arbor

INSIGHTS

A morphologically-detailed neuronal network simulation library for contemporary high performance computing architectures

12TH DECEMBER 2018 I ALEXANDER PEYSER & ANNE KÜSTERS

Human Brain Project

JÜLICH Forschungszentrum

FET FLAGSHIPS

Co-funded by the European Union

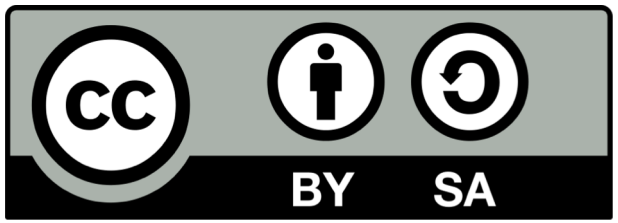
AGENDA

Insights into Arbor

- Introduction
- Features
- Model
- Performance

Hands-on session

- Build and run a ring network with python



RECENT COLLABORATORS

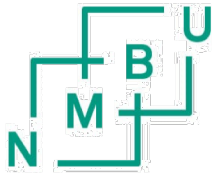
From different institutions

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

- Ben Cumming
- Stuart Yates
- Nora Abi Akar
- Vasileios Karakasis

- Alexander Peyser
- Wouter Klijn
- Anne Küsters
- Felix Huber

- Simon Oerl



Norwegian University
of Life Sciences

- Susanne Kunkel

openly available @ https://github.com/arbor-sim/arbor

WHAT IS ARBOR?

A morphologically-detailed neuronal network simulation library for contemporary HPC architectures

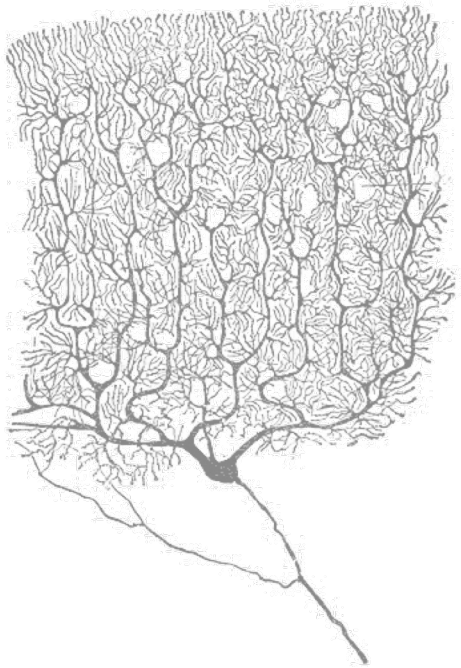
A **library** for the simulation

- of **large networks** of morphologically-**detailed, spiking neurons**
- for all **HPC** systems in the HBP

Runs on multiple architectures

- **GPU** systems,
- **vectorized** multicore,
- Intel **AVX** and **laptops**

Modular design for **extensibility** to new computer architectures



Purkinje cell by Santiago Ramón y Cajal

WHY ARBOR?

To solve multi-compartment simulations with large networks on new HPC architectures

Problems and models that are challenging to explore with current software and systems, e.g.

- Near real-time multi-compartment simulations
- Large networks with long simulations, parameter search, statistical validation
- Field potential calculations of large networks with volume visualization

Adapting existing simulators to **new HPC architectures** is hard, e.g. for

- Highly parallel architectures such as Intel Xeon and Intel KNL
- Wider vector operations such as AVX, AVX2, AVX512
- Specialized accelerator hardware as GPUs

Source of picture: flaticon.com

FEATURES OF ARBOR

Aiming for interoperability by being a simulator as library

* available soon

Interoperability
Simulator as library

- **Visualization** (with coupling to in-situ visualization and analysis tools*)
- **Multi-physics:** can be integrated with other tools
- **Multi-scale** from single neurons to large multi-compartmental networks
- **Usability:** installable target and simple configuration, python front-end (as basis for PyNN integration*), efficient sampling of voltage and currents

Extensibility
Modular internal API

Performance
HPC targeted

Source of picture: flaticon.com

FEATURES OF ARBOR

Aiming for Extensibility by having modular internal API

* available soon

Interoperability
Simulator as library

Extensibility
Modular internal API

- New **integration schemes,** (high-order time stepping, error control, and efficient gap junction schemes*)
- Custom **spike communication** and event systems,
  API for receiving spikes live from external simulators (e.g. NEST*)
- **Specialized cells:** leaky integrate-and-fire, Hodgkin-Huxley, Poisson spikes

Performance
HPC targeted

Source of picture: CitiXsys

FEATURES OF ARBOR

Aiming for high performance on HPC targets

* available soon

Interoperability
Simulator as library

Extensibility
Modular internal API

Performance
HPC targeted

- **Highly parallel and performance portable** with task-based threading implementation, GPU and SIMD vector targets using NMODL and `modcc`
- Design for **scalability** with fine-grained allocation of CPU and GPU resources
- **Reporting** on memory and energy consumption
- Unit **testing**, continuous integration*, **validation** and a benchmarking suite*

Source of picture: flaticon.com

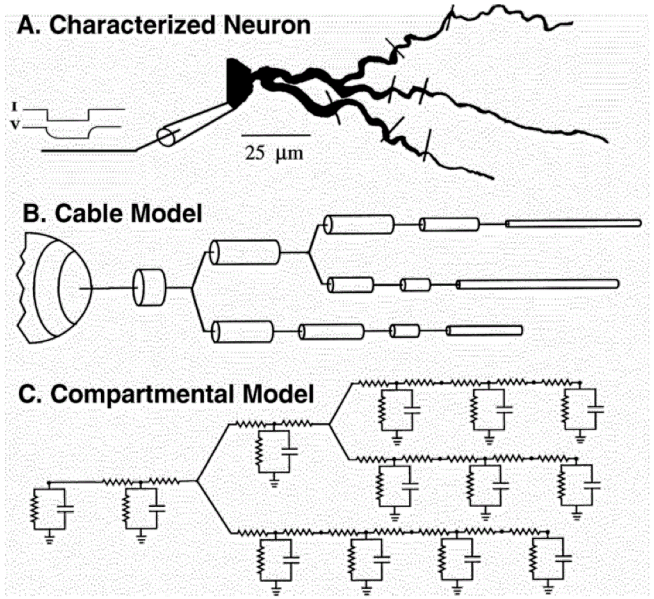
INTRODUCTION

Summary

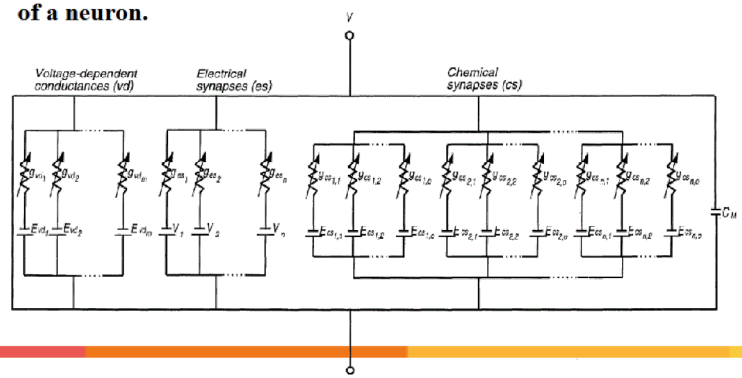
- Arbor is a new library for simulation of morphologically detailed spiking network
  - Specialized for GPUs, vectorized multicore, AVX and laptops
  - Designed to handle very large, very long and computationally intensive problems
- Goals:
  - Interoperability with visualizations and simulators at other scales/problems
  - Modular internal API for extensibility for custom integration, spike communication and cell types
  - And targeted to highly parallel architectures, both existing and emerging
    - with an open development model, validation and testing

NEURON MODEL

Arbor simulates networks of multi-compartment neurons



D. Electrical circuit equivalent of the membrane of a segment of a neuron.



- **Neurons:** approximated by axonal delay, synaptic functions and a set of cables (for dendrites + soma) connected in a tree.
- **Cables:** characterized as 1D electrical compartments (of variable diameter) composed of ion channels, cable resistance and capacitance.
- Neurons represented as sparse, close-to-band matrices to be solved (e.g. by Hines solver) against known current states due to synaptic conductance.
- **Network** and spike exchange between neurons at synapses are represented by concatenations of matrices.

