Building a Rat Brain in 20 seconds

Massively Parallel Neuronal Network Model Construction

Tammo Ippen[1,2], Jochen M. Eppler[3], Markus Diesmann[1,4,5], Hans Ekkehard Plesser[2,1]

[1] Institute of Neuroscience and Medicine (INM-6) and Institute of Advanced Simulations (IAS-6), Jülich Research Centre and JARA, Jülich, Germany
[2] Department of Mathematical Science and Technology, Norwegian University of Life Science, Ås, Norway
[3] Simulation Laboratory Neuroscience – Bernstein Facility Simulation and Database Technology, Institute of Advanced Simulations, Jülich Research Centre and JARA, Jülich, Germany
[4] Department of Psychiatry, Psychotherapy and Psychosomatics, Medical Faculty, RWTH Aachen University, Germany
[5] Department of Physics, Faculty 1, RWTH Aachen University, Germany

- Building rat-brain-sized networks in **20 seconds** (**~20 x faster** than before)
  ➡ more comprehensive in silico experiments.
- *Scalable* construction of neuronal networks from *single compute node* simulations to *supercomputer* simulations.
  ➡ better usage of available supercomputer resources

Introduction

With the neural simulator NEST [1] biological neuronal networks can be simulated and researched. Being a hybrid OpenMP and MPI parallel application, NEST is already capable of simulating neuronal networks of spiking point neurons of the size of ~1% of the human brain [7]. To further investigate the brain, more complex and larger networks will become necessary. NEST's data structures [2] enable efficient storage of those networks. We present our ongoing work to provide efficient and scalable algorithms to construct the networks.

Future computer architectures increase the number of cores on single compute nodes to keep the energy consumption at a reasonable level, while increasing compute capabilities. Using purely MPI-based parallelization on such systems entails a huge overhead, which makes the use of efficient methods for node-based parallelism essential. However, previous implementations of a parallelized network setup did not scale well when using OpenMP (**Fig. 1**).

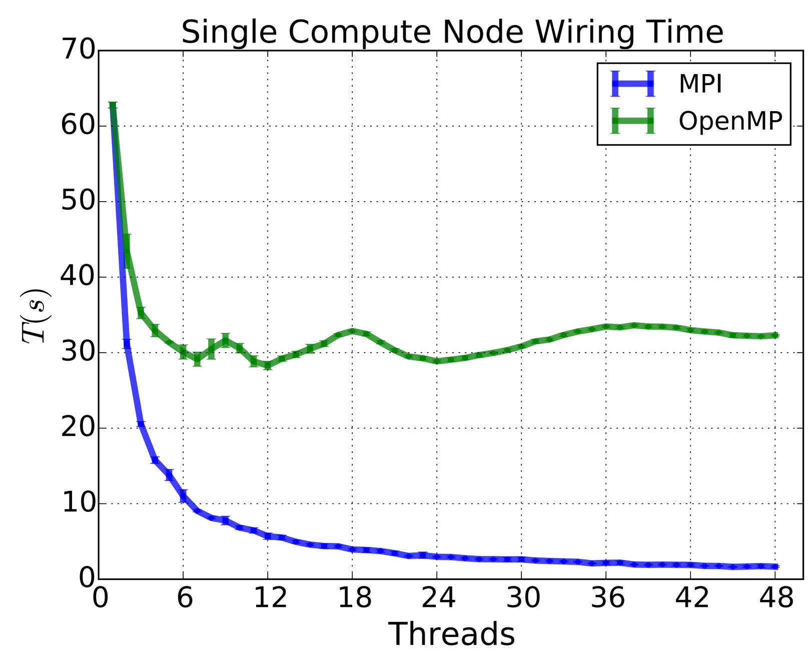


Fig. 1: Constructing a random balanced network [3] with 25.000 point neurons and ~62.5 x 10^9 synapses (mean and std. deviation of 5 samples, NEST 2.6.0, AMD Opteron 6174, 48 cores, 2.2 GHz). The construction with pure MPI shows scaling, while construction with pure OpenMP basically shows no scaling beyond 6 threads

Network Setup in NEST

The networks in NEST consists of nodes (neurons and devices for stimulating and recording from the neurons) and connections that allow the communication between nodes. As the number of connections is about 10^4 times the number of neurons in biological neuronal networks, their creation takes up the largest part of the setup time. To understand the runtime behavior, we need to look at the data structures and algorithms for the creation of connections.

