A&QT-R 2004 (THETA 14)
2004 IEEE-TTTC - International Conference on Automation, Quality and Testing, Robotics
May 13 – 15, 2004, Cluj-Napoca, Romania

PRINCIPLES OF DESIGN FOR LARGE-SCALE NEURAL SIMULATORS

Raul C. Mureşan[1,2], Iosif Ignat[1]

1. Technical University of Cluj-Napoca,
Faculty of Automation and Computer Science, Gh. Baritiu 26-28
2. Nivis Research, Gh. Bilascu 85
3400 Cluj-Napoca, ROMANIA
raul.muresan@nivis.com, iosif.ignat@cs.utcluj.ro

Abstract

We present a review of design principles that are to be used for large-scale neural simulators. This paper emphasizes the most important problems encountered in the simulation of biologically plausible neural systems and provides some solutions derived from modern simulation techniques. We stress upon the idea that a modern simulator should be able to perform generic simulations of as many as possible neural architectures, with various neural models. At the same time the amount of processing effort should be tunable, with the possibility of using various simulation methods simultaneously (e.g. iterative and event-driven). The neuroscientist should be able to use different types of electrophysiological models with complete inter-operability. We provide an example of a neural simulator that complies with modern design guidelines: "The Neocortex Simulation Environment".

Key words: Neuron, synapse, simulator, large-scale simulation, neural spike.

1. INTRODUCTION

As the field of theoretical neuroscience develops and the electrophysiological evidence accumulate, researchers need more and more efficient computational tools for the study of neural systems. In cognitive neuroscience and neuro-informatics the modeling of neural systems is the essential approach for understanding the complex processing performed by the brain. The most recent trend in these fields is to use spike-based or detailed models of neurons and detailed (or quasi-detailed) models for synaptic transmission.

During the past five decades, the most frequently used types of artificial neural networks have been the perceptron-based models. However, the biological relevance of such models is very limited, the perceptron being a phenomenological model of the neuron. It has long been accepted that neurons encode information in the rate of their firing, which has led to rate-based models such as the perceptron. However, more recently, scientists began to understand that individual spikes of neurons could play a very important role in the encoding of information in the brain [15]. Moreover, millisecond processes triggered by individual spikes shape the synaptic transmission efficacy, in a spike time dependent (STDP) way [6].

The modeling of detailed electrophysiological phenomena seems to be essential for the understanding of brain processes. At the same time, models inspired from biol-

ogy prove to be very efficient in various types of applications like image processing [12] and automated robot control [2]. Getting an insight on sensory-motor processing from biological systems could inspire robot designs and future bio-mimetic technologies.

Within this context, neural simulation seems to be a more and more important aspect in neuroscience and artificial intelligence. The detailed and quasi-detailed spiking models are a great deal different from the models of multi-layer perceptrons [9]. Also, recent trends in research emphasize the fact that brains are by their nature, very large dynamical systems. Implicitly, the more realistic the simulations are, the more complex the neural model becomes. Designing such "large-scale" simulation capable tools is a non-trivial task [13].

2. SIMULATION BOTTLENECKS

In general, when analyzing the performance of neural simulators there are at least three criteria of evaluation:

- simulation speed;
- required computational resources (memory);
- the accuracy of simulation.

These three aspects are usually in conflict, since high-speed requires simplified modeling which reduces the accuracy of simulation. At the same time, accurate models use many variables, thus increasing the amount of required computational resources. Many detailed neural models, like the Hodgkin-Huxley model, are based on systems of differential equations, which are solved by numerical, iterative integration. The complexity of the mathematical model influences the computational effort of simulation. Unfortunately, the most accurate models are also the worst to simulate in terms of speed and computational resources required. A review on this topic has been published by Izhikevich [7].

The simulation speed is mainly influenced by the complexity of the mathematical models that are to be employed. Consequently, increasing the processing power (CPU performance), usually increases the speed of simulation. Speed becomes an issue for large-scale systems, since biologically relevant models may require days or even months to simulate on desktop computers [8].

There is also an overhead due to "data dispersion" throughout the memory and the required caching time. Usually, there is no possible optimization on the mathematical model that, once chosen, has to be implemented. However, when designing the simulation environment, one should keep in mind that computers today make extensive use of caching mechanisms. Carefully choosing the data structures and representations of neurons and synapses can dramatically improve performance.

The required computational resources usually refer to the amount of memory that the simulation requires. Detailed models are based on differential systems that are to be solved numerically, thus, the overhead of memory per neuron / synapse is greater. However, it turns out that the main problem comes not from the representation of neurons (and their state), but from the representation of synapses, that normally exceed the number of neurons by a factor of $10^3 – 10^4$ [11].