Source: Koch, *Methods in Neuronal Modeling: From Ions to Networks*

CABLE EQUATION

A cell is modelled as a branching, one-dimensional electrical system

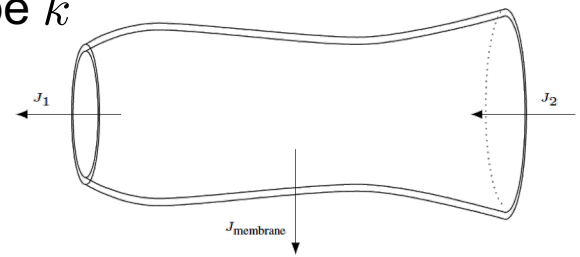
$$\frac{\partial}{\partial x}\left(\sigma\frac{\partial v}{\partial x}\right) = \left(c_m \cdot \frac{\partial v}{\partial t} + \sum_{\text{channels } k} g_k(\underline{s}_k(x,t))(v - e_k^{\text{rev}})\right) \cdot \frac{\partial S}{\partial x}$$
$$+ \sum_{\text{synapses k}} I_i^{\text{syn}}(\underline{s}_k^{\text{syn}}(t), v(x_k^{\text{syn}}))\,\delta x_k^{\text{syn}}$$
$$+ \sum_{\text{injections k}} I_k^{\text{inj}}(t)\,\delta x_k^{\text{inj}},$$

, where

$$\frac{d}{dt}\underline{s}_k(x,t) = f_k(\underline{s}_k, v(x,t)),$$
$$\frac{d}{dt}\underline{s}_k^{\text{syn}}(t) = f_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, v(x_k^{\text{syn}}, t), t),$$

with

- Axial conductivity σ of the intracellular medium
- Membrane areal capacitance c_m, areal conductance g_k for an ion channel of type k as a function of channel's state $\underline{s}_k$
- Corresponding reversal potential e_k^{ref}
- Membrane surface area $S(x)$ as a function of axial distance x



- Current I_k^{syn} produced by a synapse at position x_k^{syn} as a function of the synaptic state $\underline{s}_k^{\text{syn}}$ and local voltage
- Injected current $I_k^{\text{inj}}(t)$ at position x_k^{inj}

NUMERICAL MODEL

Cell state evolution is numerically solved with first order methods

- **Space discretization:** Vertex-centered 1D finite volume method using first-order approximation for axial current flux

$$c_i \frac{dV_i}{dt} = \sum_{j:\ X_j \cap X_i \neq \emptyset} \sigma_{i,j}(V_j - V_i) - \sum_{k:\ x_k^{\text{inj}} \in X_i} I_k^{\text{inj}}(t) - \sum_{k:\ x_k^{\text{syn}} \in X_i} I_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, V_i) - \sum_{\text{channels } k} S_i \cdot g_k(\underline{s}_{k,i})(V_i - e_k^{\text{rev}}),$$

with

$$\frac{d\underline{s}_{k,i}}{dt} = f_k(\underline{s}_{k,i}, V_i),$$
$$\frac{d\underline{s}_k^{\text{syn}}}{dt} = f_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, V_i, t),$$

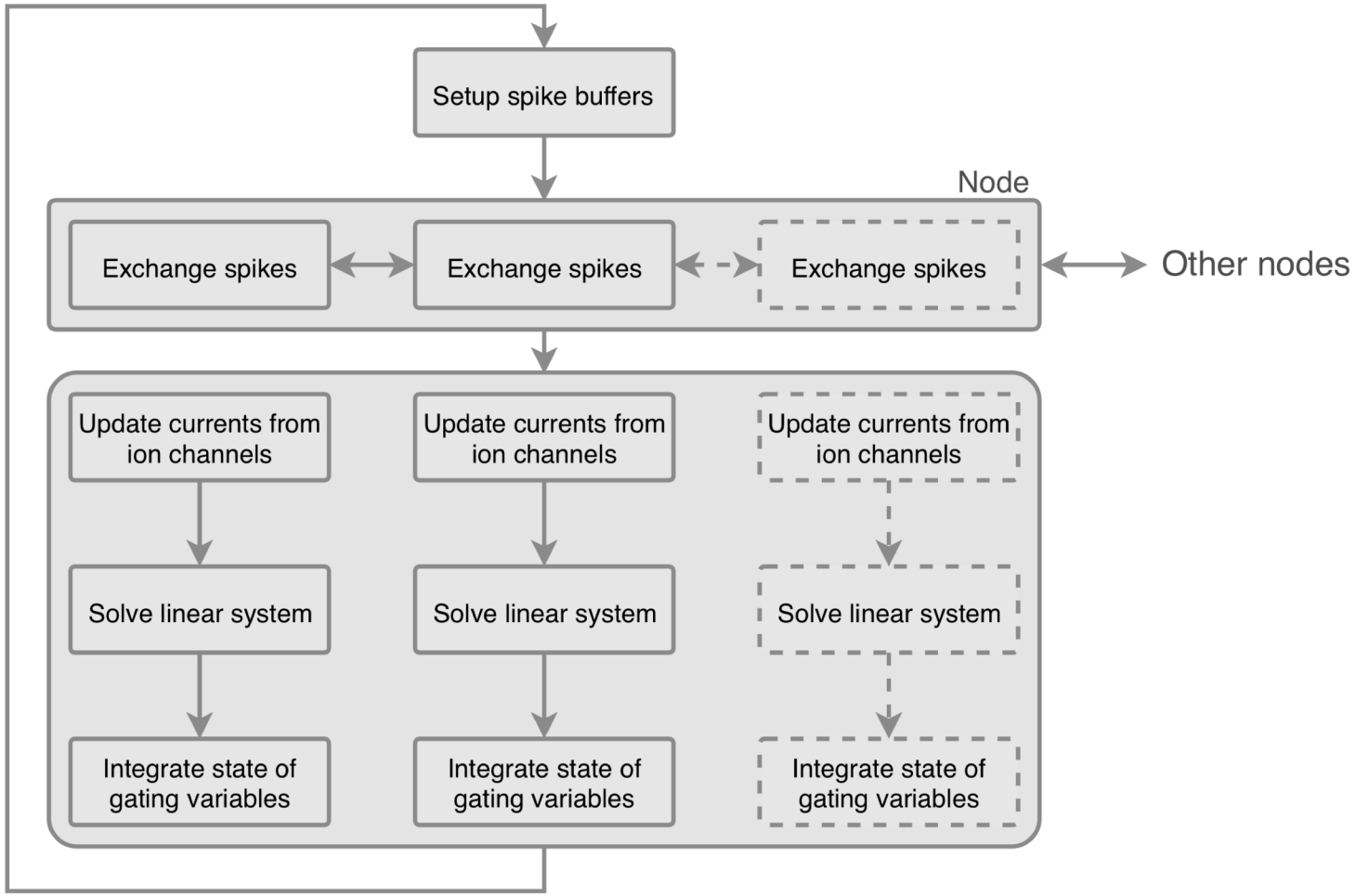
- Voltage and channel state **time evolution split:** Lie-Trotter
  - **Time discretization:** First-order implicit Euler integration

$$\frac{c_i}{\delta t} V_i' + \sum_j \sigma_{i,j} V_i' - \sum_j \sigma_{i,j} V_j' = -I_i^{\text{memb}} + \frac{c_i}{\delta t} V_i$$