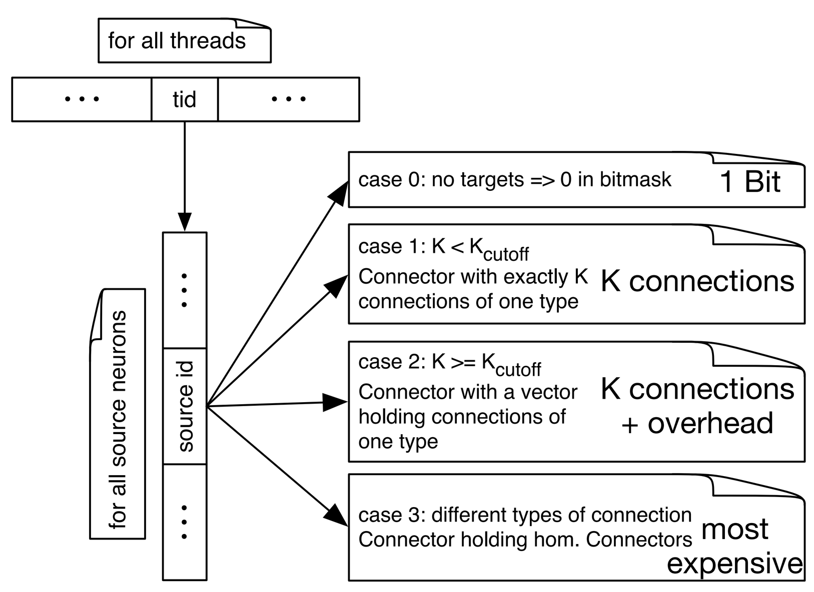


Fig. 2: NEST's connection infrastructure [2] is optimized for large scale simulations, where a source neuron only has a few local target neurons. For smaller scale simulations the data structures evolve into more general containers.

