Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

The NESTio project – replacement data recording backend for NEST

NEST Conference 2017

December 19, 2017

Jochen M. Eppler (j.eppler@fz-juelich.de)
Alexander Peyser (a.peyser@fz-juelich.de)
Wolfram Schenck (wolfram.schenck@fh-bielefeld.de)

Outline

Motivation

The `RecordingBackend` interface

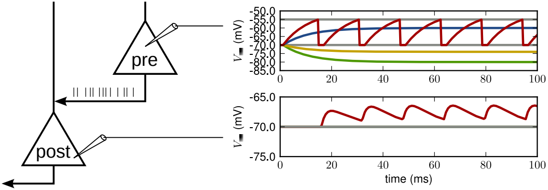
Parallel I/O with SIONlib

Benchmarks and performance

Current status and future work

Current situation

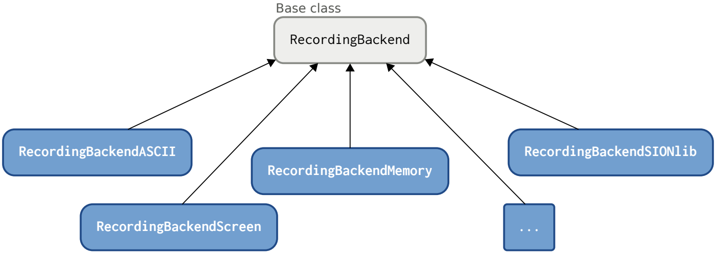
Simulation data is recorded by virtual spike detectors and multimeters. Data is regularly written file (one per thread)



Problem: The current implementation does not scale well to large simulations on large clusters or supercomputers with several hundreds to tens of thousands of compute nodes.

NESTio recording backends

Main idea: Recording device output can be redirected to various different file formats and interfaces, implemented in specific*backend classes*.



The `RecordingBackend` interface

- **void pre_run_hook()**

  Initialize global backend-specific data structures. Called on each backend on simulator startup as well as upon changes in the number of threads.

- **void synchronize()**

  Synchronize backends at the end of each simulation cycle. Called once per simulation cycle *after* all nodes have been updated.

- **void post_run_hook()**

  Clean the backend up at the end of a simulation run. Called right before `SimulationManager::run()` terminates and allows the backend to flush open files, write remaining data to the screen, . . .

The `RecordingBackend` interface

- **void cleanup()**

  Clean up the backend at the end of a call to Simulate. Called by `SimulationManager::cleanup()` and allows the backend to close open file, network connections, . . .

- **void clear(const RecordingDevice& device)**

  Discard all data recorded for a recording device. Called when the user sets the property *n_events* on `device` to `0`.

The `RecordingBackend` interface

- **void enroll(const RecordingDevice& device)**

  Register a `device` to record data using a certain backend. Variant to be called by each *digital* recorder during network calibration.

- **void enroll(const RecordingDevice& device,**
  const vector< Name >& value_names)

  Register a `device` to record data using a certain backend. Variant to be called by each *analog* data recorder during network calibration.

The `RecordingBackend` interface

- **void write(const RecordingDevice& device,**
  const Event& event)

  Write the data from the `event` to the backend specific channel. If the `device` is not enrolled, this function has to return immediately.

- **void write(const RecordingDevice& device,**
  const Event& event,
  const vector< double >& values)

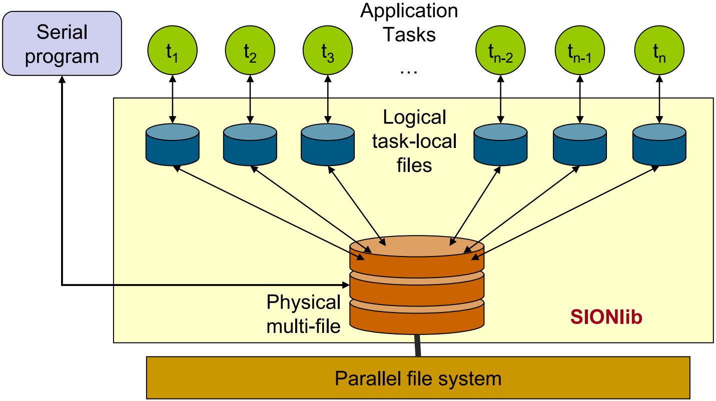
  Write the data from the `event` to the backend specific channel together with the `values` given. If the `device` is not enrolled, this function has to return immediately.