  - **Channel state ODEs:** Integration with updated voltages depending on set of ODEs

CELL SIMULATION

Most time consuming parts on a CPU are updating currents and integrating gating variables



DESIGN MODEL

Scalability through the abstraction of recipes

Cells

- A “cell” **represents the smallest model** to be simulated
- A “cell” forms the **smallest unit of work** distributed across processes
- Types:
  - Specialized **leaky integrate-and-fire** cells
  - **Artificial spike** sources
  - **Multi-compartment** cells

Recipes

- A “recipe” **describes models** in a cell-oriented manner and supplies methods to
  - Map **global cell identifier** `gid` to cell type
  - Describe cells
  - List all **connections** from other cells that terminate on a cell
- Advantage: parallel **instantiation** of cell data

DESIGN MODEL

Extensibility through cell group abstraction

Cell groups

- A “cell group” represents a **collection of cells of the same type** together with implementation of their simulation
- **Partitioning** into cell groups provided by decomposition
- A “simulation” manages **instantiation** of model and **scheduling** of spike exchange as well as **integration** for each cell group

Mechanisms

- In a recipe, mechanisms are specifications of **ion channel** and **synapse dynamics**
- Implementations of mechanisms:
  - Hand-coded for CPU/ GPU execution

    or
  - A translator (`modcc`) is used to compile a subset of NEURONs mechanism specification language NMODL to architecture-optimized vectorized `C++` or CUDA source

MODEL

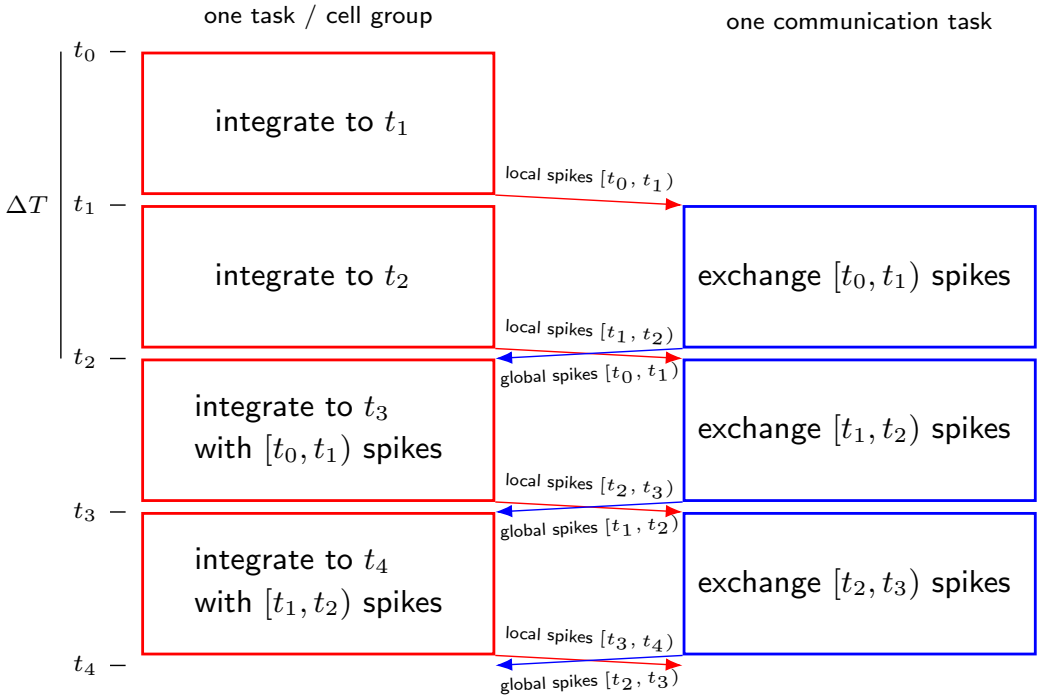
Summary

- Arbor models:
  - Multicompartment neurons using a cable model transformed into a sparse matrix
  - Neurons characterized by axonal delays, synaptic functions and cables connected in a tree
  - Spike exchanges are global across computer nodes, functionally concatenating matrices
- Numerical solutions are discretized in time and space, and channel states are discretized ODEs
- Accelerator (GPU) optimization is focused on updating currents and integrating gating variables
- Models are composed of:
  - Cells representing the small unit of computation (LIF, Artificial sources, Multicompartment cells)
  - Recipes representing a parallelizable set of neuron construction and connections
  - Cell groups computed together on the GPU or CPU
  - Mechanism representing ion channel and synapse dynamics

SPIKE EXCHANGE

With a minimum delay

Overlapping computation and communication with a minimum spike propagation delay ΔT

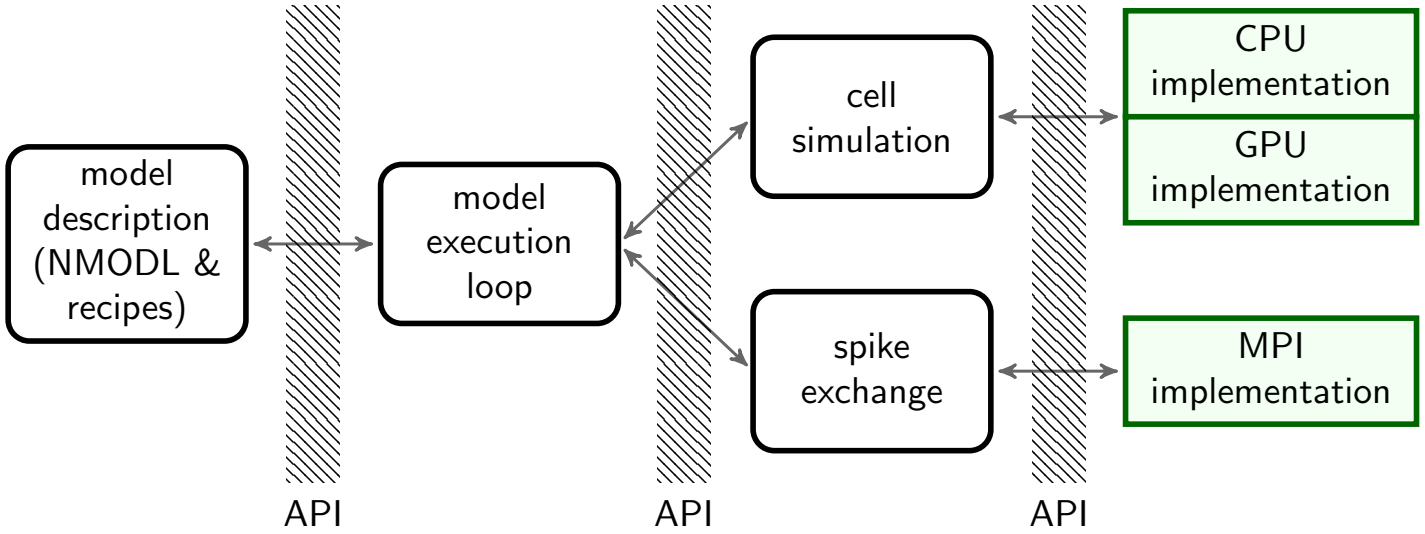


Integration of states in epoch i requires spikes from epoch $i-2$ and are exchanged in epoch $i-1$.

