Parallel and distributed simulation of large biological neural networks with NEST

Jochen Martin Eppler[1,2,3], Abigail Morrison[2,3], Markus Diesmann[2,3,4], Hans Ekkehard Plesser[5], Marc-Oliver Gewaltig[1]

eppler@biologie.uni-freiburg.de, abigail@biologie.uni-freiburg.de, diesmann@brain.riken.jp, hans.ekkehard.plesser@umb.no, marc-oliver.gewaltig@honda-ri.de

1
Honda Research Institute Europe GmbH
Carl-Legien-Str. 30
D-63073 Offenbach
http://www.honda-ri.de

2
Bernstein Center for Computational Neuroscience
Hansastraße 9a
D-79104 Freiburg
http://www.bccn.uni-freiburg.de

3
Computational Neurophysics, Institute for Biology III
Albert-Ludwigs-Universität
D-79104 Freiburg
http://www.brainworks.uni-freiburg.de

4
Brain Science Institute
RIKEN
Wako
Japan

5
Dept. of Mathematical Sciences and Technology
Norwegian University of Life Sciences
N-1432 Ås
http://www.umb.no

- NEST is an environment for the simulation of large neural systems
- We present three major improvements:
  1. Support for multi-core processors, SMP machines and computer clusters
  2. Support for different models of connections, including Hebbian learning, spike-timing dependent plasticity and short-term dynamics
  3. Increased performance and better scaling

Introduction

The Neural Simulation Tool NEST is a simulation environment for large networks of point neurons or neurons with a small number of electrical compartments. *Large* here means more than 10^4 neurons and 10^7 synapses. It is implemented as an efficient C++ library and provides a command line user interface with its own simulation language, SLI.

In previous work (e.g. [1]) we focused mainly on the simulation technology used in NEST 1.0. In this contribution we present two major developments that improve both its flexibility and its performance:

1. The combination of techniques for multi-threaded and distributed simulation
2. A framework for heterogeneous synapses.

The first two columns of this poster show these improvements in detail. The third column contains the results of benchmark tests, which were performed on different architectures.

The innovations described here will be incorporated into NEST 2.0, which will be released later this year.

In a companion contribution (49, Exploring large-scale models of neural systems with the Neural Simulation Tool NEST) we demonstrate NEST from a user perspective.

References

[1] M. Diesmann and M.-O. Gewaltig. NEST: An environment for neural systems simulations. In T. Plesser & V. Macho (Eds.), Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001, Volume 58 of GWDG-Bericht, pp. 43–70. Göttingen: Ges. für Wiss. Datenverarbeitung, 2002.

[2] A. Morrison, C. Mehring, T. Geisel, A. Aertsen and M. Diesmann. Advancing the boundaries of high connectivity network simulation with distributed computing. Neural Comput. 17 (8), 1776–1801, 2005.

[3] N. Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. J. Comput. Neurosci. 8 (3), 183–208, 2000.

Partially funded by DAAD 313-PPP-N4-lk, EU Grant 15879 (FACETS), and BMBF Grant 01GQ0420 to the Bernstein Center for Computational Neuroscience Freiburg

