ELSEVIER

Neurocomputing 32–33 (2000) 1095–1102

NEUROCOMPUTING

www.elsevier.com/locate/neucom

PARALLEL NEUROSYS: A system for the simulation of very large networks of biologically accurate neurons on parallel computers

Peter Pacheco[a,*], Marcelo Camperi[b], Toshi Uchino[a]

[a]*Computer Science Department, University of San Francisco, 2130 Fulton St., San Francisco, CA 94117, USA*

[b]*Physics Department, University of San Francisco, 2130 Fulton St., San Francisco, CA 94117, USA*

Accepted 13 January 2000

Abstract

We present a software package for the simulation of very large neuronal networks on parallel computers. The package can be run on any system with an implementation of the Message Passing Interface standard. We also present some example results for a simple neuronal model in networks of up to a quarter of a million neurons. The full software package as well as usage and installation guidelines can be found in [http://cs.usfca.edu/neurosys]. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Very large networks; Parallel computers; MPI; Clusters

1. Introduction

Since the pioneering research of Hodgkin, Huxley, and Katz, work has continued on improving single-neuron models and on arranging these models into networks of increasing sophistication. However, computational power often severely limits the complexity of the neuronal model one can study or the effective number of neurons that can be included in a network. Multi-compartment neuron models that attempt to account for many known physiological aspects of a neuron can require hundreds of equations. Furthermore, even when the neuron models are kept simple, they still may

* Corresponding author. Tel.: + 1-415-422-6630; fax: + 1-415-422-5800.
E-mail address: peter@usfca.edu (P. Pacheco).

PII: S0925-2312(00)00283-6

require from 2 to around 10 differential equations. Thus, trying to model a large network of such neurons may prove to be a formidable computational task.

A great deal can be learned from single-cell modeling or using networks with very few simple model neurons. Indeed, great insight into dynamical behavior can be gathered with just two neurons. However, sometimes very large systems are needed, as there may be emergent complex behaviors in large networks without a small network counterpart. Moreover, statistical properties of model firing rates and synaptic transmission may not be representative if models are drastically scaled down.

Thus, it is clear that the need to go beyond the capabilities of existing simulations is real and pressing. Given the nature of the problem and some obvious qualitative and "philosophical" connections between networks of neurons and parallel computers, it is almost inevitable to conclude that programming on parallel systems is the natural solution for large-scale neuronal network simulations.

In this work we present an outgrowth of a project started in the University of San Francisco's Applied Mathematics Research Laboratory. The goal of the project was to produce powerful, portable, modular, and readily accessible software. The program only uses freely available software libraries and runs on commonly available hardware. A user only has to input the desired neuronal model (in the form of a set of differential equations and initial conditions) and the structure of the network, without having to learn anything beyond the handling of ASCII files and some rudiments of C syntax. The output of the system is an animated depiction of the network, which can show, by suitable use of colors, time-dependent changes in such variables as membrane voltages and synaptic currents. The system obtains parallelism through use of the Message Passing Interface or MPI standard, and it is well suited to run on low-cost, parallel computers built from commodity hardware.

In the remainder of this paper we briefly discuss the Message Passing Interface standard, commodity clusters, the system, and some results we have obtained which illustrate the importance of simulating large networks and the scalability of the system.

2. The message passing interface standard for parallel computers

A program written in ANSI C will compile and run on any computer with an ANSI compiler, despite possibly enormous differences among hardware systems. The Message Passing Interface (MPI) Forum [1,2] wanted to achieve something similar for parallel programs. They developed a standard for parallel programming by specifying a library of functions that can be called from C or Fortran programs. The foundation of this library is a group of functions that can be used to achieve parallelism by message passing, the transmitting of data from one process to another by using "send" and "receive" functions. The MPI functions take care of the interprocess sharing and distribution of data, and several other issues, such as synchronization of processors. A program written using the MPI standard will run on any parallel hardware with an MPI implementation installed — regardless of underlying

architecture. Furthermore, there are several free MPI implementations that can be installed on a vast range of hardware [3,8].