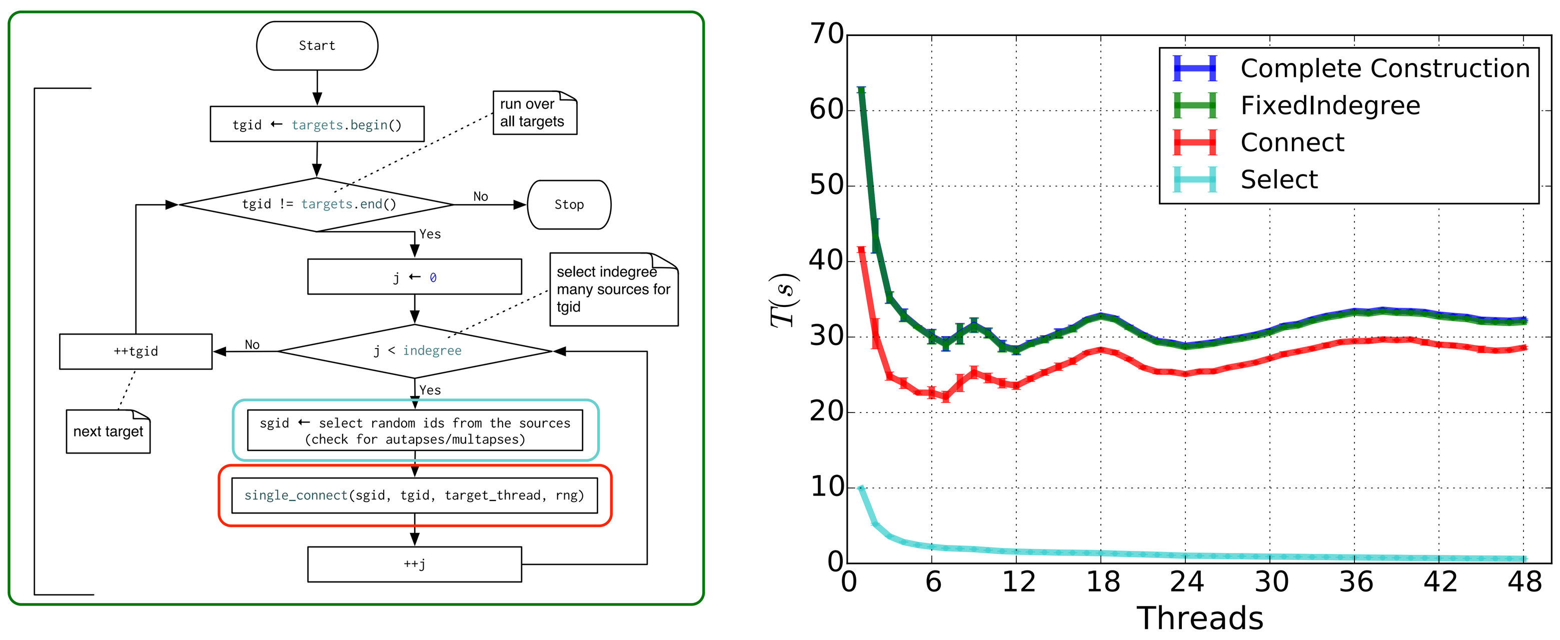
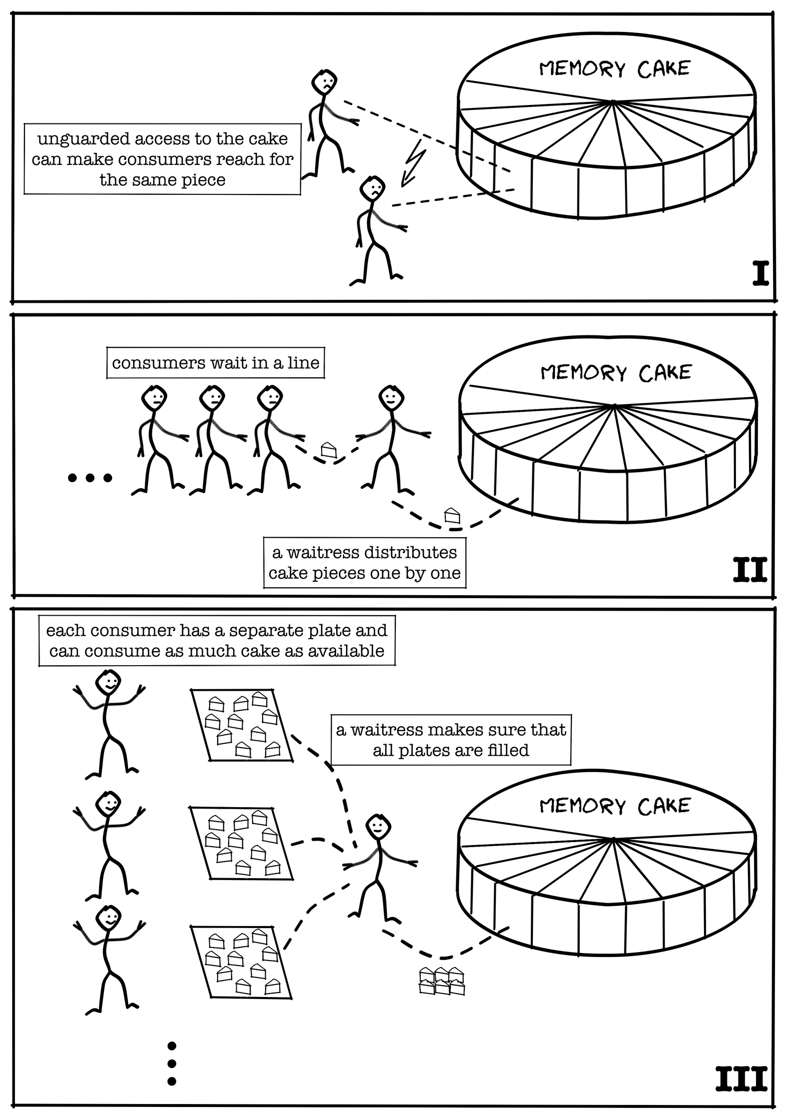