Reason: latency hiding

DESIGN MODEL

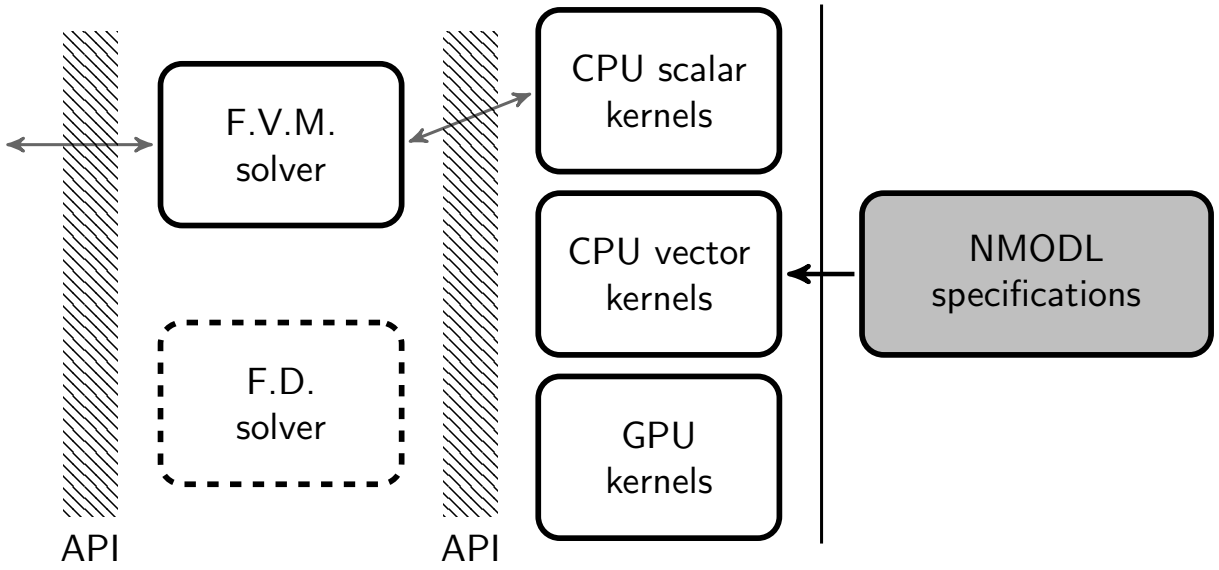
Programming interface ensures extensibility



- Components can be substituted according to the **internal API**.
- Models are described in **NMODL**, a DSL used for the NEURON simulator.
- **Python interface** for building networks is under development.

DESIGN MODEL

Computational work is hidden in backends



- Cell simulation modules share **computational backends** for channel and synapse state evolution.
- CPU-hosted **finite volume** cell simulation.

STRUCTURE

Summary

- Spikes are exchanged at ½ the minimal spike propagation delay to overlap computation and communication
- Internal API uncouples model description, execution, spike exchange and cell simulation
- Computational work is hidden in pluggable backends, allowing automatic generation for different architectures
- Python interface is under development

VECTORIZATION PERFORMANCE

Used systems and benchmark model

Systems

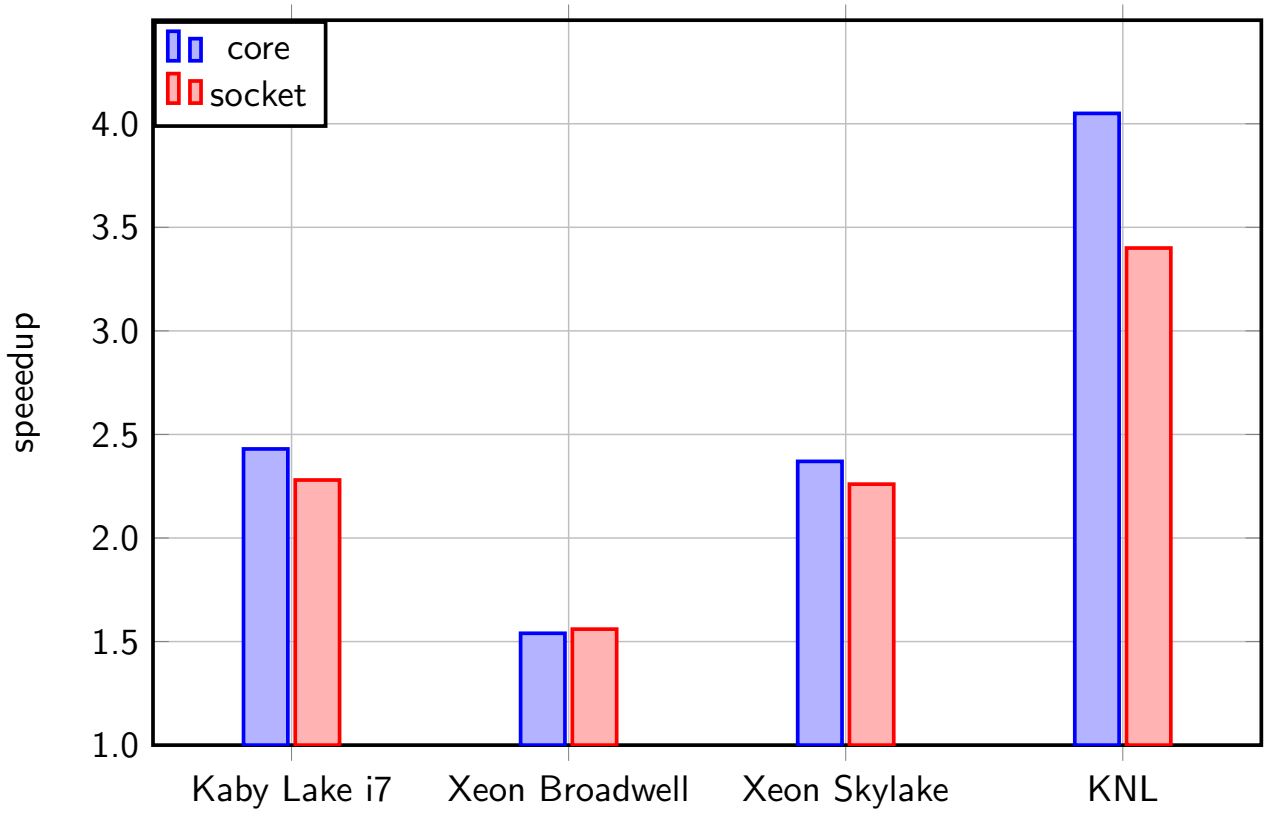
CPU	cores	threads	ISA
Kaby Lake i7	2	4	AVX-2
Broadwell	18	36	AVX-2
Skylake-X	18	36	AVX-512
KNL	64	256	AVX-512

Benchmark model

- Cells: 300 compartments with Hudgkin-Huxley mechanisms, 5.000 randomly connected exponential synapses
- Network: 100 cells per single core
  1000 cells per socket
- Duration: 100 ms

VECTORIZATION PERFORMANCE

Comparison of explicit vectorization relative to the compiler's auto-vectorization



Speedup of total time to solution with vectorization

- **1.5 x** for Broadwell socket
- **3.4 x** for KNL socket

Use of **data-pattern optimized loads and stores** contributes to speedup.

Less improvement for Broadwell due to **poor performance of vectorized division**.

