with large number of neurons. Though the simplicity and homogeneity of the algorithm seems appealing, one can question the biological plausibility of using IF neurons to model real neuron behavior. We will try to justify some of the simplifications we made here and also look at possible extensions of the integrate-and-fire approach.

4.1. The IF model is a good approximation for network behavior

Leaky IF neurons cover many of the currently known neural network behaviors. For example, oscillations such as alpha rhythms, similar to those recorded in the brain have been observed in populations of IF neurons [24]. Second, Golomb et al. [25] were even able to find connectivity schemes in which IF neurons behave as bistable units, and may thus form a basis for short-term memory. Finally, a model of hippocampus using IF neurons can behave as an associative memory [26] and converge to stable attractors [27].

While IF neural networks can model macroscopic behavior of real neural systems, they have also been used to simulate the behavior of individual neurons. For instance, leaky IF neurons have been

used to model accurately real neuron spike discharges in the Lateral Geniculate Nucleus [15, 28] and the authors argued that any further simplification of this model was unable to capture the underlying dynamic of spike discharge.

At a low frequency of discharge, IF neurons can be made indistinguishable from HH neurons by including the rough dynamics of voltage dependant channels [13]. Destexhe [12] has also shown that by adding the bimodal intrinsic properties of voltage dependant channels, which become active only if the membrane potential crosses a threshold (and modeling continuous changes of these parameters in real HH models during the action potential), it is possible to mimic real HH dynamics with great accuracy. In comparison, simple IF models are more imprecise because they lack the duration of the action potential. Adding these properties in SpikeNET would be straightforward as the transient activation of channels occurs at the same time as the discharge onset. Following a discharge, the reset of the neuron potential would be calculated depending on the time lag separating the current update with the next one (in a process similar to equation (2)).

The basic cellular model can also be made more interesting by including a sensitivity parameter that modulates the effect of incoming action potentials. We have used this feature to implement the rank-order coding scheme originally proposed by Thorpe [29]. According to this scheme, before each image propagation, the sensitivity parameter is initially fixed at 1.0 and decreases by a fixed percentage with each incoming impulse, resulting in a progressive desensitization of the post-synaptic neuron which can be thought of in terms of fast shunting inhibition in the visual system. This mechanism was implemented in an extended version of equation (3) (for more details see [21, 23]). The net result of this mechanism is that activation is maximal only when the spikes arrive in the order of the weights - with the highest weight synapses being activated first.

Many other parameters could be calculated rapidly including, for example, calcium concentration. Compartmental neurons can also be modeled and updated each time a neuron receives a spike. As long as the solution is deterministic, it can be calculated using the difference in update time between two incoming spikes (using a process similar to equation (2)). If the global analytical solution for a set of differential equations governing the neuron behavior cannot be calculated from the time step difference, the solution must be calculated using derivative methods. Thus between each stimulation of the neuron, we would have to compute the underlying dynamics using standard techniques such as a first order (Euler) or a slightly modified second order (Runge-Kutta) integration algorithm.

As we have seen a limitation of SpikeNET is that all the process in the neurons must be either discharge locked or delayed discharge locked, which mean that spikes cannot be triggered by sub-threshold dynamics. But is it possible to model noisy integrate-and-fire neurons? Real neuron behavior can be modeled accurately either by introducing noise in the neuronal potential and keeping a fixed threshold or by simply introducing noise in the threshold [15]. Thus SpikeNET, as a first approximation, can model noisy neurons by introducing random walk in neuronal thresholds each time neurons are updated. Also, noise can be introduced in the synaptic connections and latency of input spikes. A more detailed study would be necessary to determine to which extent SpikeNET is able to simulate noisy integrate-and-fire neurons accurately.

4.2. Towards implementation of delays and synaptic dynamics

In real neural systems and models, a spike can stimulate efferent neurons with different delays: some synapses being activated just after the spike discharge and some others being activated later. A simple way of implementing such delays is to postpone the propagation of lists of spikes (see also Mattia and Del Guidice [6] who implemented a similar mechanism by delaying individual spikes). In SpikeNET, delays must be discretized in term of time steps but this simplification is not restrictive: as we will see, even a large increase in the number of time steps does not impair the computation speed. Moreover, standard time steps values used in SpikeNET range from 1 ms to 0.01 ms which is on the order of accuracy needed to model these mechanisms. As illustrated in figure 5 delayed propagation would be associated to stored lists of spikes from previous time steps.

