Parallel and Distributed Simulation of Large Biological Neural Networks with NEST

Jochen Martin Eppler[1,2,3] **Abigail Morrison**[2,3]
Markus Diesmann[2,3] **Hans Ekkehard Plesser**[4]
Marc-Oliver Gewaltig[1]

[1] Honda Research Institute Europe GmbH, Offenbach, Germany
[2] Bernstein Center for Computational Neuroscience, Albert-Ludwigs University, Freiburg, Germany
[3] Computational Neurophysics, Institute for Biology III, Albert-Ludwigs University, Freiburg, Germany
[4] Dept. of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, Norway

1 Introduction

In Computational Neuroscience, simulations are an important tool to investigate and answer questions which are not tractable by experimental or theoretical methods. The neural simulation tool NEST is a simulation environment for large heterogeneous networks of point neurons [3] or neurons with a small number of compartments [2]. It is aimed at simulations of large neural systems, i.e. networks with more than 10^4 neurons and 10^7 to 10^9 synapses. NEST is implemented in C++, using an object-oriented approach. It is extensible in several ways and users can add their own model neurons and analysis functions. Starting with its first incarnation in 1995 [1], NEST has been in constant use and development. Since 1999, NEST has been developed and maintained by the NEST Initiative (see `www.nest-initiative.org`), which also provides public releases under an open-source type license. If employed on multi-processor or multi-core computers, NEST can use multiple threads to speed up the simulation of the network, however, it is not possible to distribute the load of a simulation across a cluster of computers.

In this contribution, we describe two major developments: first, the combination of parallel and distributed simulation. Second, the introduction of a more flexible connection and communication framework. In a companion contribution, we present the main features of NEST from a user's perspective (Gewaltig and Diesmann).

2 Parallel and Distributed Simulation

The current version of NEST uses threads to perform the network update in parallel on computers with multiple processors. Threads have the advantage that parallel execution takes place within the same process such that each thread has access to the same data. However, since a multi-threaded process is bound to a single computer, the network size is limited by the amount of available memory. Distributing the simulation across several computers offers a solution to this problem, since each additional computer not only contributes computing power, but also its own memory [5]. However, several new problems arise which have to be addressed:

1. The use of multiple threads if a computer offers more than one processor or core.
2. The even partitioning of the network across the different computers.
3. The higher latency and slower communication between neurons on different computers.

The first two points are easily solved by introducing the concept of local and remote threads, and counting their indices continuously across all machines. In a network representation that builds on this view it is not necessary to distinguish between the distributed and the parallel case. The third problem, communication, can be ameliorated by following the scheme outlined in [5] to reduce the transmitted data as much as possible, so that communication costs are small with respect to computation costs.

Neural Network Representation

NEST represents the network as list of nodes, which are either neuron models, devices or sub-networks. Device nodes are created for each thread on each computer in order to allow parallel data access and avoid bottlenecks during simulation. This is particularly important for nodes which have to deliver large amounts of events to their targets (e.g. random spike generators). Neuron nodes are only instantiated on a single thread to achieve a distribution of the load across all computers. On all other threads a light-weight proxy node is created as representative. The thread on which the actual node is created is determined by a modulo operation that depends on the index of the node and the overall number of processes and threads. The proxy node for a neuron serves as a local anchor for a connection which would otherwise join neurons on two separate threads. In this way, from the perspective of an individual thread, all connections appear to be local, thus minimising cache problems. Figure 1 gives a simple example of how this scheme applies to a small network of five nodes using two processes with two threads each.