PERFORMANCE BENCHMARKS

Setup of ring network on HPC architecture

System

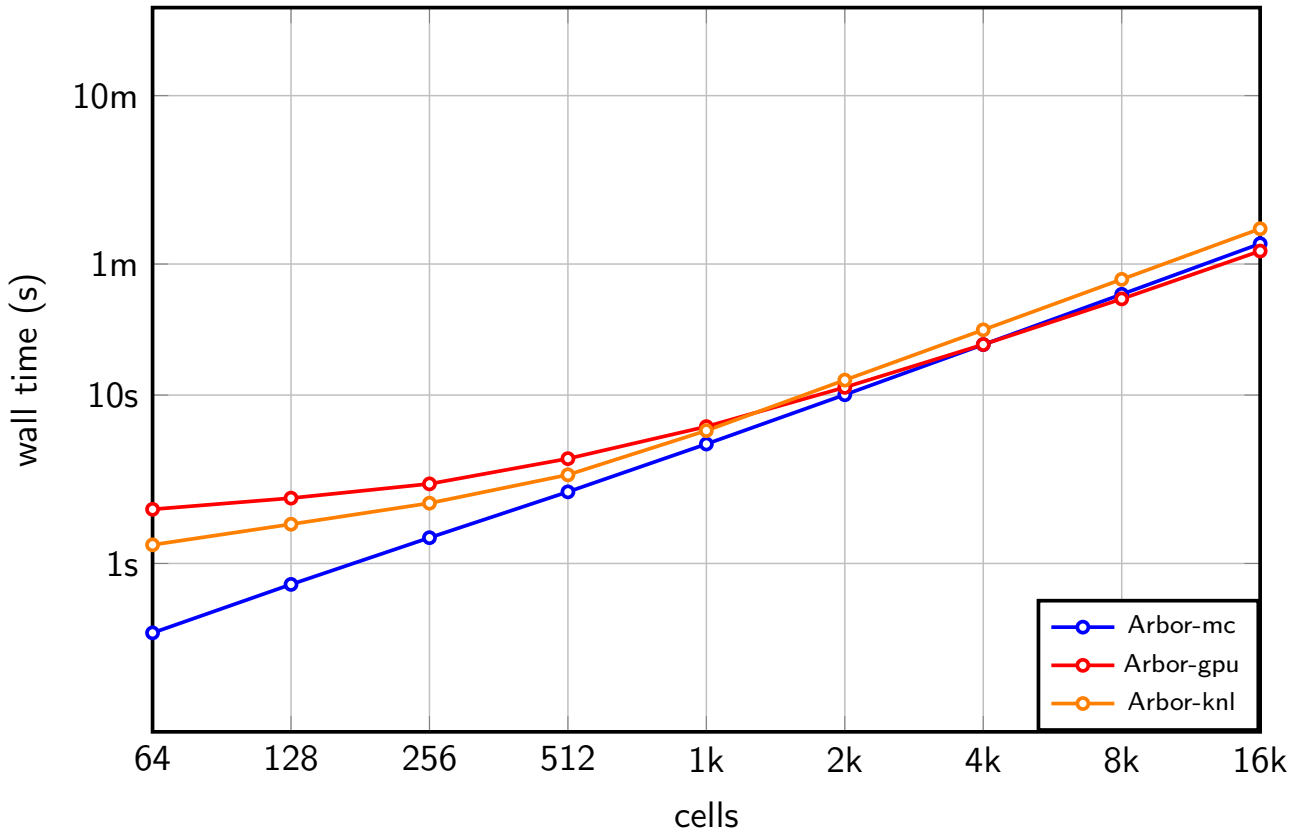
	Daint-mc	Daint-gpu	Tave-knl
CPU	Broadwell	Haswell	KNL
memory	64 GB	32 GB	96 GB
cores/socket	18	12	64
threads/core	2	2	4
vectorization	AVX2	AVX2	AVX512
accelerator	–	P100 GPU	–
MPI ranks	2	1	4
threads/rank	36	24	64
configuration	–	CUDA 9.2	cache,quadrant
compiler	GCC 7.2.0	GCC 6.2.0	GCC 7.2.0

Ring model

- Cells: Randomly generated morphologies with on average 130 compartments
- Synapses: 10 000 exponential synapses per cell with only one synapse connected to a spike detector on the preceding cell
- Soma: Hodgkin-Huxley mechanism;
- Dendrites: Passive conductance

PERFORMANCE

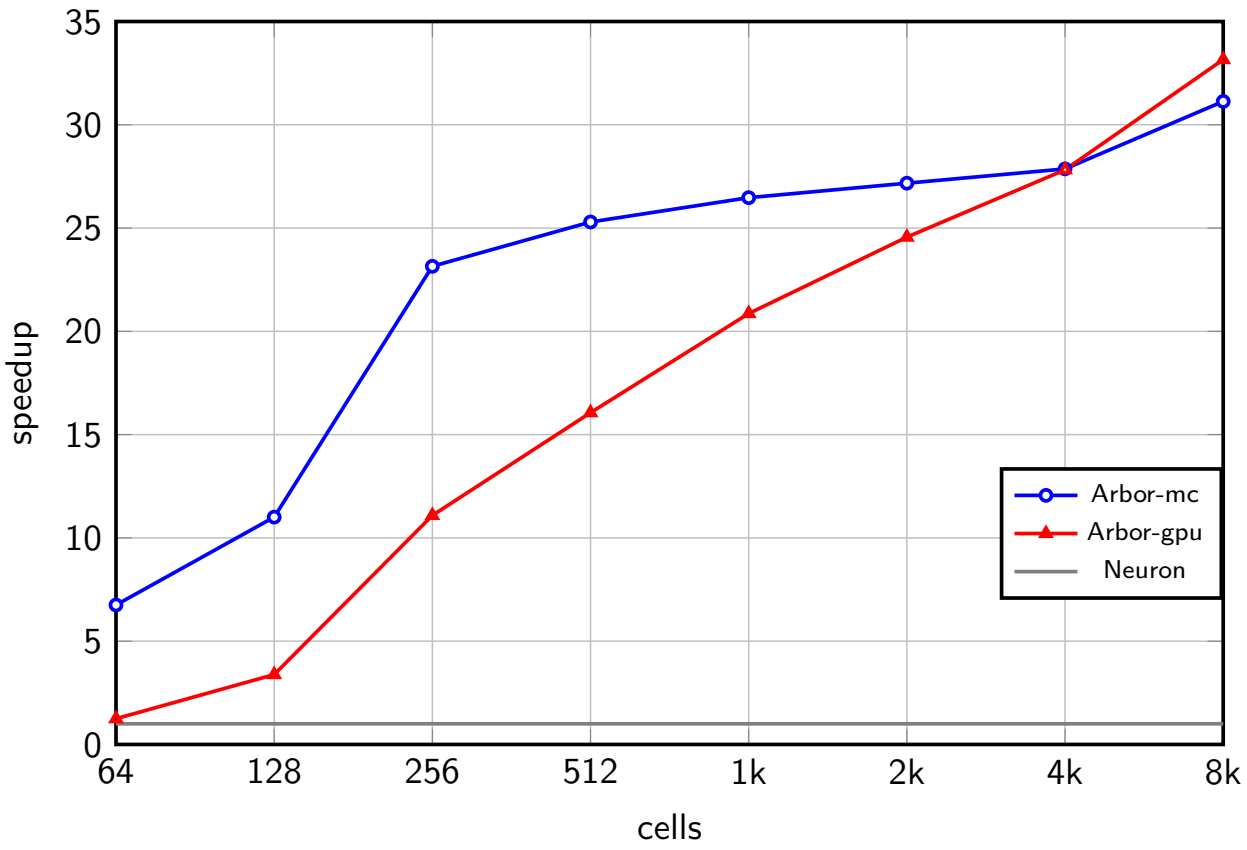
Single node scaling - time: utilization of computational resources on one node at various model sizes



- Models with **fewer cells take less time** to execute
- Scaling is **architecture and model size dependent**
  - MC scales well for 64 or more cells
  - KNL scales well for 512 or more cells
  - GPU scales well for 1024 or more cells
- Below scaling thresholds node resources are **under-utilized**
- **GPU catches up** at 4000 cells

PERFORMANCE

Single node scaling – speedup: comparison with NEURON



Memory

- Arbor significantly more **memory efficient** with 4.4 GB for 16k model,
- NEURON unable to run 16k model due to running out of 64 GB memory available on Daint-mc

Speedup

- Arbor is **faster for all model sizes** with speedup increasing with model size
  - 5-10x faster for less than 128 cells
  - over 20x faster for more than 256 cells

