NestMC

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

Wouter Klijn[a], Ben Cumming[b], Alexander Peyser[a], Vasileios Karakasis[b], Stuart Yates[b]
[a]Simulation Lab Neuroscience, Forschungszentrum Jülich [b]Swiss National Supercomputing Center

Why NestMC?

Local Field Potential

Both as model output and as part of model feedback loop.

Larger and Longer!

Running more or larger models faster to improve parameter searches and statistical validation.

Dynamic Models

Time-varying morphology, connectivity and synapses.

New and emerging HPC *many core* architectures can achieve these aims

Two HBP prototype systems installed at Jülich. Both are a radical departure from current technology. *left*: IBM Power8+GPU "fat node". *right*: Intel many core KNL "blade".

Current simulators were designed for single core systems, with parallel implementations added later. There are efforts to add many core support to existing codes, however they are subject to the law of diminishing returns. This presents an opportunity to start work on the next generation of simulators, designed from the ground up to support diverse many core architectures. *NestMC* aims to fill this gap.

Who

NestMC is developed by a team from three HPC centers at Jülich, CSCS and BSC.

- Part of HPC infrastructure work package in the *Human Brain Project*.
- With the *NEST Initiative*.
- The centers are motivated to prepare neuroscience users for new HPC architectures.
- We provide know-how in computer science, math and software development.

How

NestMC is designed from the ground up for *many core* architectures.

- Written in modern C++, CUDA, Intel Threading Building Blocks and HPX.
- Is *open source*.
- Uses sound development practices including *unit testing*, *continuous Integration*, and *validation*.
- Aims to be user and community-driven: you can help!

Prototype

Start with a prototype

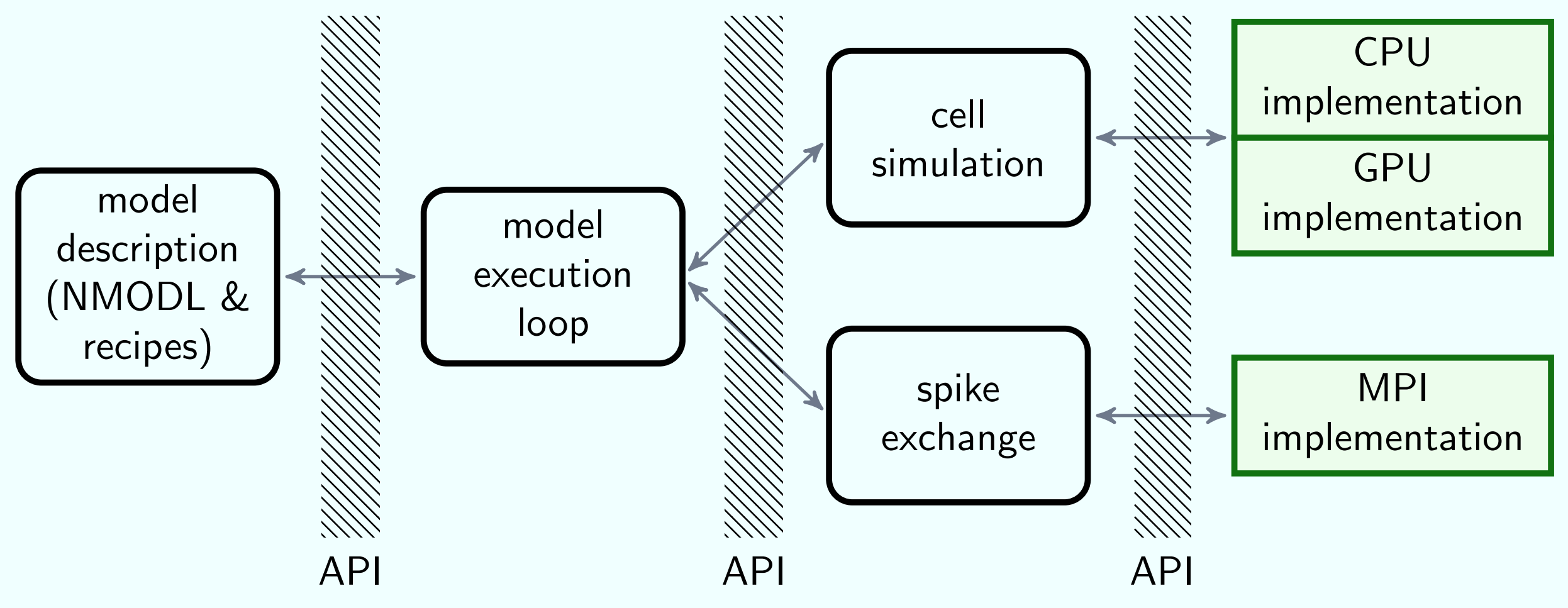
The first step is to develop a *sufficiently complex* prototype to inform important design considerations:

- Explore the trade-offs between abstraction and performance.
- Support for representative architectures: multicore, Intel KNL, and GPU.
- The features in the prototype test our assumptions and inform the design:
  - Test *interopability* with an external API (e.g. LFP).
  - Test the *extensibility* of the internal API used to implement algorithms (e.g. gap junctions).
  - Test an exotic *back end* (e.g. GPU).
- Aim to understand the design space and domain before committing to a design.