Fig. 3: Inspecting the thread-parallel construction phase. The `FixedIndegree` connecting algorithm (sketched on the left) dominates the construction phase. Within `FixedIndegree`, most time is spend in the `single_connect` part, which allocates memory for the connection objects. The colors in the figure correspond to the colors in the algorithm.

Improved Memory Allocation

In **Fig. 3** we show that most time is spent in thread-parallel allocation of memory for the objects. Optimized thread-aware memory allocators (tcmalloc [8], jemalloc [5], TBB [4], ptmalloc3 [6], ...) have thread-private heaps and cache small objects on each thread. The comic illustrates the different memory allocation strategies. **Fig. 4** shows the performance of different allocators supporting the network construction.



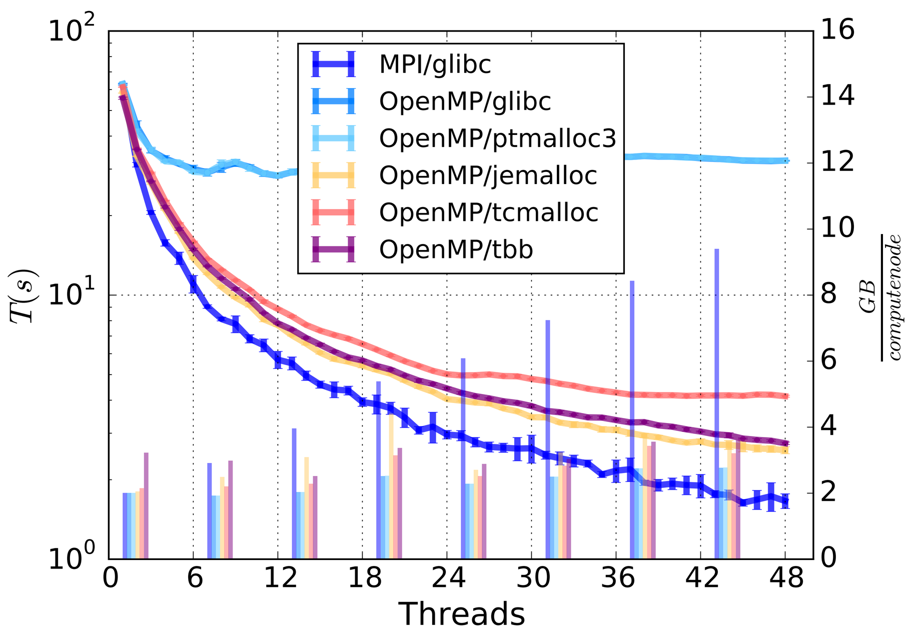


Fig. 4: Comparing wiring time using different memory allocators. Continuous curves show the network construction time (left axis) and the bars show the memory consumption (right axis). Scalable performance for threads is achieved at the expense of larger memory consumption, which is still considerably smaller than the memory consumption of MPI.

Large-Scale Simulations

In simulations large enough to exploit the entire JUQUEEN supercomputer [7] other effects dampen the wiring performance. Neurons are distributed among the compute nodes in a round robin fashion. When executing the `FixedIndegree` algorithm from **Fig. 3**, only now and then there is a local target neuron (every 30k up to every 2000k neuron is local): most of the time, there is no local target and no connections are created. **Fig. 5** to **7** show the impact of an improved iteration scheme and the use of optimized memory allocators when constructing a random balanced neuronal network [3] with ~200 x 10^6 point neurons and ~2.25 x 10^{12} synapses (about the size of a rat brain).