The accuracy of simulation depends not only on the time steps chosen for the integration of the differential equations but also on the complexity of the model. There are many models of neuronal dynamics, with different degrees of biological plausibility. The Hodgkin-Huxley model is one of the oldest and well studied detailed models in

neuroscience [3]. It is also very difficult to simulate because of its inherent complexity. At the other end, there are largely simplified models, like the "integrate-and-fire model" which unfortunately loose much of the biological relevance. However, recently, there have been developed intermediate models, with acceptable low complexity but still rich in behavior [5].

3. HOW TO OVERCOME PROBLEMS?

In this section we focus on possible solutions to the problems presented previously. We start by enumerating some objectives that modern designs have to accomplish:

- A modern simulator has to allow the implementation of *various models of neurons and synapses*. In this way, the modeler can employ different types of models for different levels of a neural architecture. For example, one may use integrate-and-fire neurons and linear synaptic transmission for sensorial modules and FitzHugh-Nagumo or Morris-Lecar [3] model for deeper brain structures such as modeling of the inferotemporal area. Such techniques can be really valuable for reducing the computational effort on incipient, well studied levels of processing, and concentrating the computing power on the deeper, more complex processing modules (that take place in the higher areas of the brain). At the same time, models with simplified input layers can take advantage of the convergent nature of neural processing, at least for the hierarchical paradigms of processing in the visual system (see the hierarchical models of Reisenhuber & Poggio [14]). In order to build such a flexible modeling system, it is obvious that we need some form of grouping of different classes of neurons. A good design solution is to use neural maps.
- Depending on the models that are chosen for the dynamics of the neuron and the synapse, it should possible to implement *iterative* or *event-driven simulations*. For detailed models, with equations that cannot be analytically solved, numerical integration is the only solution (e.g. the Euler method). Is such cases, iterative simulation with fixed time steps is usually required. However, more simplified models, like the integrate-and-fire (IF) model, can be described analytically allowing one to compute a future state of the neuron, given that no stimulation has occurred until that time:

$u_{Tf} = u_r + (u_{Ti} - u_r) * e^{\frac{-(Tf - Ti)}{\tau}}$, where u_{Tf} is the membrane potential of the IF neuron to be computed (final state), u_r is the resting potential (typically -65 mV, or 0 mV in shifted models), u_{Ti} is the initial time when the neuron's state was last computed (initial state), *Ti* and *Tf* are the initial and final time values, and τ is the membrane's time constant of decay. Again, this formula is only applicable if no stimulation has been recorded during *Ti -> Tf*.

Having this in mind, a designer might postpone the update of the neuron's state until it is stimulated again. In this way, a stimulation event on a dendrite triggers the update of the neuron's state. This is called event-driven simulation because the updates in the system are performed on spike-events. Event-driven simulations are very efficient, in sparse spiking models especially, allowing for the simulation of large networks in real-time [1, 13]. Since in many areas of the brain, the above baseline spike density is small, neural simulators can take advantage of this sparseness, and employ spike-driven updates. As an example, for the same neural architecture, using only integrate-and-fire neurons, the event-driven simulation can be as much as 500 times faster than its iterative counterpart. We have to mention that the applicability of event-driven simulations

is limited not only by the neuron's model but also by the synaptic model. As a general rule, any model that allows for the computing of both the synapse's and neuron's state from an arbitrarily previous state, given no stimulation event has occurred since the previous state, can be simulated in an event-driven fashion. The updates are only necessary when a stimulation event appears.

- Modern simulators should be able to take advantage of both *explicit* and *implicit synaptic representations*. Explicit synapses have the advantage of being able to store their state independently, for exactly one pair of neurons. However, when the number of synapses is high, the available memory becomes an issue. A way around this is to represent stereotyped receptive-field profiles, as implicit synapses. For example, the retinotopy of the lower visual system can be exploited in modeling [13]. Instead of defining a synapse for each pair of neurons, one might define a rule of connectivity between two maps, as an implicit connection pattern. This is called s*ynaptic sharing*. Each time the value of a synapse is required, a "correspondence algorithm" determines which synapse from the implicit pattern is to be accessed, based on the positions of the afferent end efferent neurons in the pair.

By using implicit synaptic representations, where it is possible, the required storage size is much reduced, up to thousands of times, eventually leading to linear memory consumption [13]. However, implicit synaptic representation assumes that all neurons are connected in the same way to their afferents, an assumption that is usually false.