Synaptic models can also have complex internal dynamics [30] that go beyond the simple full open or closed synapses. For instance, we might want to implement alpha synapses, whose

conductance decreases exponentially after transient rises. Dynamic synapses might also switch between different states. To implement these mechanisms, we must discretize them in time, the maximum sampling rate being the number of time steps per second. As shown in figure 5, this is equivalent to propagating many EPSPs at different delays, i.e. propagating the same spikes at different times and with different synaptic weights. As for delayed synaptic propagation, standard time steps are of sufficient accuracy for these mechanisms. Delayed propagation might not take additional time to compute but the simulation of complex synaptic dynamics strongly impair computation speed, the speed being virtually be proportional to the number of events in synapses.
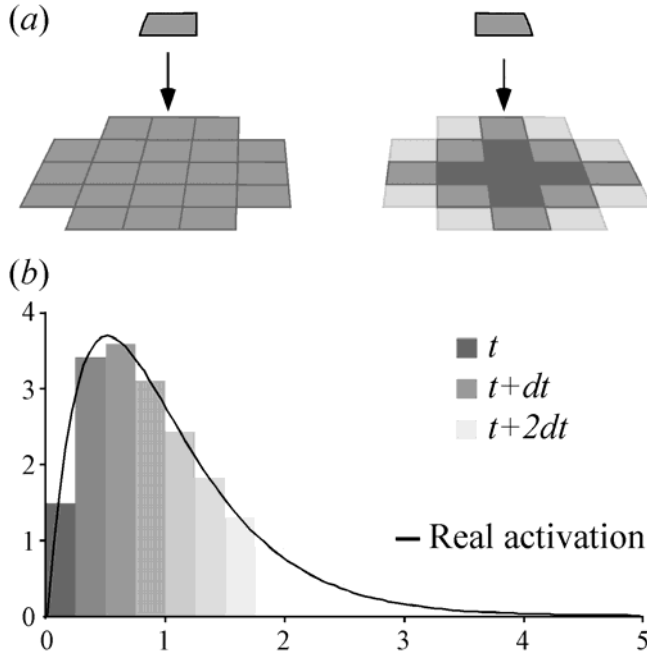

Figure 5. (*a*) Propagation with delayed synapses. Each square represents a neuron and the upper neuron projects toward the lower map of neurons. On the left, when the upper neuron spikes, all the connections are processed at the current time step. On the right however, some synapses will be processed at future time steps, thus introducing a delay in the synapse propagation. (*b*) Complex synaptic dynamics. Propagating a synaptic weight at different delays (gray bars) can approximate synapses with complex conductances (dark curve); brightest bars will be processed later than darkest ones.

4.3. Comparison with other software packages

Comparing SpikeNET to other neural simulation packages is quite hard. Such packages typically either model detailed neuronal mechanism using standard techniques (for a review that is a bit outdated but which still reflects the state of biological modeling software see [31]), or they deal with abstract entities that can hardly be linked to real neuron firing rates. SpikeNET is one of only a few simulators that lie between these two extremes.

Though SpikeNET is, as far as we are aware, the only event-driven software package of its kind available for public download on the internet [32], event-driven approaches have been developed elsewhere [3, 5, 6]. Compared to other event-driven approaches, the map-centered processing technique we used prevents most of problems inherent to synchronization of parallel program flows. The alternate solution implemented by others would have been to propagate list of spikes in an event-driven way: as soon as lists of spikes would be generated they would be propagated through the network. Lists of spikes would be organized in a stack filled initially with input lists of spikes. Each time a map generates a list of spike, the spikes are piled up into the stack to be propagated and popped out when they have touched all their targets. While this standard event-driven approach may be efficient to process individual neuron activity, it may not be optimal for neuronal maps where all neurons of each map are supposed to be updated in a synchronous fashion. When a map is connected to many others, to which map should the spikes be propagated first? The other problem with this implementation would be that two maps with recurrent excitations might indefinitely excite each other thus blocking the computation and one must implement some controls to counteract these side effect behaviors. For parallel applications, a purely event-driven approach can also generate strong

synchronization problems even in the case of small groups of less than 100 neurons (Graßmann, personal communication).