Parallel I/O with SIONlib

Particularly appropriate for large-scale simulations on supercomputers is the implementation of a backend writing to SIONlib container files.

Container concept

Individual ranks write to task local streams that map into a physical multifile



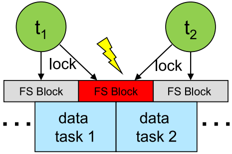
Wolfgang Frings, 2016

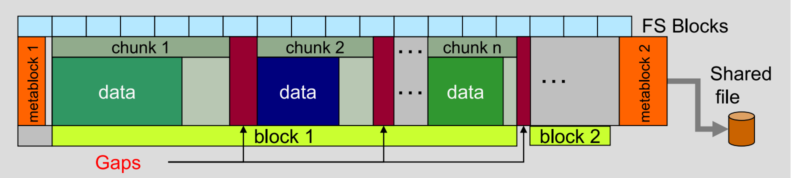
Data is a weakly ordered stream optimized for writing, with a final metadata block describing the devices, measures and neurons in the file.

File format

Logical partitioning of shared file(s)

- Dedicated data chunks per task, one or more
- Alignment to boundaries of file system blocks (no contention)
- Metadata blocks: e.g. chunk position, size and usage
- Contention when writing to same file-system block in parallel

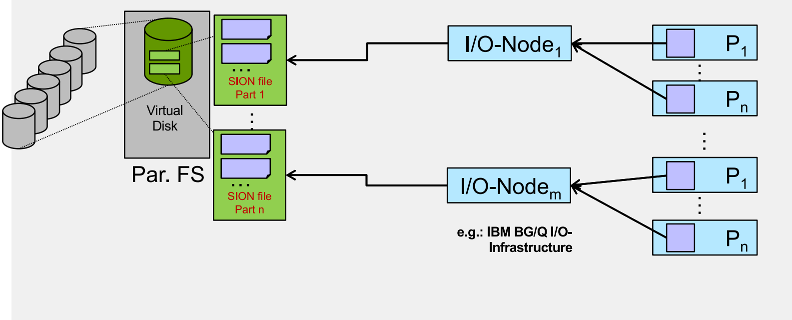




Wolfgang Frings, 2016

Multiple files per container

For 10k's of nodes with complex GPFS topologies

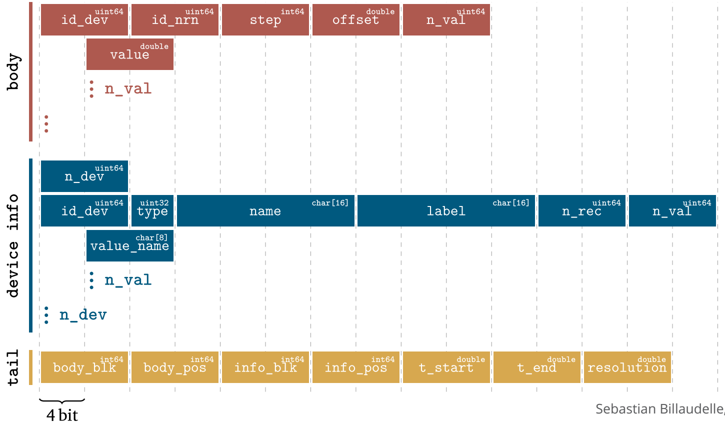


Wolfgang Frings, 2016

Nodes are split by common I/O paths sharing one file, but belonging to the same container. Seen by the postprocessor as a single unit with common metadata.

NEST IO Sionlib format

Format is self-describing with a tail metadata block



Sebastian Billaudelle, 2015

Plus Sionlib entry metadata (such as endianness) and final pointer to tail chunk. Metadata is stored on task 0 stream.