3. Commodity supercomputers

With recent advances in high-speed networks and improved microprocessor performance, it has become possible to obtain extremely high performance from commodity hardware at a small fraction of the cost required for commercial systems, such as the Cray T3E or the SGI Origin 2000. Authors distinguish between two types of systems: networks of workstations (NOWs) [6] and Beowulf clusters [7]. Both are built by interconnecting a collection of inexpensive computers using a relatively inexpensive network such as 10- or 100-Megabit ethernet. The principal distinction is that in a NOW, the computers, or nodes, are accessible individually to users for general, sequential tasks, and parallel programs "scavenge" available CPU cycles. The different nodes do not need to have any special software installed and may physically reside anywhere. A user starts a "cluster-aware" process from a central node, which has parallel software (such as MPI) installed and a list of IP addresses of the workstations in the NOW. On the other hand, the nodes of a Beowulf cluster are dedicated to the parallel system: a user only accesses them if he or she is running a parallel program. Further, the Beowulf interconnection network is isolated: the network is used exclusively for loading programs onto the nodes and communication among the nodes of running parallel programs. Thus, using off-the-shelf hardware and specialized (but freely available) software, NOWs and Beowulf clusters provide institutions and users with the means to achieve a powerful and relatively inexpensive parallel-computing platform.

4. The system

User input to the system consists of two main components:

- A Hodgkin–Huxley style model for single-cell and synaptic dynamics in the form of a set of differential equations written in C. This includes a set of model parameter values.
- A directed graph describing the interconnection among neurons in the form of an adjacency list. This is an ASCII file listing the neurons and their interconnections.

The directed graph is automatically partitioned among the available processors by assigning each processor the same number of neurons. After the network is partitioned, each processor builds a system of differential equations involving the variables representing the neurons in its subnetwork and synaptic currents from neurons that interact with neurons in its subnetwork. Then, a solver computes the solution. Currently we use a parallel solver based on the fourth-order Runge–Kutta method. Depending on the host computer capabilities, the solution can be stored in a single file or it can be distributed across multiple files (one per processor).

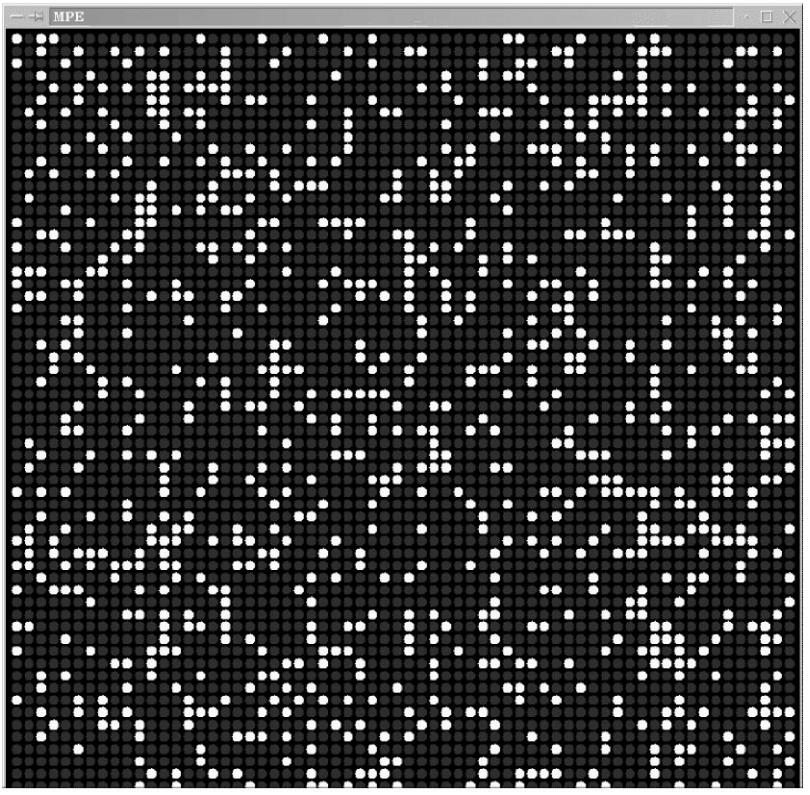

Fig. 1. A typical "Neurondiz" display output.

In either case, the visualization software takes the adjacency list and the output from the solver, displays the network and, by varying the colors of the nodes and edges of the graph, gives an animated depiction of its time-dependent behavior (see Fig. 1). The visualization system employs a master-slave model. One processor is devoted to managing the display, while the remaining processors compute its contents. The visualization software currently uses the MPE library of functions, which is distributed with the freely available MPICH [3] implementation of the MPI library. In order to insure portability and high-performance, all of the code is written in ANSI C using the MPI-1 library of communication functions. Input/output is currently carried out with the standard I/O libraries in C.