- The design has to be able to handle both *homogenous* and *heterogeneous* architectures. Within the same system, there might be not only different models of neurons but also different types of synapses, with different properties (excitatory, inhibitory, simple, STDP, etc). In order to avoid redundancy, classes of properties should be created and the representations of neurons and synapses should use a minimum amount of information, like a pointer to the neural / synaptic class, for example. Implementing classes with different properties allows the modeler to study heterogeneous architectures and also avoid the great redundancy that would arise if the properties would be stored in the representation of the neuron or the synapse. The synapse, for example, should only represent the most common data that all synapses share, the detailed electrophysiological description being moved in a property class object, which is referred by the instance of the synapse.
- A good design should allow for a high degree of *interoperability* between different types of models and different simulation techniques (iterative or event-driven). There should be a uniform low level representation for both neurons and synapses, that could provide the compatibility between different neuron and synapse models. For example, a researcher may use integrate-and-fire and Hodgkin-Huxley neurons within the same architecture, depending on the level of detail that is required. At the same time he might use static or dynamic synapses [10], excitatory or inhibitory, with or without STDP.

Sometimes, it is a good idea to simulate the maps of IF neurons using the event-driven technique and at the same time some more complex maps in an iterative way. The system has to provide a complete interoperability between the two maps although the simulation techniques are quite different. This can be done by using a global synchronization clock.

As for the solutions to the simulation bottlenecks, there are a few hints for the design of neural simulators which help the implementer overcome most of the problems.

To increase the **speed** of simulation:

- implement multiple neuron models and use the simplest ones where possible (for example in input modules);
- allow for both event-driven and iterative simulations to be performed at the same time and use the event-driven as much as possible;
- design your data in order to take advantage of the caching mechanisms.

To reduce the **required computational resources (memory)**:

- use implicit synaptic patterns wherever possible;
- separate the representation of neural and synaptic properties from the representation of the dynamical state (for example, put the resting potential, the threshold, the time constants in a single shared object - property class - and represent only the dynamical state - membrane potential - for each neuron).
- use the simplest possible models for non-critical simulation modules (like the input).

To increase **accuracy**:

- try to employ the most detailed electrophysiological model that is sufficient for your purposes, but isolate it only in critical modules (that are essential for the results of the simulation).
- use the smallest time steps possible for the integration of the model equations but keep in mind that too small steps might not always increase the accuracy but they will for sure slow down the simulation. Find a trade-off.

We designed and implemented a neural simulator that makes use of all these modern principles of design, called "The Neocortex Simulation Environment". It is implemented in Visual C++ and it is completely object-oriented. The "Neocortex" simulator extensively uses inline function expansions in order to increase the speed and avoids, as much as possible, virtual functions and class inheritage.

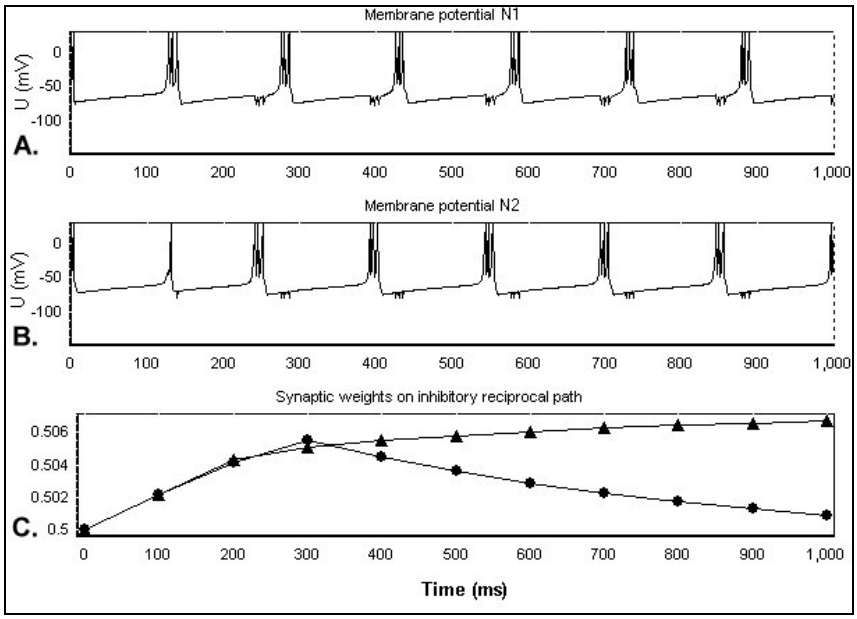

Fig. 1. A "Central Pattern Generator" simulated with "Neocortex". A., B. Membrane potentials (U) of neuron #1 and neuron #2 respectively. C. The release probability of the reciprocal STDP synapses.