The map-centered processing solution we developed holds other advantages for parallel applications. First, processing on cluster of processors preserves the structure of the single processor propagation; computation in cluster of processors is implemented using a new kind of map – network repository maps that provide lists of spikes just as an input map would do - so the main circular list remains blind to this minor change (figure 2). Because of the limited bandwidth between clusters of processors, spike propagation must be uniformly distributed in time. It is actually the case with the map-centered solution we implemented: because of the sequential processing of maps, lists of spikes are sent continuously to the network in an asynchronous way. Finally, this structure helps storing lists of spike from previous time steps that we will use for learning and biological modeling purposes.

Another difference between SpikeNET and other neural network packages concerns the scale of the simulations. For instance, simulation sizes in SpikeNET are an order of magnitude larger than those used in other event-driven approaches [3, 5, 6]. In other event-driven approaches, the authors aimed at modeling detailed networks of intermediate size. For instance the network described by [6] contained 1200 neurons for a total of at most 22.5 million connections. In figure 4, using a computer with similar processing power, we presented a network of simpler units using 32 million neurons and 245 billion synaptic connections [23]. Unlike other neural network packages, the memory organization and processing in SpikeNET make it suitable to compute and visualize results with very large neural networks.

Because of their computational power and their biological relevancy, large networks of integrate-and-fire neurons simulating using event-driven computation are likely to be popular for future understanding of the behavior of real neural networks. Assuming that the internal neuron dynamic is deterministic, we have showed how such an implementation could be done and have highlighted some of the benefits of using it. We believe our approach, only based on time labeled lists of spikes is much more simple and straightforward than others. The use of homogeneous connections and neuronal maps has also allowed us to scale up the network to millions of neurons and billions of connections, far beyond what is currently achievable by any other neural network package.

Although primarily designed as a tool for modeling biological neural networks, the level of performance obtained with SpikeNET is such that in a variety of tasks, processing architectures developed using SpikeNET can perform at least as well and in many cases substantially better than more conventional image processing techniques [23]. The commercial company SpikeNET-technology (www.spikenet-technology.com) has produced a highly optimized version of SpikeNet specifically designed for image processing applications that differs in a number of ways from the version described here. In particular, the commercial version lacks the open architecture of the publicly available version, making it difficult to use it to investigate novel network architectures and connection patterns. On the other hand, because the commercial version uses a particular set of pre-defined processing strategies, it has been possible to use specific computational techniques to go well beyond the sorts of performance levels described here. Nevertheless, both versions share many of the same underlying features, relying on relatively sparse spiking and event-driven simulation techniques to allow simulation of very large networks of highly interconnected arrays of neural processing elements.

References

[1]   M. L. Hines and N. T. Carnevale, "The NEURON simulation environment," *Neural Comput*, vol. 9, pp. 1179-209, 1997.

[2]   J. M. Bower and D. Beeman, *The book of GENESIS: Exploring realistic neural models with the GEneral SImulation System.*, Second Edition ed: Springer-Verlag, New York, 1998.

[3]   L. Watts, "Event-driven simulation of networks of spiking neurons," in *Advances in Neural Information Processing Systems*, vol. 6. San Mateo, CA: Morgan Kaufmann Publishers, 1994, pp. 927-934.

[4]   A. Delorme, J. Gautrais, R. VanRullen, and S. J. Thorpe, "SpikeNET: A simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26-27, pp. 989-996, 1999.

[5]   C. Grassmann and J. K. Anlauf, "Fast digital simulation of spiking neural networks and neuromorphic integration with SPIKELAB," *Int J Neural Syst*, vol. 9, pp. 473-8., 1999.

[6]   M. Mattia and P. Del Guidice, "Efficient Event-Driven Simmulation of Large Networks of Spiking Neurons and Dynamical Synapses," *Neural Comput*, vol. 12, pp. 2305-2329, 2000.

[7]   S. Thorpe and J. Gautrais, "Rapid Visual Processing using Spike Asynchrony," in *Neural Information Processing System 1996*, vol. 9, M. J. Michael M., Thomas P., Ed.: MIT Press, 1997, pp. 901-908.

[8]   L. Lapicque, "Recherches quantitatives sur l'éxitation électrique des nerfs traté comme une polarisation," *J. Physiol. Pathol. Gen.*, vol. 9, pp. 620-635, 1907.

[9]   A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation nerve.," *J. Physiol. (London)*, vol. 117, pp. 500-544, 1952.

[10]   W. Gerstner, "Population dynamics of spiking neurons: fast transients, asynchronous states, and locking," *Neural Comput*, vol. 12, pp. 43-89, 2000.

[11]   D. Hansel, G. Mato, C. Meunier, and L. Neltner, "On numerical simulations of integrate-and-fire neural networks," *Neural Comput*, vol. 10, pp. 467-83, 1998.