PERFORMANCE BENCHMARKS

Setup of connectivity model on HPC architecture

System

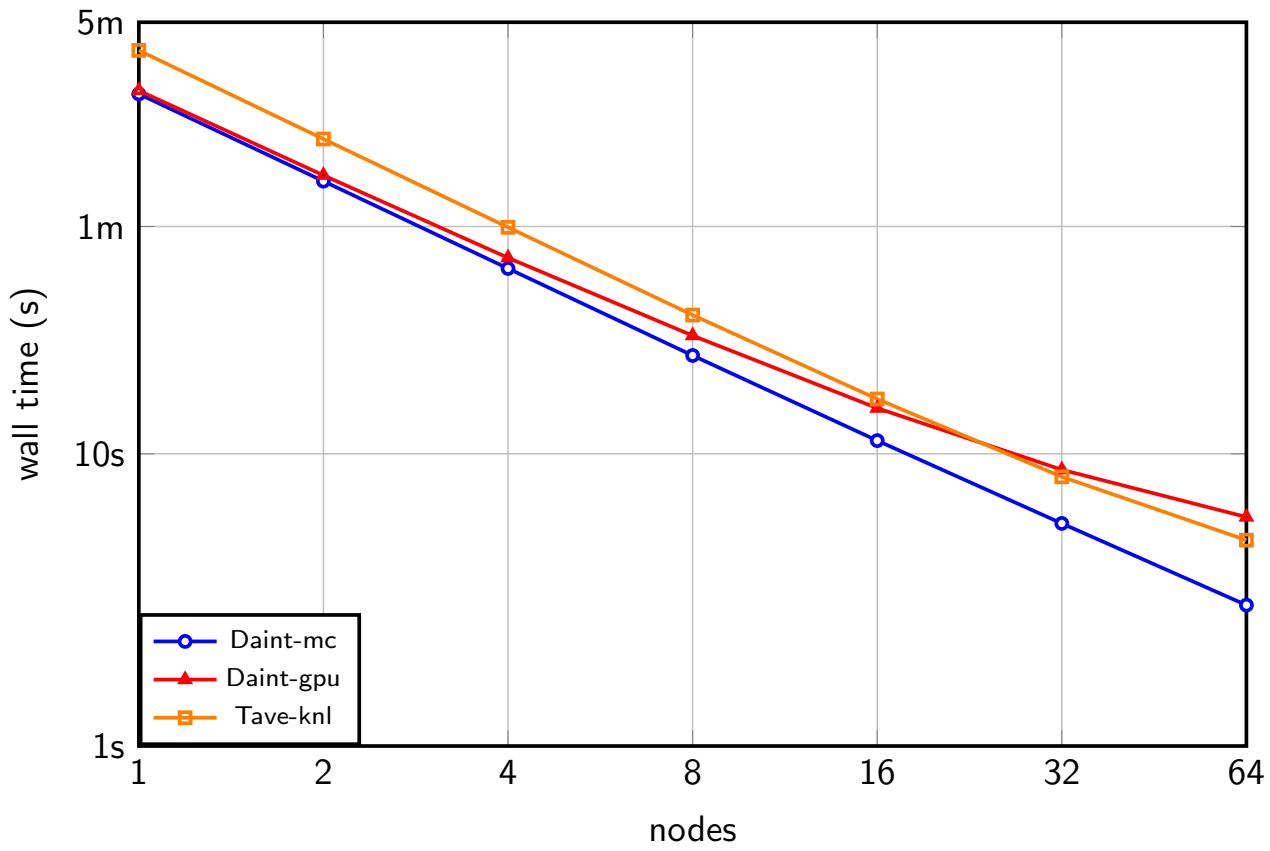
	Daint-mc	Daint-gpu	Tave-knl
CPU	Broadwell	Haswell	KNL
memory	64 GB	32 GB	96 GB
cores/socket	18	12	64
threads/core	2	2	4
vectorization	AVX2	AVX2	AVX512
accelerator	–	P100 GPU	–
MPI ranks	2	1	4
threads/rank	36	24	64
configuration	–	CUDA 9.2	cache,quadrant
compiler	GCC 7.2.0	GCC 6.2.0	GCC 7.2.0

10k connectivity model

- Cells: As in ring model with 16k cells for duration of 100 ms
- Network: 10 000 way randomly connected with no self-connections
- Minimal delay: 10 ms or 20 ms
- Synapses: All excitatory
- Spiking: All cells spike synchronously with frequency 100 Hz or 50 Hz

PERFORMANCE

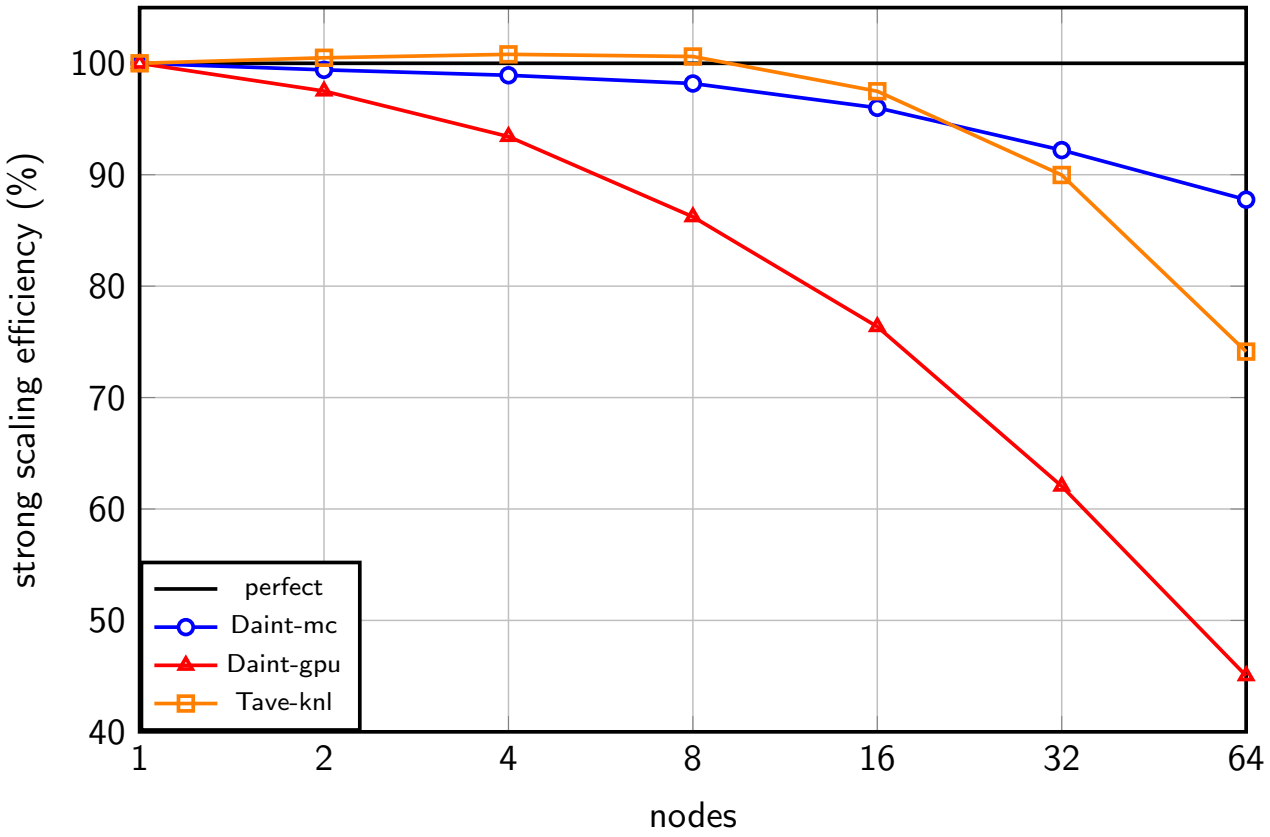
Strong scaling: minimizing time to solution for a fixed model size with increasing number of nodes



