CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

arbor

ArborIO Collaboration Kick Off

Team Arbor
February 9, 2021

Human Brain Project

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 720270 (Human Brain Project SGA1), Specific Grant Agreement No. 785907 (Human Brain Project SGA2), and Specific Grant Agreement No. 945539 (Human brain Project SGA3).

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Welcome!

Some introductions.

Arbor

`Arbor` is library for simulation of morphologically-detailed cells in large networks on HPC systems.

- **key aim**: enabling simulation on all HPC systems.
- **key aim**: provide rich interface enabling diverse use cases.

All features are implemented and optimised on **all platforms**

- All GPUs (CUDA, HIP, Clang-CUDA)
- SIMD CPU backends: (AVX, AVX2, AVX512, Neon, SVE).
- Distributed simulation (MPI).

There is a non-trivial amount of software complexity required to meet these obligations.

Models are varied and complicated

Drawing of a Purkinje cell in the cerebellar cortex by Santiago Ramòn y Caja.

Performance Portability

Portability has two aspects:

1. **Performance portability**, software that:
   - run efficiently and scale on different architectures;
   - can be adapted to support new architectures.
2. **Model portability**, model descriptions that are:
   - simulator and architecture agnostic;
   - **flat**, which means:
     - what, not how.
     - translated, not interpreted.

Arbor's design is constrained and complicated by meeting these two requirements.

Step 1: Model abstraction (which model)

User models are described by a **recipe**, which take a cell number and give:

- a description of the cell
  - piecewise linear morphology
  - named regions and locations
  - ion channel and synapses
- spike targets
- spike sources
- network connections that terminate on the cell

Recipe descriptions are functional, with lazy evaluation for efficient parallel model construction.

Recipes contain no hardware or implementation details.

Step 2: Hardware context (which hardware)

Select hardware resources

```
import arbor
from mpi4py import MPI

rec = my_recipe() # user defined model
ctx = arbor.context(threads=12, gpu_id=0, mpi=MPI.COMM_WORLD)
```

Users can select hardware resources at run time:

- Number of threads in thread pool
- Which GPU [optional]
- Which MPI communicator [optional]

Step 3: Instantiate model

Instantiate model on target compute resources

```
import arbor
from mpi4py import MPI

rec = my_recipe() # user defined model
ctx = arbor.context(threads=12, gpu_id=0, mpi=MPI.COMM_WORLD)
sim = arbor.simulation(rec, ctx)
```

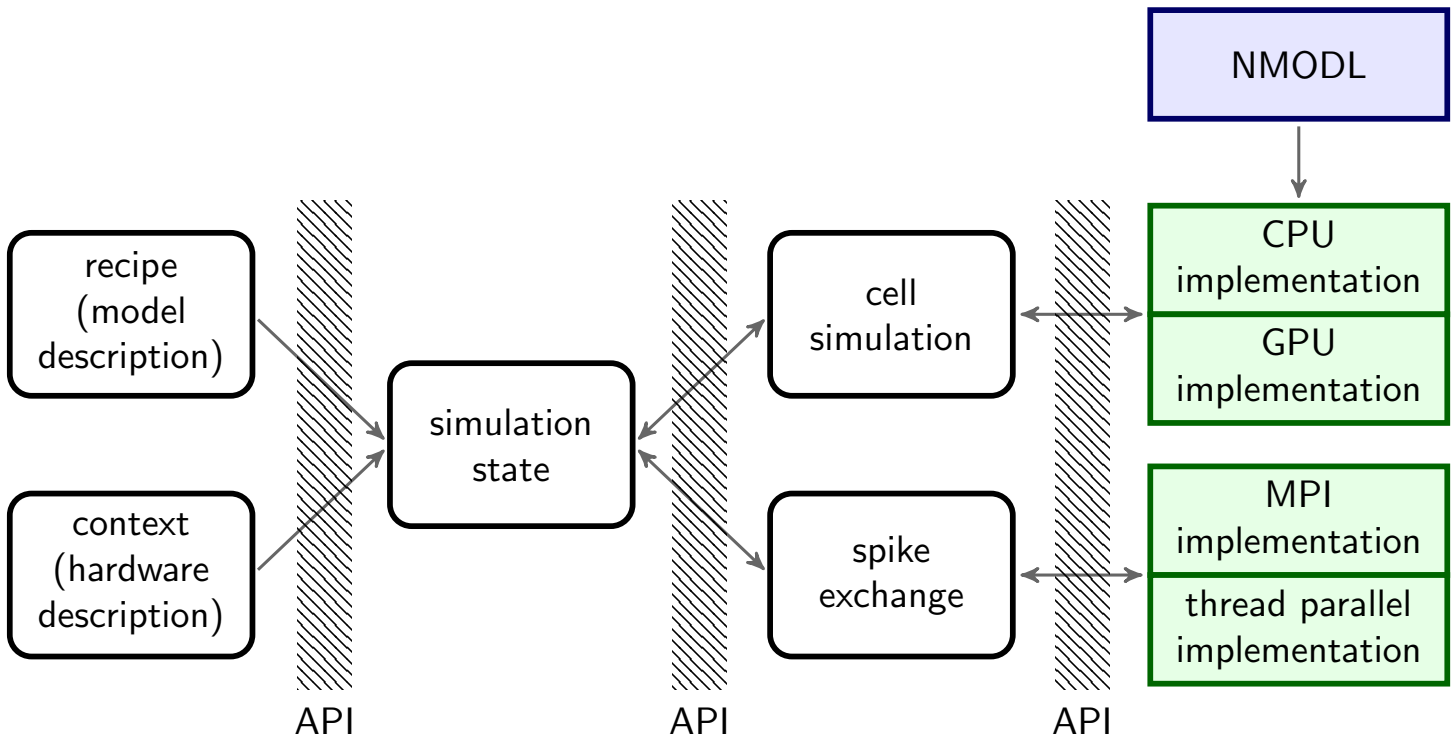
A simulation object:

