Human Brain Project

JÜLICH FORSCHUNGSZENTRUM

Barcelona Supercomputing Center — Centro Nacional de Supercomputación

CSCS — Centro Svizzero di Calcolo Scientifico — Swiss National Supercomputing Centre

Co-funded by the European Union

Arbor

A morphologically detailed neural network simulator for modern high performance computer architectures

Ben Cumming[a], Stuart Yates[a] , Wouter Klijn[b], Alexander Peyser[b], Vasileios Karakasis[a], Ivan Martinez Perez[c]

[a]Swiss National Supercomputing Center [b] Simulation Lab Neuroscience, IAS, JARA, Forschungszentrum Jülich [c]Barcelona Supercomputing Center

Heterogeneous many-core HPC architectures are the new norm

The HBP PCP pilot systems at Jülich. Both are a radical departure from multicore systems. *left*: IBM "fat node" with Power8 CPUs and 4 GPUs. *right*: Cray XC "blade" with 4 Intel KNL nodes.

These new architectures are already ubiquitous – we need to develop simulators designed to exploit these architectures now. Arbor aims to meet this need, alongside other efforts to add many core support to existing software. Designing software from the ground-up for heterogeneous and many-core systems will pay off in the medium to long term.

What is Arbor?

Arbor is developed by a team from HPC centers:

- CSCS, Jülich and BSC in *WP 7.5.4*
- Aim is to prepare neuroscience users for new HPC architectures

Arbor is designed from the ground up for *many core* architectures:

- Written in C++11 and CUDA
- Using MPI, TBB and C++11 threads
- *Open source* and *open development*
- Sound development practices: *unit testing*, *continuous integration*, and *validation*

Progress & features

Highlights of progress since the last HBP Summit:

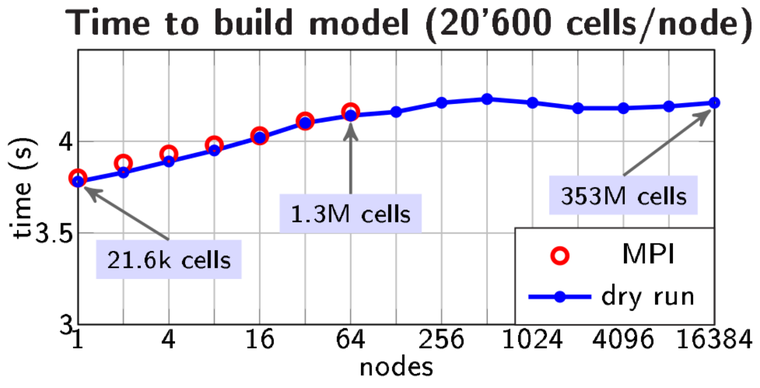
- Optimized back ends for CUDA, KNL and AVX2
- Asynchronous spike exchange that overlaps compute and communication
- Efficient sampling of voltage and current on all back ends
- Efficient implementation of all features on GPU
- Reporting of memory and energy consumption (when available on platform)
- An API for addition of new cell types, e.g. LIF neurons and Poisson spike generators
- Validation tests against numeric/analytic models and NEURON

Parallel model and network construction

In Arbor models are described by *recipes*:

- Provide a functional model description
- Lazy generation of cell attributes to be processed in parallel by a load balancer
- Work and memory requirements per node are proportional to cells-per node
- Low resource requirements mean model building and simulation need not be separated

Time to build model (20'600 cells/node)



Code generation

Performance portability presents two challenges:

1. Efficient cell state integration requires hardware-specific implementations
2. Extensibility demands that new ion channels or synapse models can be provided by simulator users

Solution: Use NMODL to describe mathematics and generate optimized code from that for each platform

Current backends

CUDA (NVIDA), AVX512 (KNL & Sky Lake), AVX2 (Haswell & Broadwell) and generic C++

Key hardware-specific features:

- *CUDA* fast key-value reduction for synapse current collection
- *AVX2* vectorised transcendentals for GCC and Clang
- *AVX512* vectorised gather and scatter for per-compartment loads & stores

Resource consumption

The resources used by a simulation can be measured in different ways:

- *Time to solution (TTS)*: wall time (s)
- *Node hours (NH)*: TTS $\times$ nodes/3600
- *Energy to solution (ETS)*: total energy (kJ)

Resource consumption for 590k cells.

	daint-mc			daint-gpu		
nodes	TTS	NH	ETS	TTS	NH	ETS
32	204.0	1.81	1771	176.0	**1.56**	**1159**
64	102.4	1.82	1769	89.8	1.60	1169
128	51.7	1.84	1743	47.1	1.67	1194
256	26.8	1.91	1742	25.5	1.82	1247
512	**13.5**	1.91	1698	15.1	2.15	1374