Prototype progress

- Support for x86 and KNL (GPU partially).
- Supports NMODL for ion channels and synapses.
- Distributed building of networks up to millions of cells.
- Finite volume discretization of the cable equation.
- Generic network connection model distributed via MPI.
- Asynchronous spike communication and computation.
- Validated against Neuron.

Internal prototype APIs



The modular internal simulation code in the prototype is designed to allow plug and play of different simulation and communication implementations.

Do you want to know more?

The source code of the prototype is not in an open repository. If you would like to know more, contribute or collaborate, please contact us directly for access to the code.

The source, tests and validation for NestMC will be in an open repository by April 2017 at the latest.

email	bcumming@cscs.ch a.peyser@fz-juelich.de
web	eth-cscs.github.io/nestmc

User-driven development is the key!

Current tools and technology influence the ideas that users consider.

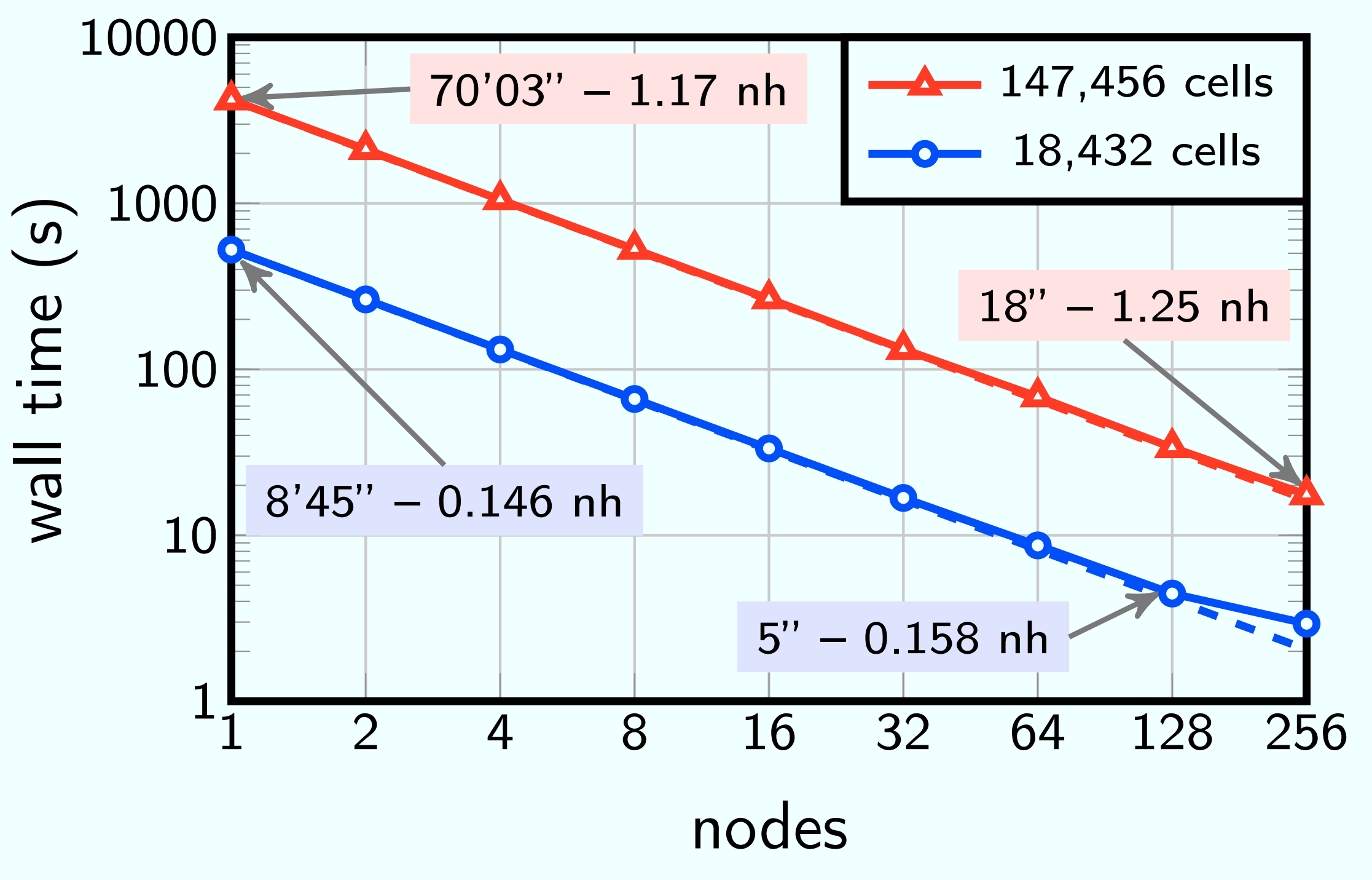
We need help from the neuroscience community.