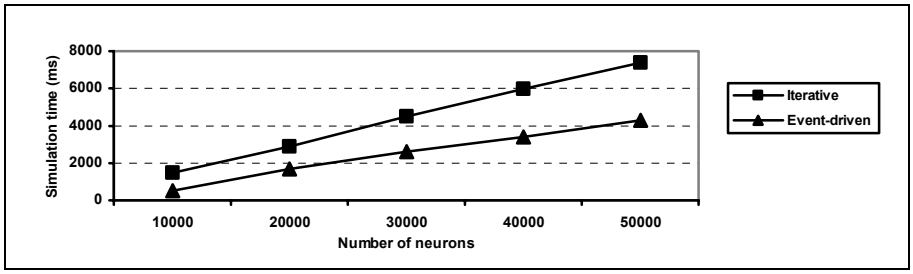

Fig. 2. The scalability of "Neocortex" for IF neurons. Spike injection density is 10% firings / millisecond.

4. CONCLUSIONS

Although the simulation of large populations of neurons is quite a challenging problem, modern design principles can guide both the designer and the modeler to overcome most of the inherent difficulties. Using event-driven simulations and implicit synaptic patterns can speed up simulations up to hundreds and thousands of times. We call these techniques "acceleration techniques".

There are also cases when the modeler cannot take advantage of acceleration techniques that are available with a neural simulator. In such situations, it seems like the only improvement can come from the evolution of hardware or the usage of expensive supercomputers. However, recent advances in research begin to unravel the extraordinary power of hardware implementation for neural systems [4]. We expect a rapid evolution for neuro-hardware in the near future.

5. REFERENCES

1. Delorme A., et al. (1999), SpikeNET: A simulator for modeling large networks of integrate and fire neurons. *In J. M. Bower (Ed.), Computational neuroscience: Trends in research 1999, Neurocomputing*, vols. 26–27, pp. 989–996.
2. Di Paolo E.A. (2002), Evolving spike-timing dependent plasticity for robot control, *EPSRC/BBSRC International Workshop: Biologically inspired Robotics*, The Legacy of W. Grey Walter, WGW'2002.
3. Gerstner W., Kistler W.M. (2002), Spiking Neuron Models: Single Neurons, Populations, Plasticity, Cambridge University Press, New York.
4. Grassmann C., Schoenauer T., Wolff C. (2002), PCNN Neurocomputers - Event Driven and Parallel Architectures, *ESANN '2002 proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium)*, 24-26 April 2002, d-side publisher, pp. 331-336.
5. Izhikevich E.M. (2003), Simple Model of Spiking Neurons, *IEEE Transactions on Neural Networks*, 14, pp. 1569- 1572
6. Izhikevich E.M., Desai N.S. (2003), Relating STDP to BCM, *Neural Computation*, 15, pp1511–1523.
7. Izhikevich E.M. (2004), Which Model to Use for Cortical Spiking Neurons? *IEEE Transactions on Neural Networks*, (special issue on temporal coding), in press.
8. Izhikevich E.M., Gally J.A., Edelman G.M. (2004), Spike-Timing Dynamics of Neuronal Groups, *Cerebral Cortex*, in press.
9. Maass W. (2002), Paradigms for computing with spiking neurons. In: *J. L. van Hemmen, J. D. Cowan, and E. Domany eds., Models of Neural Networks. Early Vision and Attention*, Springer (New York), vol. 4, chapter 9, pp. 373-402.
10. Maass W., Markram H. (2002). Synapses as dynamic memory buffers. *Neural Networks*, 15, pp. 155-161.
11. Mattia M., Del Giudice P. (2000), Efficient Event-Driven Simulation of Large Networks of Spiking Neurons and Dynamical Synapses, *Neural Computation*, 12, pp. 2305–2329.
12. Mureşan R.C. (2002), Complex Object Recognition Using a Biologically Plausible Neural Model, *in: Advances in Simulation, Systems Theory and Systems Engineering*, WSEAS Press: Athens, pp. 163-168.
13. Mureşan R.C. (2003), RetinotopicNET: An Efficient Simulator for Retinotopic Visual Architectures, *European Symposium on Artificial Neural Networks*, Bruges, April 23-25, pp. 247-254.
14. Riesenhuber M., Poggio T. (1999), Hierarchical models of object recognition in cortex, *Nature Neuroscience,* vol. 2, no. 11, pp. 1019-1025.
15. Thorpe S.J., Delorme A., Van Rullen R. (2001), Spike-based strategies for rapid processing, *Neural Networks*, 14, 715-725.