Multi-threaded and distributed simulation

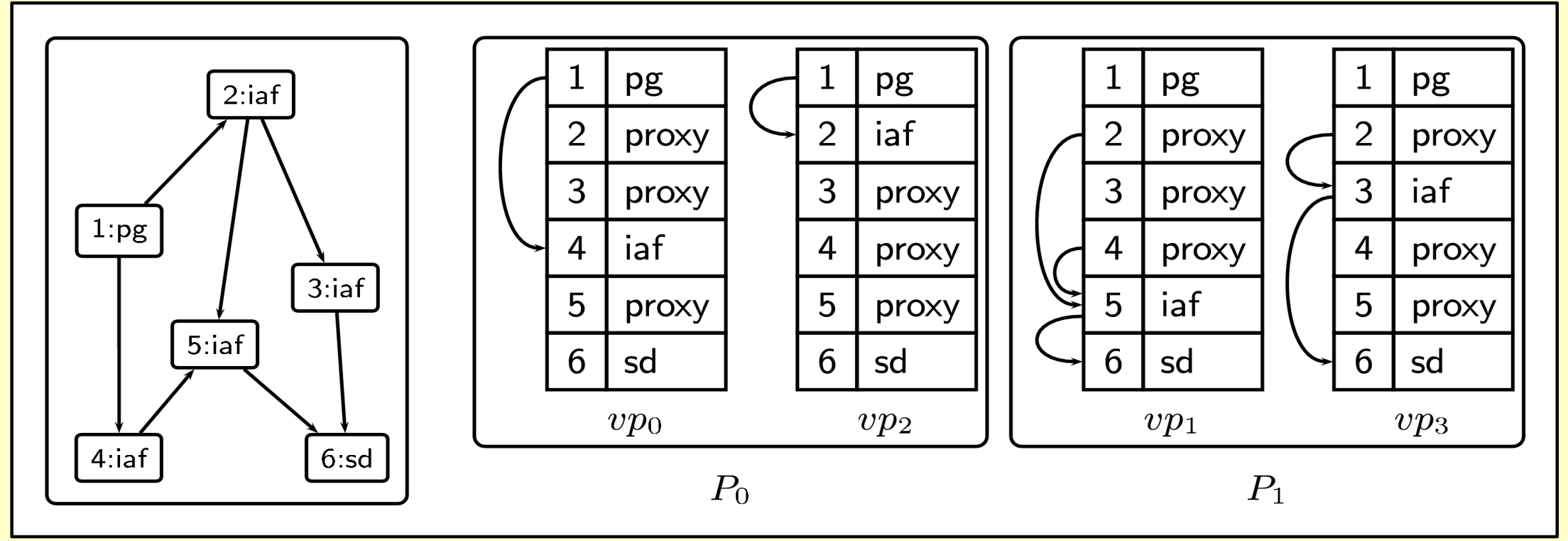
On multi processor machines, NEST currently uses threads to update the network in parallel. As threads are executed in a single process and thus bound to a single computer, the network size is limited by the amount of locally available memory. We solve this problem by distributing the network across multiple computers. Communication between them is carried out using the Message Passing Interface (MPI). However, several new problems arise:

1. An even partitioning of the network across different computers has to be found
2. The higher latency and slower communication between neurons on different computers has to be taken into account

We solve the first point by introducing the concept of local and remote threads (*virtual processes*), on which the network elements are distributed in the following way:

- Neurons are assigned to a single virtual process, on all others a proxy node is created
- The ID of a neuron's virtual process is calculated by a modulo operation
- Devices (e.g. voltmeters, spike generators) are replicated for each virtual process
- Connections between elements are only established on virtual processes, where the post-synaptic node is not a proxy
- The virtual processes (vp_i) are distributed equally onto the real processes (P_i) according to a modulo operation

The following figure shows this scheme applied to a small example network, which is distributed onto four virtual processes:



pg = poisson spike generator, iaf = integrate-and-fire neuron, sd = spike detector. The commands to build the network are shown in the next column

The improved network representation offers a direct solution to the communication problem (2.), which is described in [2]:

- We only transmit the IDs of neurons that spiked. The events are reconstructed on the machine of the post-synaptic neuron.
- We communicate in intervals of the minimal synaptic delay in the network. This is sufficient, because the elements are functionally decoupled during this time.

In the multi-threaded mode, the network is still constructed serially. In distributed mode, NEST constructs the network in parallel, which reduces the simulation time.

Flexible synapse management

In NEST 1.0, connections were characterized by the static parameters *sender*, *receiver*, *weight* and *delay*. This made the simulation of plasticity and learning rather difficult. We have now implemented a framework that provides different types of synapses, currently

- Static synapses with static connection parameters
- Short term dynamics for synaptic depression and facilitation
- Synapses for spike-timing dependent plasticity.

Below, we give a short example of the connection facilities in NEST's simulation language SLI:

```
① poisson_generator Create
  iaf_neuron 4 CreateMany
  spike_detector Create

② << /weight 0.3 >> SetSynapseDefaults

③ 1 [2 4] DivergentConnect
  [2 4] 5 ConvergentConnect

④ 2 3 0.5 1.0 Connect

⑤ stdp_synapse SetSynapseContext
  [5 3] 6 ConvergentConnect
```

The syntax of SLI is such that arguments are given before the function that uses them. Built-in functions are printed in bold face.

① The `Create` command and its variants create neural elements as well as device nodes.

② `SetSynapseDefaults` sets the default values for all subsequent connections.

③ Many varieties of the `Connect` command are available to realize different wiring structures and topologies. If no additional parameters are given, the default values are used.

④ It is also possible to specify weight and delay directly in the call to `Connect`.

⑤ To set the default synapse type for following connections, the command `SetSynapseContext` is used.