Resource measurements for strong scaling described in "Performance" section

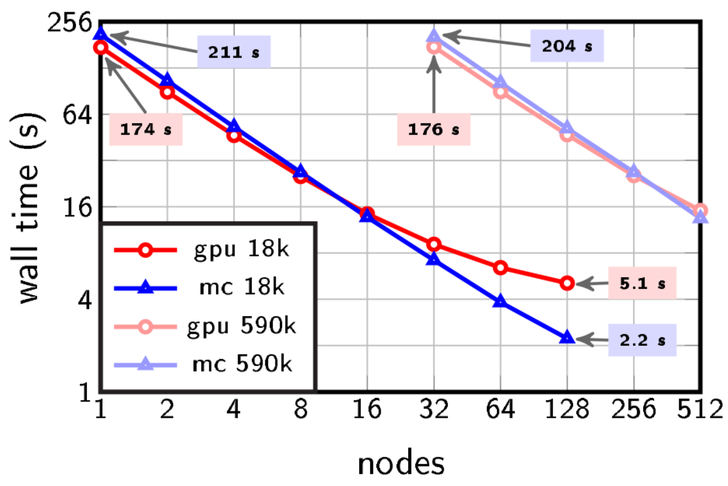
- GPU maximises user allocation (86% NH required to run on CPU)
- The GPU uses 65% energy to solution
- CPU can be used when reducing time to solution is most important

Arbor provides good performance on both systems, meeting our *performance portability* requirement.

Performance

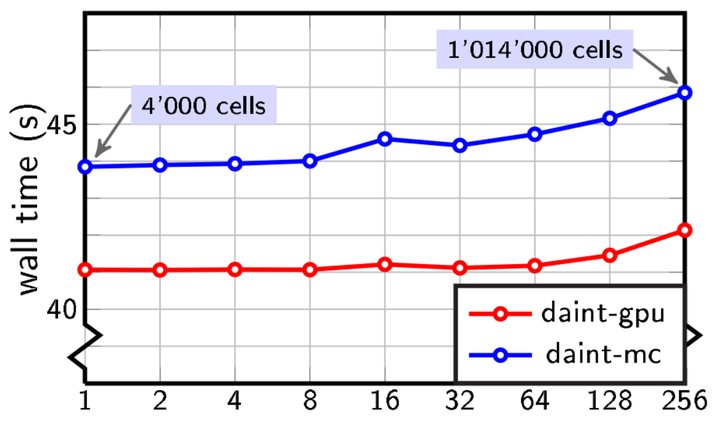
daint-mc	Cray XC40: 2$\times$ 18-core Broadwell per node
daint-gpu	Cray XC50: 1$\times$P100 GPU per node
cells	50 compartments & 5000 synapses per cell Passive dendrites, Hodgkin-Huxley soma
network	random
duration	200 ms

Strong Scaling



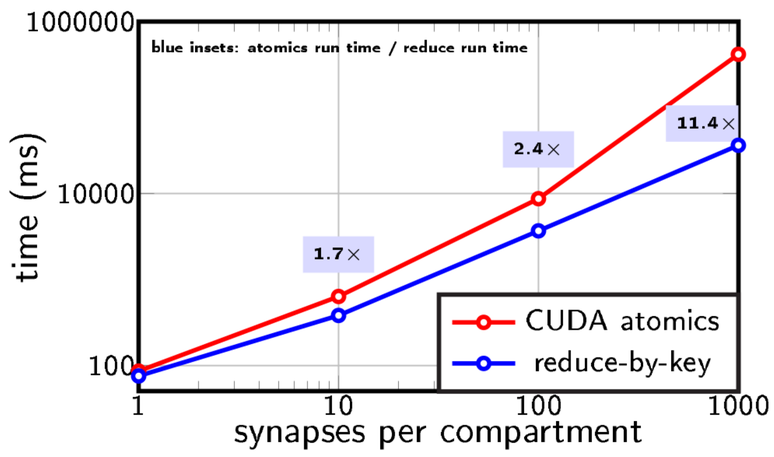
Time to solution on CPU and GPU for models with 18'000 and 590'000 cells. GPU performance is better in "throughput mode", i.e. with more cells per node, while CPU is faster with fewer cells per node.

Weak Scaling



Time to solution with 4000 cells per node, as the number of nodes is increased. Run time increases by less than 25% when the problem is scaled by 256$\times$. Arbor has a *dry-run* mode that was used to tune scaling issues with many nodes.

Optimization: Updating Membrane Currents on GPU



Time taken to update 10'000 compartments as the number of synapses per compartment varies. Race conditions occur when multiple synapses on the same compartment update compartment current simultaneously. Arbor has an optimized reduce-by-key operation that improves performance over naive CUDA atomics when many synapses are attached to the same compartment.

Get in touch!

source	github.com/eth-cscs/arbor
email	bcumming@cscs.ch a.peyser@fz-juelich.de