Fig. 5: Wiring performance of 'vanilla' NEST 2.6.0. The color codes are the same as explained in **Fig. 3**. There is no scaling beyond 16 threads. Further two gaps can be observed, which are caused by idling in the connection algorithms:

```
for (size_t i=target_from; i <= target_to; i++)
{
  // This is true for neurons on remote processes
  if ( !is_local_gid(i) )
    continue;

  Node* target = get_node(i);

  // Check, if target is on our thread
  if (target->get_thread() != tid)
    continue;

  // select random sources & do the connecting
}
```

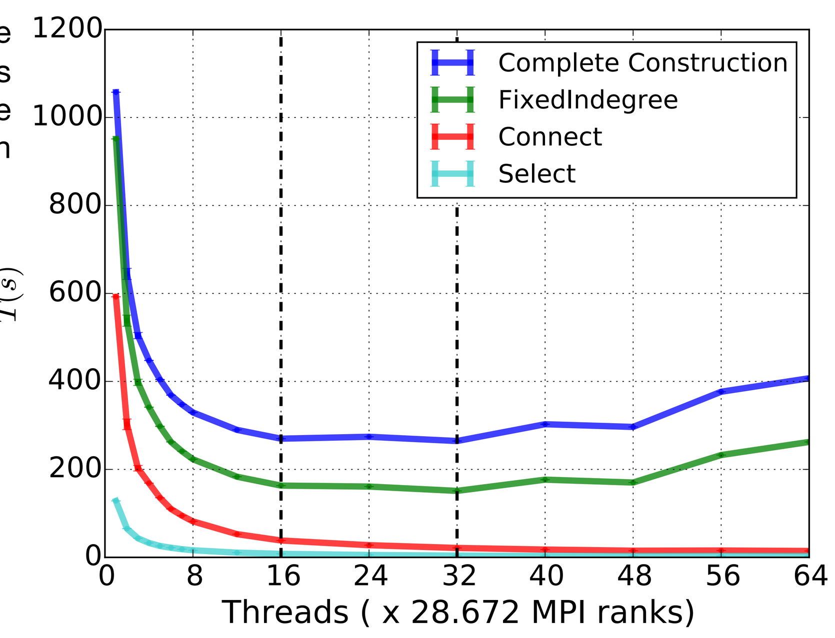


Fig. 6: Wiring performance of modified NEST 2.6.0. The color codes are the same as explained in **Fig. 3**. After iterating local targets only, performance increases considerably. Scaling is again limited by memory allocation:

```
for (size_t k = 0; k < local_nodes_.size(); ++k)
{
  Node* target = local_nodes_.get_node_by_index(k);
  size_t i = target->get_gid();

  if (!(target_from <= i && i <= target_to))
    continue;

  // Check, if target is on our thread
  if (target->get_thread() != tid)
    continue;

  // select random sources & do the connecting
}
```