For further details on the design and use of the program, see [9].

5. Some example results

In this section we present a couple of examples. The first illustrates the importance of network size in statistical analyses. The second shows the scalability of the software.

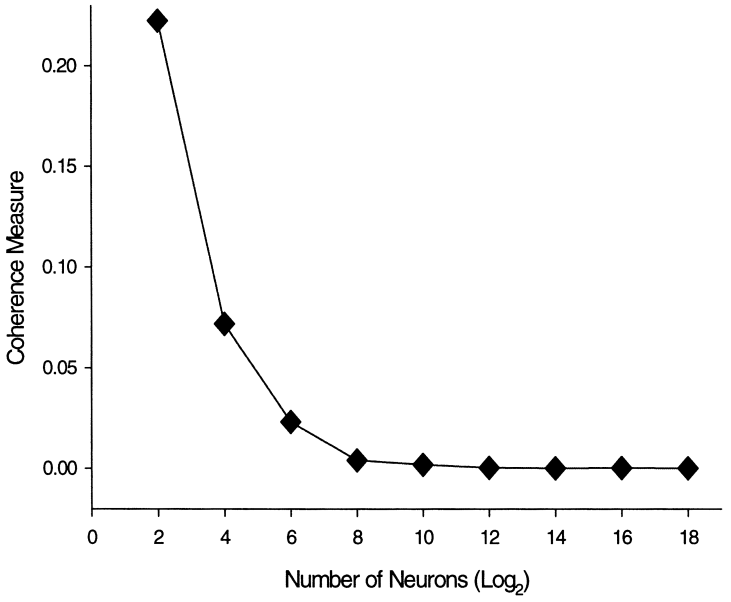

Fig. 2. Degree of network synchrony as a function of the number of neurons for otherwise similar networks. Large coherence measures determined for small networks may not be representative of systems' behaviors.

As our test example, we used the single-cell model and parameter values found in [4], following the usual Hodgkin–Huxley dynamics. Moreover, we assumed the synaptic gating variable s follows first-order kinetics:

$$\frac{ds}{dt} = F(V)(1 - s) - \frac{s}{\tau_s},$$

where $F(V)$ is a sigmoidal function. The total synaptic input drive into a neuron is given by

$$I_{syn} = g_s S_{tot}(V - V_S),$$

where

$$S_{tot} = \frac{1}{N} \sum \varepsilon_j s_j.$$

Here N is the number of neurons, the sum is over incident neurons, and $\varepsilon_j = \pm 1$, depending on whether the incident synaptic current is excitatory or inhibitory.

The neurons were arranged in a square grid, and the interconnections were generated using a random number generator: the probability that any two neurons are interconnected is proportional to a decaying exponential function of the distance

between the two neurons. In order to reduce communication overhead, any interconnections joining neurons belonging to non-consecutive processors are deleted.

Computations for this paper were carried out on USF's NOW. It consists of 16 dual-processor Pentium II computers connected by a fast-ethernet switch. The nodes run Linux, a free version of the Unix operating system. Both the MPICH [3] and LAM [8] versions of MPI are installed on the systems.

In the first test, we measured the network coherence of networks containing from 2 to 2^{18} neurons. The coherence, Σ_N, measures the network's degree of synchrony as a function of the number of neurons, N. It was suggested by Hansel et al., in [5], and is defined as,

$$\Sigma_N = \frac{\langle A_N(t)^2\rangle_t - \langle A_N(t)\rangle_t^2}{1/N\sum_{i=1}^{N}[\langle V_i(t)^2\rangle_t - \langle V_i(t)\rangle_t^2]} \quad \text{with } A_N(t) = \frac{1}{N}\sum_{i=1}^{N} V_i(t).$$

were $\langle\rangle_t$ is time average and $V_i(t)$ is the time-dependent membrane voltage for neuron i. Results of the simulations are given in Fig. 2. These suggest that large coherence measures determined for small networks may not be representative of large networks with the same neuron models and similar interconnection structures.

The following chart illustrates the scalability of the differential equation solver running on the USF NOW with the MPICH implementation of MPI (Times are in seconds).