- Tell us which features you need.
- Help us build a simulator for your community.

Performance of prototype

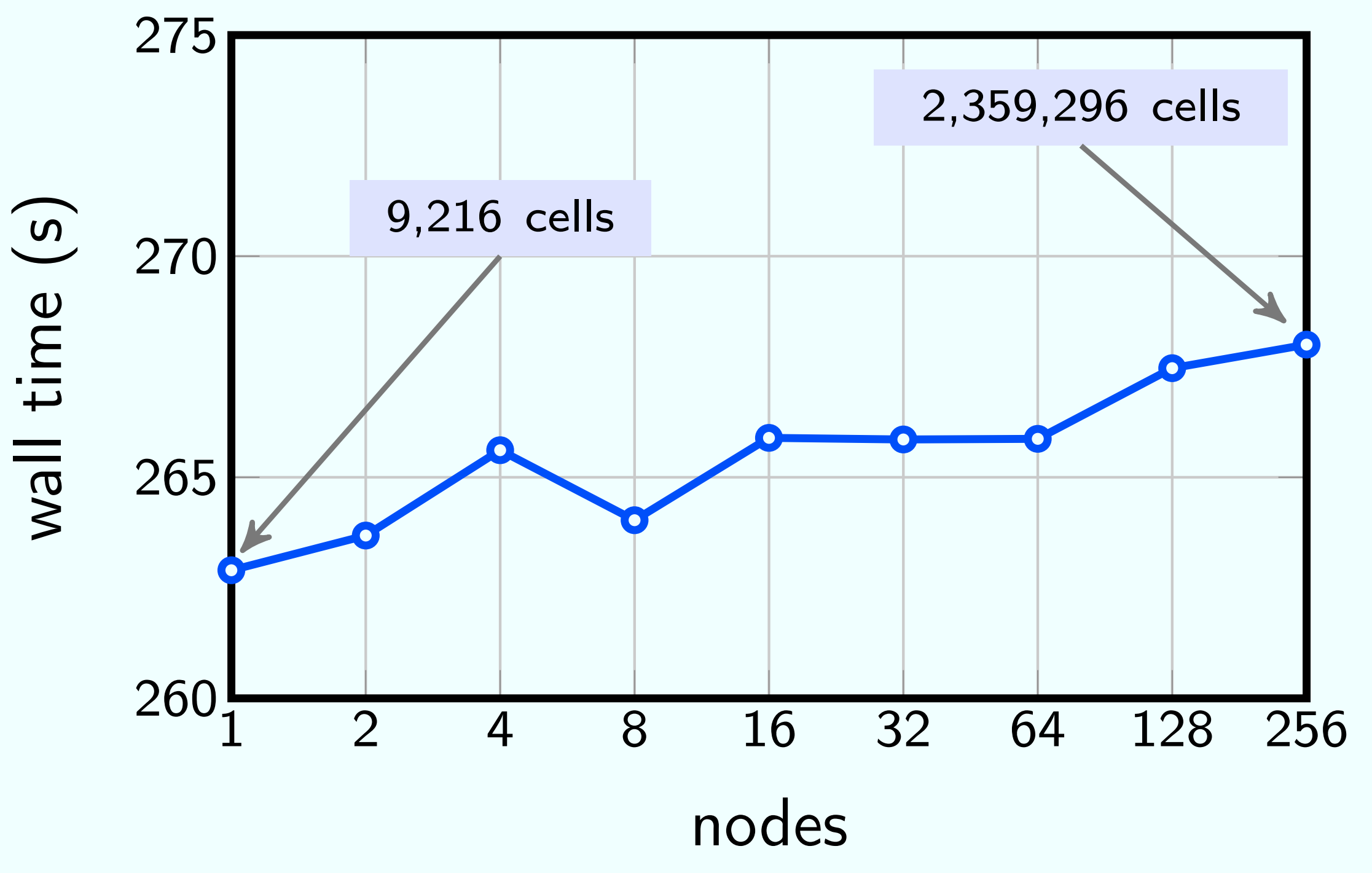
cluster	Cray XC40: 36 cores & 64 GB per node
cells	350 compartments & 2000 synapses each Passive dendrites, Hodgkin–Huxley soma
duration	500 ms

