Human Brain Project

JÜLICH FORSCHUNGSZENTRUM

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Co-funded by the European Union

Arbor

A morphologically detailed neural network simulation library for modern high performance computer architectures

Ben Cumming[a], Stuart Yates[a], Nora Abi Akar[a], Anne Küsters[b], Wouter Klijn[b], Alexander Peyser[b]
[a]Swiss National Supercomputing Center [b]Simulation Lab Neuroscience, Forschungszentrum FZ-Jülich

What is Arbor?

Arbor is a library for the simulation of large networks of morphologically-detailed neurons for all HPC systems in the HBP.

Runs on GPU systems, vectorized multicore, Intel KNL and laptops.
Modular design for extensibility to new computer architectures.

Arbor is developed by a team from HPC centers.

CSCS and FZ-Jülich in SGA2 WP 7.3.

Progress & Features

Features added since the last HBP Summit:

- Vectorization of NMODL kernels with vector intrinsics. See *Vectorization* box below.
- A spike communication and event system that scales with negligible overheads to very large models and clusters.
- A new task-based threading implementation.
- CMake installable target and simple configuration for users of the Arbor library.
- Fine-grained allocation of CPU and GPU resources.
- An API for receiving spikes from external simulators.

Features coming soon to Arbor include:

- A Python wrapper. See the *Python Interface* box for details.
- Gap junctions.
- Coupling with NEST & TVB.
- Coupling with in-situ visualization and analytics.
- A benchmark and validation suite.
- Higher-order time stepping and error control.

Talk to us about features that you need.

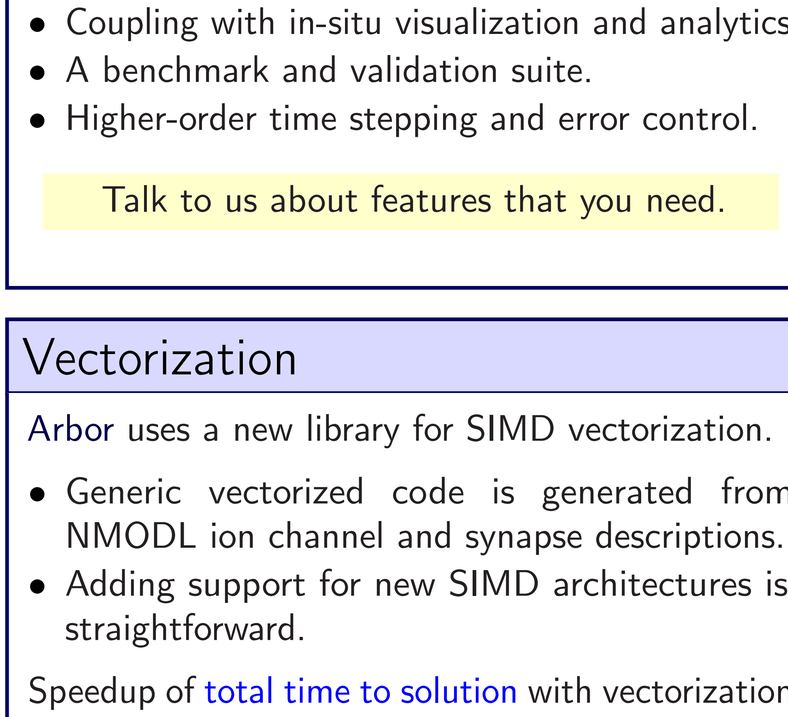
Vectorization

Arbor uses a new library for SIMD vectorization.

- Generic vectorized code is generated from NMODL ion channel and synapse descriptions.
- Adding support for new SIMD architectures is straightforward.

Speedup of total time to solution with vectorization is 0.5–$2.5\times$. The plot below shows speedup on a range of Intel CPUs, both for single core and for a full socket.

Vectorization Speedup of Wall Time



Python Interface

A Python wrapper for Arbor will be released this year.

- Simple wrappper of the C++ API.
- Basis for PyNN integration.
- Working prototype with MPI & GPU on Cray.

Ask us for a demonstration of the Python wrapper running in a Jupyter notebook on Piz Daint.

Below is a simple example of defining, building and running a network with 4 soma-only cells:

```
#!/usr/bin/python3
import pyarb as arb
class ring_recipe(arb.recipe):
  def __init__(self, n):
    super().__init__()
    self.ncells = n
  def num_cells(self):
    return self.ncells
  # each cell is a soma-only cell
  def cell_description(self, gid):
    cell = arb.make_soma_cell()
    loc = arb.segment_location(0, 0.5)
    cell.add_synapse(loc)
    cell.add_detector(loc, 20)
    if gid==0: # add a stimulus to first cell
      cell.add_stimulus(loc, 0, 20, 0.01)
    return cell
  # 1 synapse target on each cell
  def num_targets(self, gid):
    return 1
  # 1 spike detector on each cell
  def num_sources(self, gid):
    return 1
  # all cells are multi-compartment
  def kind(self, gid):
    return arb.cell_kind.cable1d
  # each cell has one
  # incoming connection from
  # the cell with gid-1
  def connections_on(self, gid):
    src_id = (gid-1) % self.ncells
    src = arb.cell_member(src, 0)
    tgt = arb.cell_member(gid, 0)
    return [arb.connection(src, tgt, 0.1, 10)]
# get parallel arbor context
# by default takes all
# available cores and GPUs
ctx = arb.context()
# make a 4 cell ring
recipe = ring_recipe(4)
# make the simulation
sim = arb.simulation(recipe, ctx)
# get a spike recorder
recorder = arb.make_spike_recorder(sim)
# run simulation for 100 ms
sim.run(100, 0.025)
# print the spikes
for spike in recorder.spikes:
  print('cell {} at {:8.3f} ms' \
        .format(spike.source.gid, spike.time))
```