- For less than 4k cells (on 4 nodes) **multicore and GPU are equivalent** (within 10% range)
- For more than 4k cells **multicore is faster**
- A KNL node is uniformly slower than multicore, using **1.4x more time**
- Still, Arbor can be **used effectively** on an HPC system available

PERFORMANCE

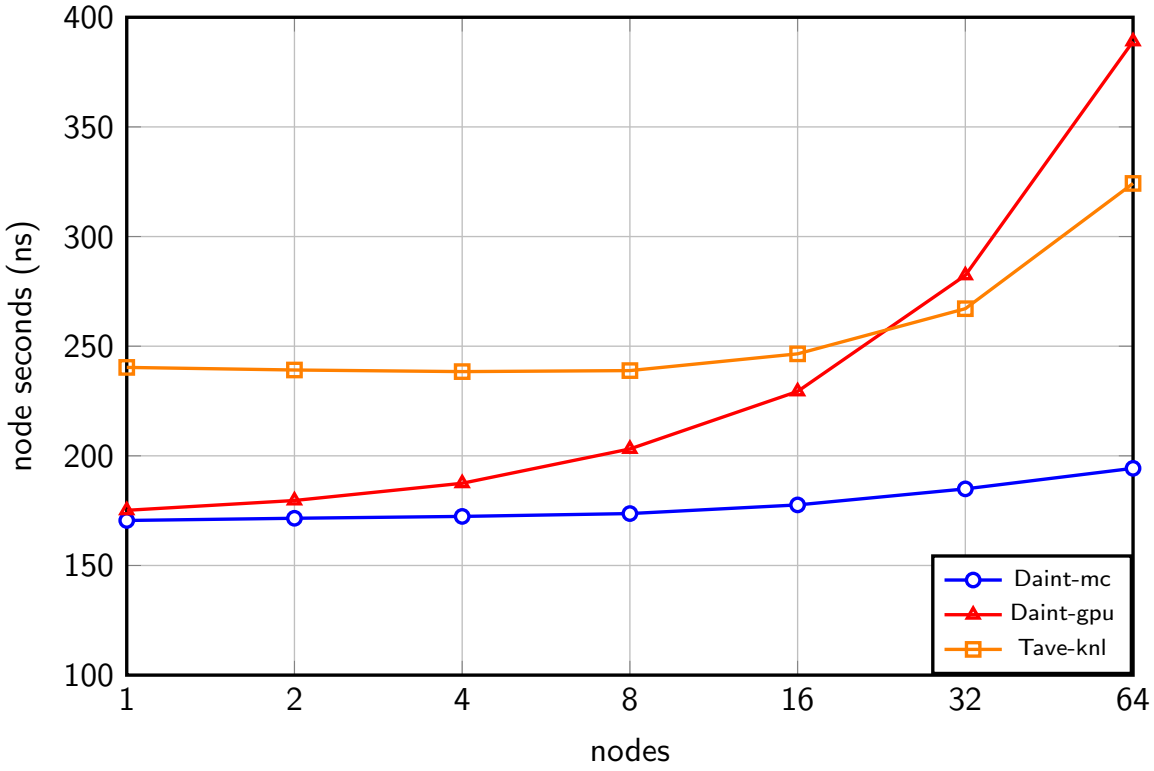
Strong scaling efficiency

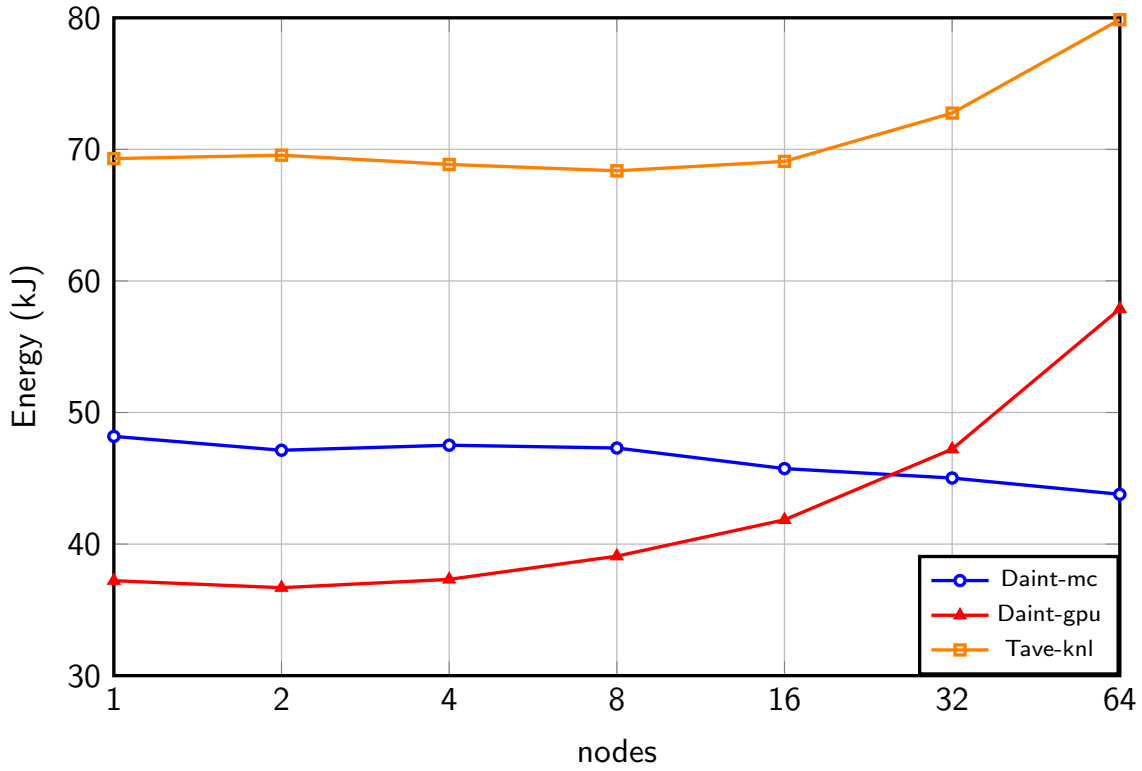


- **Resource utilization is effective** where strong scaling efficiency is good
- **Efficiency decreases** as the number of nodes increases
- Only the multicore system scales with **90% efficiency** to 64 nodes (256 cells per node) and minimizes time-to-solution
- **GPU system is still effective** for running large models