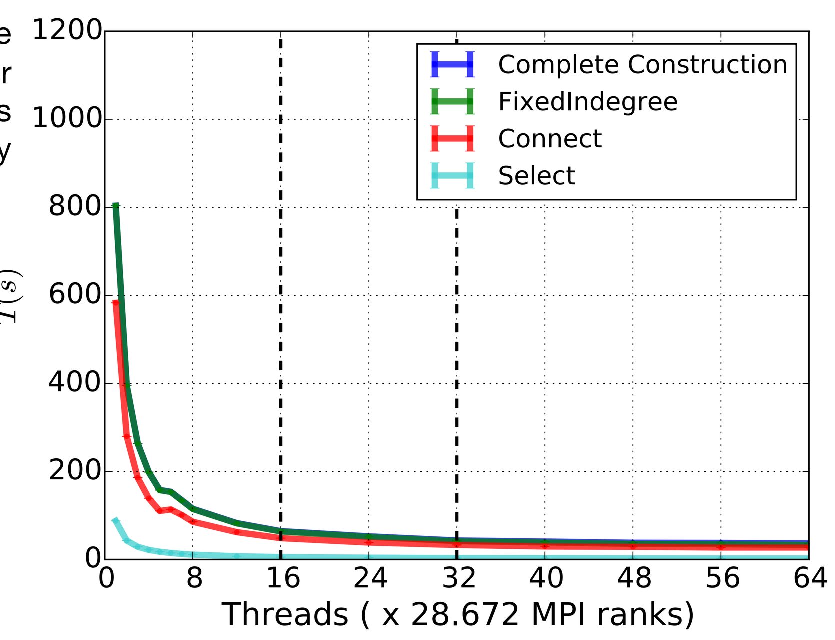
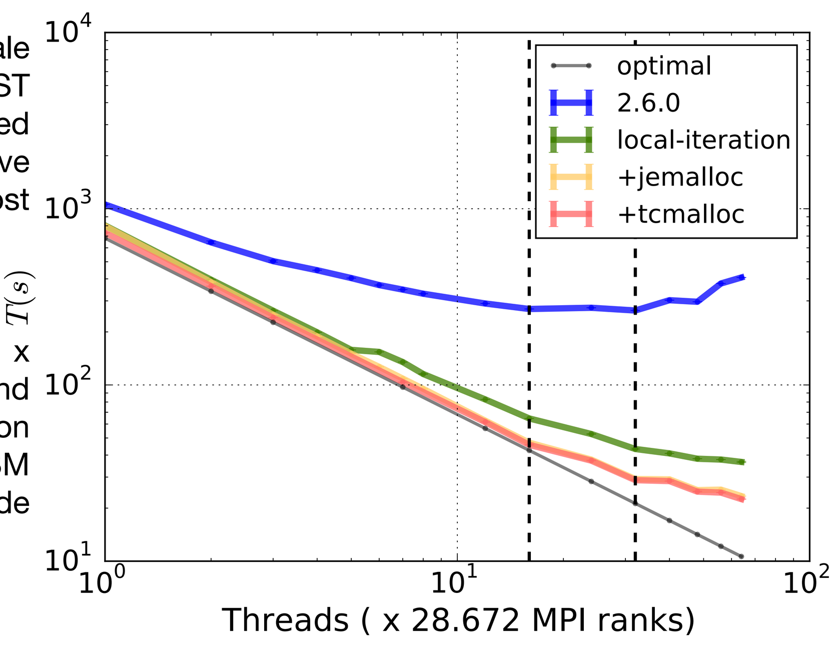


Fig. 7: Comparing wiring performance of large-scale simulations: vanilla NEST (dark blue), modified NEST (green) and modified NEST with specialized allocators (red for tcmalloc, yellow for jemalloc). Using the alternative iteration schema and specialized allocators allow almost perfect linear scaling.

Constructing random balanced networks [3] with ~200 x 10^6 point neurons and ~2.25 x 10^{12} synapses (mean and std. deviation of 3 samples, NEST 2.6.0, performed on JUQUEEN supercomputer in the FZ Jülich (28.672 x IBM PowerPC A2, 1.6 GHz, 16 GB RAM, 16 cores per node with up to 4 hardware threads per core)).



References

[1] Gewaltig & Diesmann (2007), Scholarpedia, doi: 10.4249/scholarpedia.1430
[2] Kunkel et al. (2014), Front. Neuroinform, doi: 10.3389/fninf.2014.00078
[3] Brunel (2000), Comp. Neuroscience, doi: 10.1023/A:1008925309027
[4] Kukanov et al. (2007), Intel Technology Journal, doi: 10.1535/itj.1104.05
[5] Evans (2006), Proceedings of the BSDCan Conference, www.canonware.com/jemalloc/
[6] Glocker (2006), www.malloc.de/en/index.html
[7] Himeno (2013), www.riken.jp/en/pr/press/2013/20130802_1/
[8] Ghemawat (2007), https://gperftools.googlecode.com/git/doc/tcmalloc.html

Acknowledgement: Supported by the Helmholtz Association through the Helmholtz Portfolio Theme "Supercomputing and Modeling for the Human Brain" and the European Union Seventh Framework Programme ([FP7/2007-2013]) under grant agreement no 604102 (Human Brain Project, HBP) and The Jülich Aachen Research Alliance (JARA). Use of the JUQUEEN supercomputer in Jülich was made possible by VSR computation time grant JINB33.