- Instantiates target-specific data structures and call backs
- Provides a generic interface for:
  - steering simulation
  - sampling spikes, voltages, etc.
- Has no global state
  - multiple simulations can be instantiated simultaneously.

Caller can optionally provide hints on how to assign model to hardware resources.

Simulation Architecture

Components communicate via APIs allow that allow implementation of new cell models, communication methods, hardware back ends etc.



Arbor's time stepping

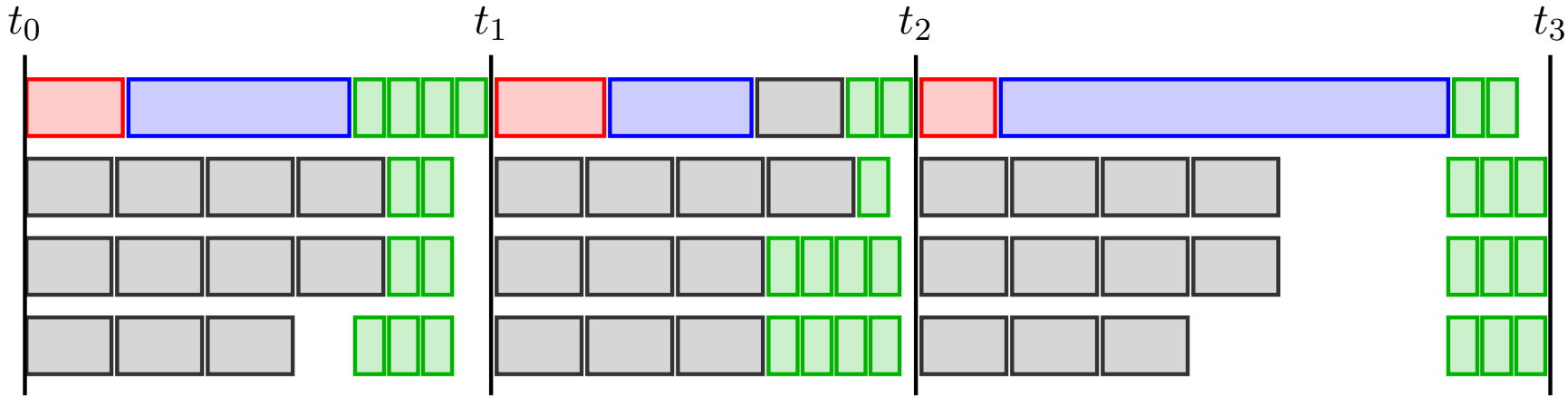
MPI communication is only used for spike exchange.
Spike exchange every `min_delay/2` → overlap communication and computation.

```
for ts = 0:min_delay/2:tfinal:
    task:
        gather spikes from epoch-1
        walk spikes -> events for epoch+1
    task:
        parallel_for group in cell_groups:
            prepare events for epoch
            integrate group over epoch
end
```

- Cell groups composed of 1 or more cells
- Cell groups are distributed over thread pool or GPU for integration over an epoch
- Fine grained parallelism is on cell groups

Arbor's time stepping

Integration of 11 cell model for 3 intervals on 4 core



MPI communication; spike walk; cell update; event enqueue.

- Idle time increases when serial tasks take more time than cell updates.

Gap Junctions

Arbor is optimized to perform multiple time steps in a cell group without interruption

- **CPU**: small cell groups retain all cell state in cache.
- **GPU**: one time step requires at least 10 kernels: the kernels for multiple time steps can be enqueued ahead of time and launch overheads ammortised.

The only biophysical model that isn't distributed is gap junctions:

- Cells connected with a gap junction are integrated together.
- Achieved by placing them in the same cell group.
- Hard scaling limits when many cells are interconnected with gap junctions.

Performing MPI communication of potentials at gap junction sites is not practical with Arbor's architecture.

Aims for this meeting

1. Understand the scale of proposed models
   - Hint: 10,000-50,000 cells won't fill a single node on LUMI
2. Define one (maybe two?) simple bootstrap model for the collaboration.

An Idle Observation

1. The LUMI system will go into production in early 2022.
2. It will use AMD GPUs
3. Arbor already supports AMD GPUs
4. Not many applications will have full support for AMD GPU on day 1
5. Proposals that can demonstrate AMD support would have a very good chance of getting large allocations.

Just saying!