Reduce wall time with more nodes



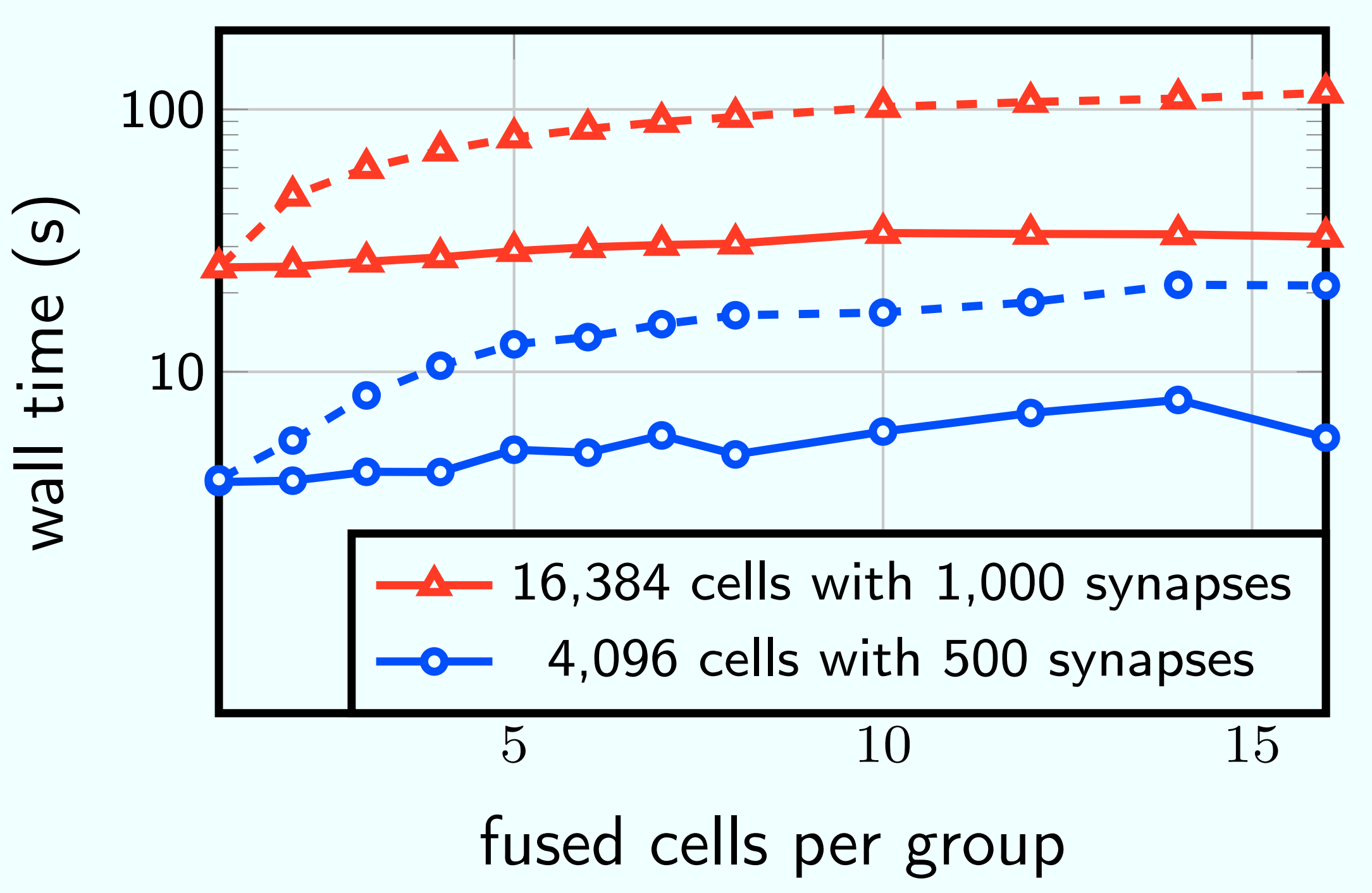
Time to solution for two models of fixed size as a function of the number of compute nodes. Increasing the number of processors reduces the time to solution, with little increase in the compute resources required, measured in *node hours* (nh).

Scale out model size



Time to solution with 256 cells assigned to each core, as the number of nodes is increased. This way one can scale up to large networks without significantly increasing wall time (about 9 minutes per simulated second for over 2 million cells on 256 nodes for this model).

Using high-speed memory on KNL



Two models are run on KNL with and without high-speed memory (solid and dashed lines respectively). The results show the effect of fusing multiple cells into groups, which will be an important optimization for some models and architectures.