New synapse types can be created from existing ones by changing their parameters and storing them with a new name. These are then also available to `SetSynapseContext`.

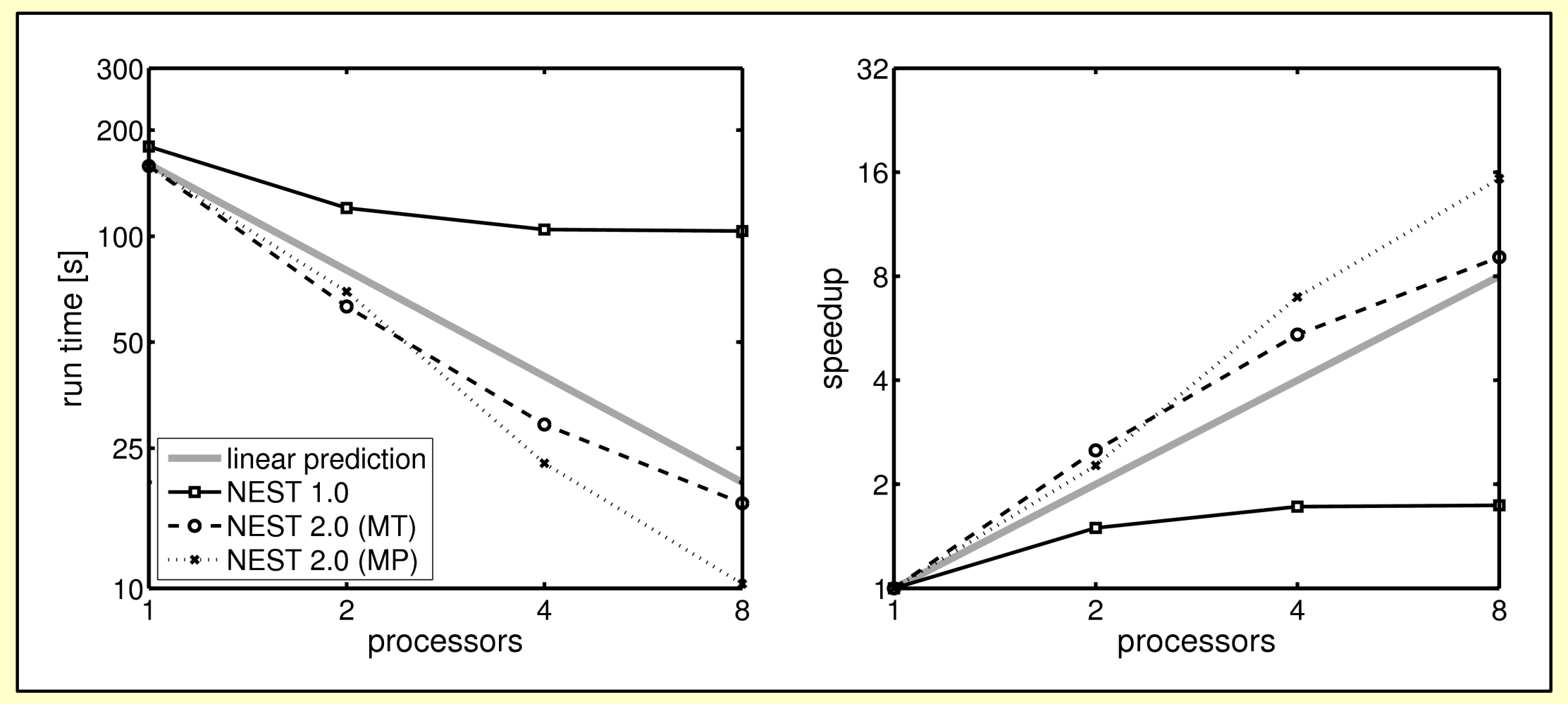
The memory requirements of neural network models are often dominated by the space for the synaptic connections. To reduce that fraction, we have implemented different methods for compressing connection data ([2]) by storing redundant parameters only once.

The different synapse types are implemented in a modular fashion so that users can implement their own models.