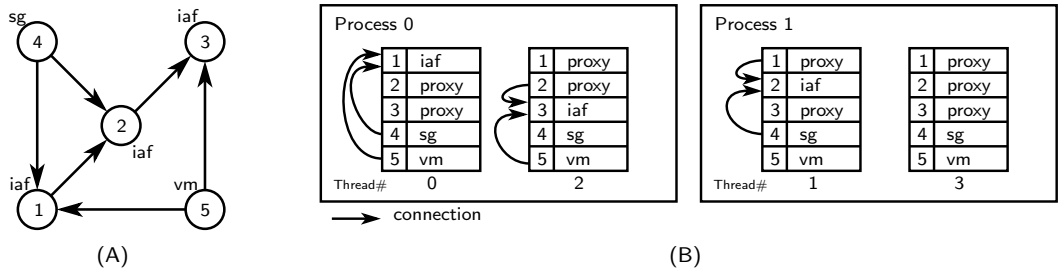

Figure 1: (A) Simple network with a spike generator (sg), three integrate-and-fire neurons (iaf) and a voltmeter (vm); (B) The same network distributed onto two processes with two threads each. Only connections local to a thread are realized. The two devices (sg, vm) are replicated for each thread, the neurons (iaf) are distributed across threads.

Communication

Communication between processes is carried out with the help of a library implementing the message passing interface standard [4]. MPI offers an abstracted and portable layer for sending and receiving data on a wide variety of platforms. Communication is performed by a single thread on each computer, thus avoiding the need for a thread-aware implementation of MPI.

3 Connection Framework

Previously all connections were represented by connection lists in the nodes, containing the target, the weight and the delay for each connection. Due to this static layout, simulations of learning synapses were difficult to implement. To resolve these restrictions, we have implemented a new connection framework, which uses synapse prototypes to accommodate different kinds of synaptic dynamics, functional plasticity and learning. In the new system, the connection information is stored separately from the nodes. This permits parallel event delivery on multi-processor computers. Moreover, it is possible to compress the connection data by storing common parameters only once. An $A \rightarrow B$ connection in the new system is realized on the unique computer on which neuron B is instantiated; on all the other computers, where B is represented by a proxy node, the connection is ignored. Thus the distribution of the nodes leads to a distribution of the connection information in a natural way. The result of a simple neural network set up this way is illustrated in figure 1.

4 Scheduling

The neural network is updated on a time grid with fixed spacing in cycles of length $d_{\min}$, the minimal synaptic propagation delay in the network. At the beginning of each cycle, all events from the last cycle are delivered to their local target nodes. This happens on all computers in parallel. The state of the network is then advanced to the next simulation time step. Events that are generated during the cycle are buffered. At the end of the cycle, the buffers are communicated between the computers. Finally, the network time is advanced by $d_{\min}$. This loop is repeated until the simulation time specified has elapsed.

5 Summary and Conclusions

In this work we have presented some important steps to make NEST more suitable for modern simulation requirements:

- Distributed computing techniques can be combined with multi-threading to produce a single application capable of exploiting the architecture of clusters, shared-memory systems and multi-core processors.
- A network representation with a strict separation of memory suitable for distributed computing also improves the cache effectivity in the multi-threaded case.
- The new connection framework extends the scope of the simulation tool, incorporating plasticity and the usage of different synapse types to build a greater variety of networks.

Preliminary results indicate that linear scaling of the simulation time with respect to the number of processors can be achieved.

6 Acknowledgments

Partially funded by DAAD 313-PPP-N4-lk and BMBF Grant 01GQ0420 to the Bernstein Center for Computational Neuroscience Freiburg

References

[1] Diesmann, M., Gewaltig, M.-O., & Aertsen, A. (1995). SYNOD: an environment for neural systems simulations. Language interface and tutorial. Technical Report GC-AA-/95-3, Weizmann Institute of Science, The Grodetsky Center for Research of Higher Brain Functions, Israel.

[2] Larkum, M. E., Zhu, J. J., & Sakmann, B. (2001). Dendritic mechanisms underlying the coupling of the dendritic with the axonal action potential initiation zone of adult rat layer 5 pyramidal neurons. *J. Physiol. (Lond) 533.2*, 447–466.

[3] MacGregor, R. J. (1987). *Neural and Brain Modeling*. San Diego: Academic Press.

[4] Message Passing Interface Forum (1994). MPI: A message-passing interface standard. Technical Report UT-CS-94-230.

[5] 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.