PERFORMANCE

Strong scaling: consumed resources in node-seconds and energy consumption





PERFORMANCE BENCHMARKS

Setup of dry-run mode on HPC architecture

System

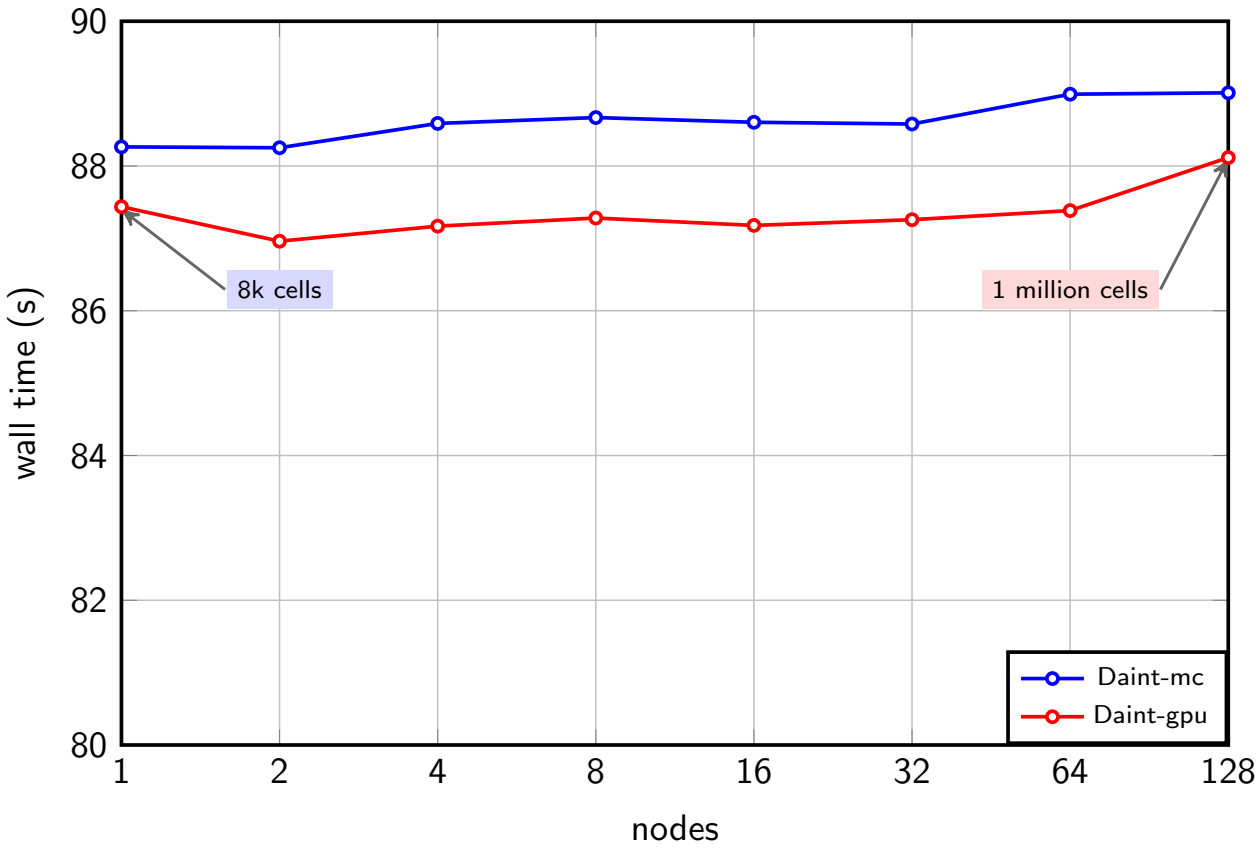
	Daint-mc	Daint-gpu
CPU	Broadwell	Haswell
memory	64 GB	32 GB
cores/socket	18	12
threads/core	2	2
vectorization	AVX2	AVX2
accelerator	–	P100 GPU
MPI ranks	2	1
threads/rank	36	24
configuration	–	CUDA 9.2
compiler	GCC 7.2.0	GCC 6.2.0

Dry-run mode

- Model: 100 ms simulation with 10 ms delay and cells firing at 87.5 Hz
  each cell connected to 10 000 random cells with no self-connection
- Mode: Run model on single MPI rank, and mimic running on a large cluster
  (here: 10 000 nodes) by generating proxy spikes from cells on other ranks
- Cells/ node: 1000 & 10 000 cells per node for total model size of 10 M & 100 M cells

PERFORMANCE

Weak scaling is near perfect



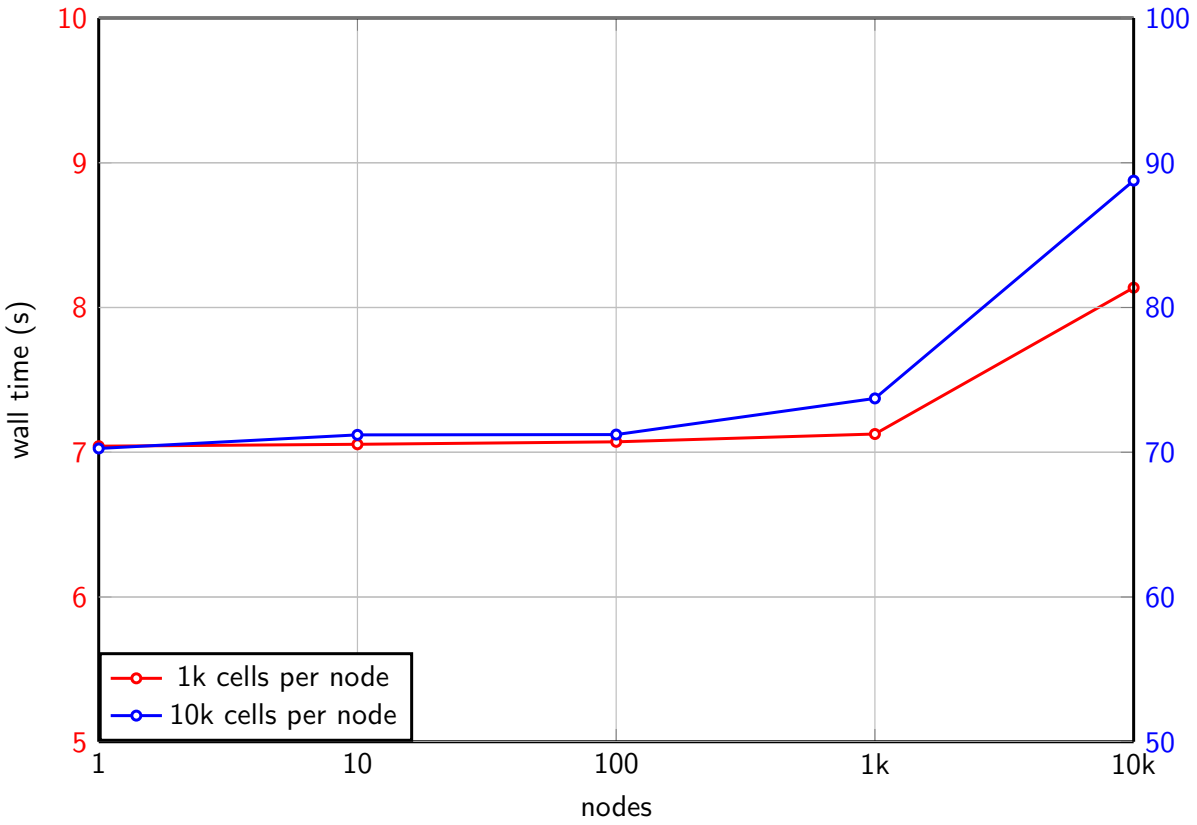
Maximize model size while increasing number of nodes with fixed number of cells

To hundreds of nodes

- Arbor **weak scales (near) perfectly** on multicore and GPU

PERFORMANCE

Weak scaling sufficient with 80% at extreme scale



To 10 000 nodes

- 1000 nodes: 1k and 10k models weak scale very well with **99% and 95% efficiency**
- 10 000 nodes: weak scaling still good with 87% and 79% for 1k and 10k models, but **decreased due to spike communication and processing**

PERFORMANCE

Summary

- Arbor has been tested on a variety of vectorized CPU architectures, showing significant improvement over compiler auto-vectorization
- Synthetic networks have been tested on multicore, GPU and KNL architectures
  - Close to linear single node scaling, with comparable performance at >1000 cells
  - More memory efficient than standard NEURON, with speedup's of 5-30x as cell numbers increase
- Strong scaling has been shown for up to 10k cells with good energy consumption scaling
- Weak scaling is near perfect up to 128 nodes (1 million cells)
  - Even at 10k nodes, weak scaling is still at 79%

CONCLUSION

Summary

- Arbor is an extensible library for multicompartment neuron models
- It is designed with the goal of optimizing usage of current HPC architectures and is ready to be ported to future architectures
- Development is fully open, developed from scratch, developed by software engineers at supercomputing centers
- It uses standard cell and network formalisms with a focus on performance
  - A subset of NMODL descriptions can be used
  - A python interface is under development
- We have focused on synthetic verification, testing and performance benchmarks
  - Current architectures are standard cpus, vectorized cpus, many core and GPUs
  - Weak and strong scaling have been shown up to 10 000 nodes
  - 5-30x faster than standard NEURON for tested morphologies and networks

QUESTIONS