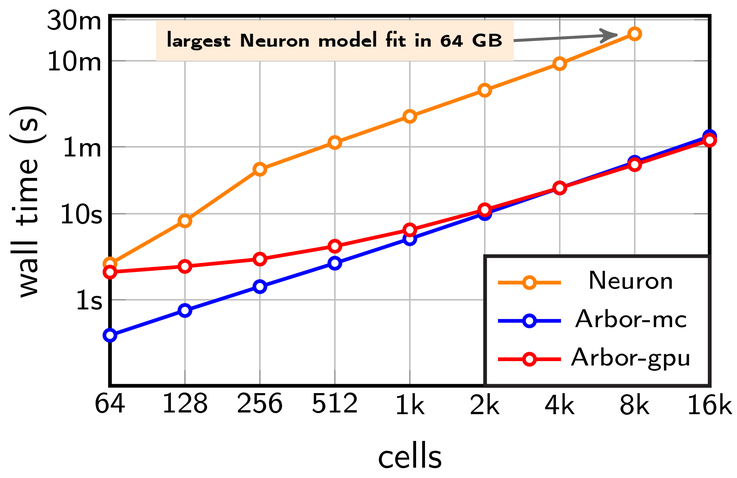
```
cell 0 at     5.375 ms
cell 1 at    15.700 ms
cell 2 at    26.025 ms
cell 3 at    36.350 ms
cell 0 at    46.675 ms
cell 1 at    57.000 ms
cell 2 at    67.325 ms
cell 3 at    77.650 ms
cell 0 at    87.975 ms
cell 1 at    98.300 ms
```

Performance

daint-mc	Cray XC40: 2× 18-core Broadwell per node
daint-gpu	Cray XC50: 1×P100 GPU per node
cells	150 compartments & 10,000 synapses per cell Passive dendrites, Hodgkin-Huxley soma
network	ring network
duration	100 ms

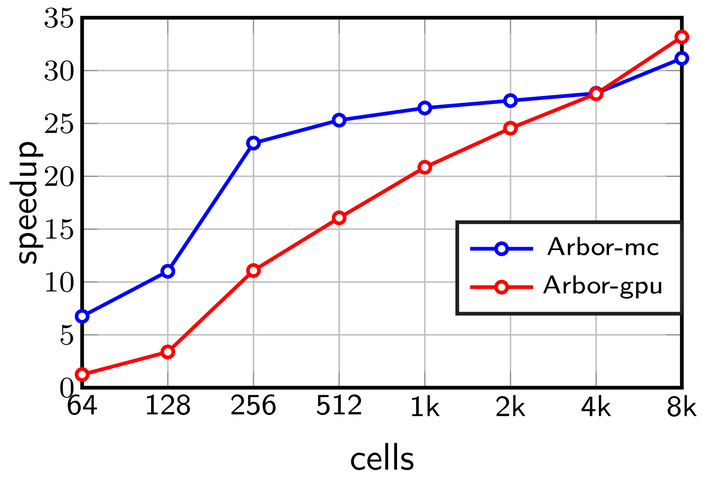
Single Node Scaling

This benchmark scales a simple ring model on a single node of Piz Daint.



- Arbor's efficient multicore memory layout gives perfect scaling.
- NEURON scales poorly from 64–256 cells as cache utilization decreases.

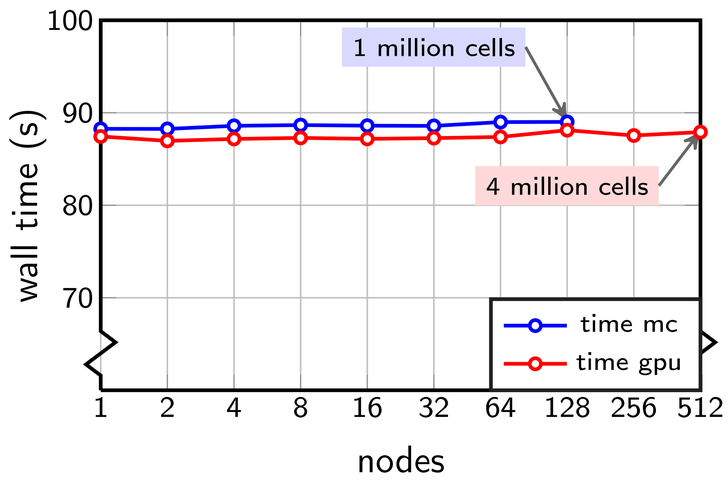
This plot shows the speedup of Arbor relative to NEURON for the single node test above.



- For few cells per core NEURON is 5–10× slower.
- With more than 7 cells per core Arbor is over $20\times$ faster.
- Arbor's GPU backend is efficient with 1000 or more cells per GPU.

Arbor Scales On Large Clusters

Here a model similar the one above with a network of 10,000 random connections per cell is weak-scaled from one to hundreds of nodes with 8000 cells per node:



- Multicore and GPU weak scale perfectly.
- The GPU requires 25% less energy.

Get in touch!

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