Neurons / Processors	1024	4096	16384	65536	262144
1	496	2636	10870	37120	—
2	257	1404	6119	21367	—
4	135	638	3107	10664	—
8	76	319	1511	5371	22570
16	43	160	694	2648	11269
32	29	85	327	1241	—

Another measure of parallel performance is the *efficiency*. This is the ratio of the runtime on one processor to the product of the number of processors multiplied by the parallel runtime:

$$E(N, p) = \frac{T_{\text{serial}}(N)}{pT_{\text{parallel}}(N, p)},$$

where N is the number of neurons, and p is the number of processors. Fig. 3 shows that for systems with more than 4096 neurons, the solver maintained efficiencies well above 0.75 and surprisingly, for very large systems, efficiencies actually improved as the number of processors was increased.

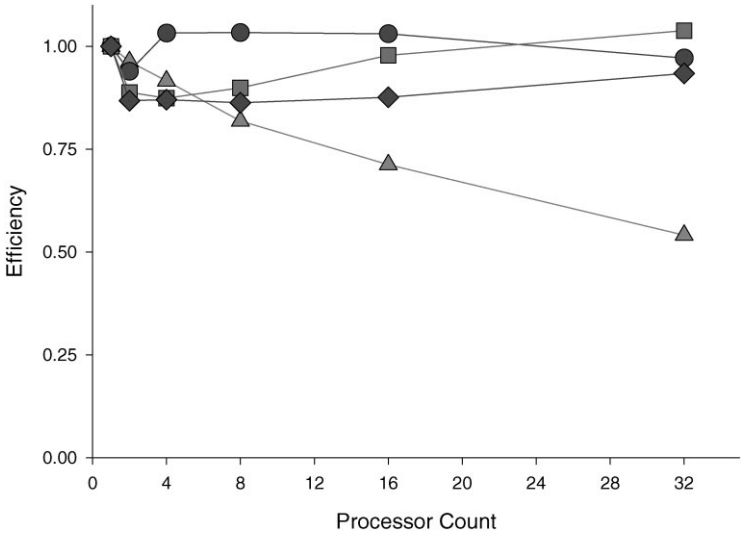

Fig. 3. Efficiency of a run as a function of the number of processors used (triangles = 1 k neurons, circles = 4 k, squares = 16 k, and diamonds = 64 k neurons, respectively).

6. Conclusions and directions for future work

Our preliminary results suggest that Parallel Neurosys will be very useful in the study of large networks of biologically accurate neurons. Thus we plan to continue both development of the system and to use it in the study of large networks of a variety of neuronal models.

Specific improvements planned for future systems include an improved user interface for both the differential equation solver and the visualization program. We also plan to look into the use of other methods for solving the differential equations, better automatic partitioning of neurons among processors, and the use of the MPI-2 I/O functions.

Acknowledgements

We thank the National Partnership for Advanced Computational Infrastructure for a grant of computer time on the Cray T3E at the San Diego Supercomputer Center.

References

[1] http://www.mpi-forum.org.
[2] P. Pacheco, Parallel Programming with MPI, Morgan Kaufmann, San Francisco, 1997.

[3] http:// www.mcs.anl.gov/mpi.
[4] J. White et al., J. Comput. Neurosci. 5 (1998) 5–16.
[5] D. Hansel et al., Neural Comput. 10 (1998) 467–483.
[6] http://www.rdt.monash.edu.au/ ~ rajkumar/tfcc.
[7] http://www.beowulf.org.
[8] http://www.mpi.nd.edu/lam.
[9] http://cs.usfca.edu/neurosys/.

Peter Pacheco is associate professor of Mathematics and associate professor and chair of Computer Science at the University of San Francisco. He is the author of the book, *Parallel Programming with MPI*, and he is the co-author of papers on the use of parallel computing in circuit simulation. He has worked on parallel programs in linear algebra, computational chemistry, and speech recognition. He received his Ph.D. in Mathematics from Florida State University.

Marcelo Camperi is an assistant professor of physics at the University of San Francisco. He received his Ph.D. in Physics from Boston University for Mathematical Physics. He works in theoretical neuroscience and computational physics. His current interests include simulating very large networks on parallel computers and dynamical mechanisms for working memory in the prefrontal cortex.

Toshi Uchino is a graduate student in computer science at the University of San Francisco. He received a BA in English in Tokyo Japan, and another BA in Psychology from USF. He is currently writing his master's thesis on simulations of very large neural networks on parallel computing systems. He is planning to pursue his interests in cognitive/neuroscience.