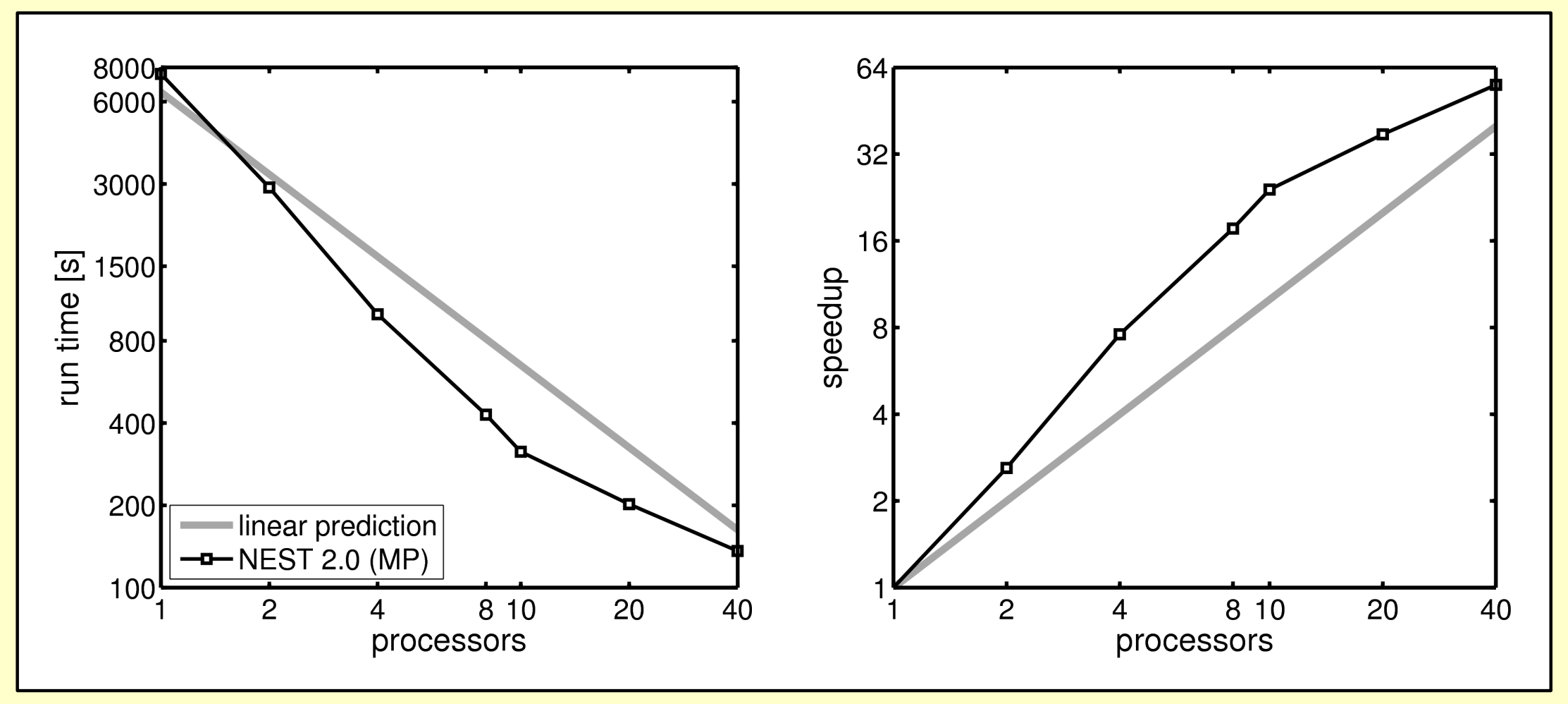
Performance and scaling

The improved network representation optimizes the performance of NEST.

Below are the simulation times of a network with 12,500 neurons and 1,000 random connections each (described in [3]).



Performance of the simulation on a SMP machine (Sun Fire V40z, 4 Dual Core AMD Opteron 875, 2.2 GHz, 1 MB cache). MT = multi-threaded, MP = distributed with message-passing. The left panel shows the absolute run times with respect to the used number of processors. MT uses a single process with varied number of threads, in MP, the number of processes is varied, with one thread in each process. The right panel shows the speedup, given by T_1/T_N, where T_1 is the run time of the serial version, T_N the run time of the parallel version with N processors.



The same network as above simulated on a computer cluster (20x2 AMD Opteron 250, 2.4 Ghz, 1 MB cache, Dolphin/Scali interconnect).

Both figures show that NEST scales better than linear, which can be explained by optimal cache utilization ([3]). However, the multi-threaded simulation still suffers from cache problems.

⇒ Future work:

- Implement a method for parallel network construction in multi-threaded simulation
- Further improve the cache utilization of multi-threaded simulations
- Implement a scheduling mode that allows to execute multiple virtual processes in a single thread