[12]   A. Destexhe, "Conductance-based integrate-and-fire models," *Neural Comput*, vol. 9, pp. 503-14, 1997.

[13]   W. M. Kistler, W. Gerstner, and J. van Hemmen, "Reduction of the Hodgkin-Huxley equations to a single-variable threshold model.," *Neural Computation*, vol. 9, pp. 1015-1045, 1997.

[14]   D. B. Jaffe and N. T. Carnevale, "Passive normalization of synaptic integration influenced by dendritic architecture," *J Neurophysiol*, vol. 82, pp. 3268-85, 1999.

[15]   D. S. Reich, J. D. Victor, and B. W. Knight, "The power ratio and the interval map: spiking models and extracellular recordings," *J Neurosci*, vol. 18, pp. 10090-104, 1998.

[16]   N. Goddard and G. Hood, "Parallel Genesis for Large Scale Modeling," in *Computational Neuroscience: Trends in Research 1997*: Plenum Publishing, NY, 1997, pp. 911-917.

[17]   N. Goddard, G. Hood, F. Howell, M. Hines, and E. De Schutter, "NEOSIM: portable large.scale plug and play modelling," *Neurocomputing*, vol. 38, pp. 1657-1661, 2001.

[18]   H. Markram, J. Lubke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs," *Science*, vol. 275, pp. 213-5, 1997.

[19]   A. Delorme, L. Perinnet, and S. Thorpe, "Network of integrate-and-fire neurons using Rank Order Coding B: spike timing dependant plasticity and emergence of orientation selectivity," *Neurocomputing*, vol. 38-40, pp. 539-545, 2001.

[20]   A. Delorme, R. Van Rullen, and S. J. Thorpe, "Rapid object recognition based on asynchronous feed-forward processing," presented at 22th European Conference on Visual Perception, Trieste (Italy), 1999.

[21]   R. Van Rullen, J. Gautrais, A. Delorme, and S. Thorpe, "Face processing using one spike per neurone," *Biosystems*, vol. 48, pp. 229-39, 1998.

[22]   W. Paquier, A. Delorme, and S. Thorpe, "Motion processing using one spike per neuron," presented at Ninth Annual Computational Neuroscience Meeting, Brugge, 2000.

[23]   A. Delorme and S. Thorpe, "Face recognition using one spike per neuron: resistance to image degradation," *Neural Network*, vol. in press, 2001.

[24]   D. T. Liley, D. M. Alexander, J. J. Wright, and M. D. Aldous, "Alpha rhythm emerges from large-scale networks of realistically coupled multicompartmental model cortical neurons," *Network*, vol. 10, pp. 79-92, 1999.

[25]   D. Golomb and G. B. Ermentrout, "Continuous and lurching traveling pulses in neuronal networks with delay and spatially decaying connectivity," *Proc Natl Acad Sci U S A*, vol. 96, pp. 13480-5, 1999.

[26]   A. Samsonovich and B. L. McNaughton, "Path integration and cognitive mapping in a continuous attractor neural network model," *J Neurosci*, vol. 17, pp. 5900-20, 1997.

[27]   J. J. Hopfield and A. V. Herz, "Rapid local synchronization of action potentials: toward computation with coupled integrate-and-fire neurons," *Proc Natl Acad Sci U S A*, vol. 92, pp. 6655-62, 1995.

[28]   D. S. Reich, J. D. Victor, B. W. Knight, T. Ozaki, and E. Kaplan, "Response variability and timing precision of neuronal spike trains in vivo," *J Neurophysiol*, vol. 77, pp. 2836-41, 1997.

[29]   S. J. Thorpe, "Spike arrival times : a highly efficient coding scheme for neural networks," in *Parallel processing in neural systems*, R. Eckmiller, G. Hartman, and G. Hauske, Eds. Amsterdam: Elsevier, 1990.

[30]   M. Tsodyks, K. Pawelzik, and H. Markram, "Neural networks with dynamic synapses," *Neural Comput*, vol. 10, pp. 821-35, 1998.

[31]   E. De Schutter, "A consumer guide to neuronal modeling software," *Trends in Neurosciences*, vol. 15, pp. 462-464, 1992.

[32]   A. Delorme and S. Thorpe, "SpikeNET: a An Event-driven Simulation Package for Modeling Large Networks of Spiking Neurons," World Wide Web publication ed, 2002.