Benchmarks and performance (1)

Benchmarking framework: Random balanced network with 11,250 neurons per compute node (= MPI rank)

- Incoming synapses per neuron: 11,250
- Excitatory (80%) and inhibitory population (20%)
  (one spike recorder and one multimeter per population)
- STDP for excitatory synapses
- Random driving input causes mean spike frequ. of ca. 3 Hz
- Simulated time: 515 ms

Benchmarking platform: JUQUEEN

- Weak scaling: Number of ranks M varied between 32 and 2048
- Number of threads T=16

Benchmarks and performance (2)

Code variations:

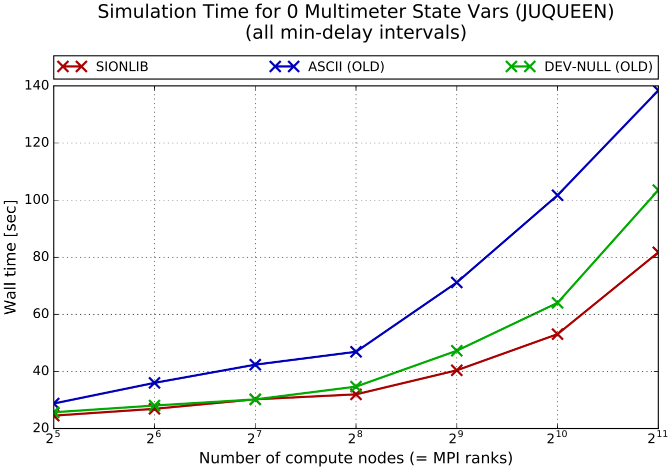
All based on commit #4a373f7c7 (March 2017)

- **ASCII (OLD):** Standard recording backend, output to text files (one file per thread per recording device)
- **DEV-NULL (OLD):** Standard recording backend, output to `/dev/null` (approximation of infinitely large bandwidth and zero latency)
- **SIONLIB:** New recording backend, output to multiple sionlib container files (number automatically determined by sionlib) [#602ca2e153]

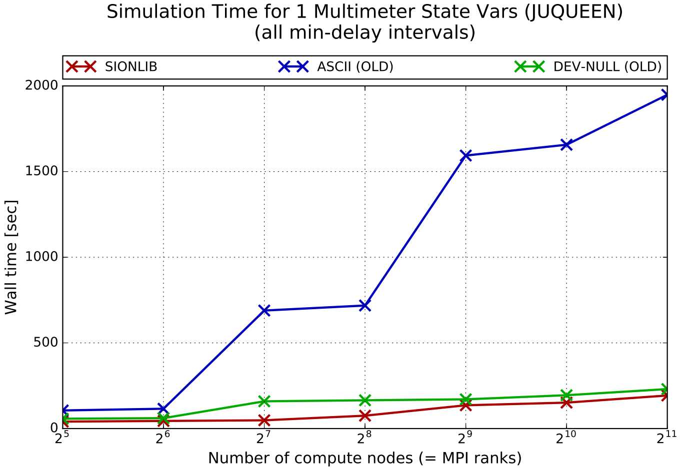
Amount of written data:

- **Only spikes:** Multimeters completely disabled
  ASCII: 0.25 MiB per Node; SIONLIB: 0.6 MiB per Node
- **Spikes and one state var.:**
  ASCII: 250 MiB per Node; SIONLIB: 490 MiB per Node
- **Spikes and four state vars.:**
  ASCII: 470 MiB per Node; SIONLIB: 750 MiB per Node

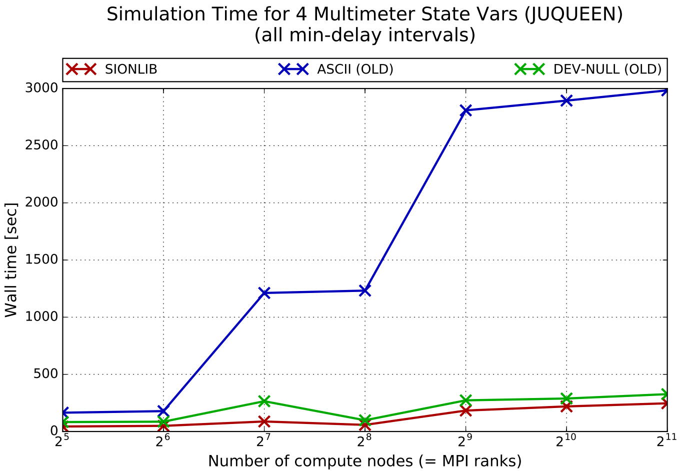
Overall simulation time (only spikes)



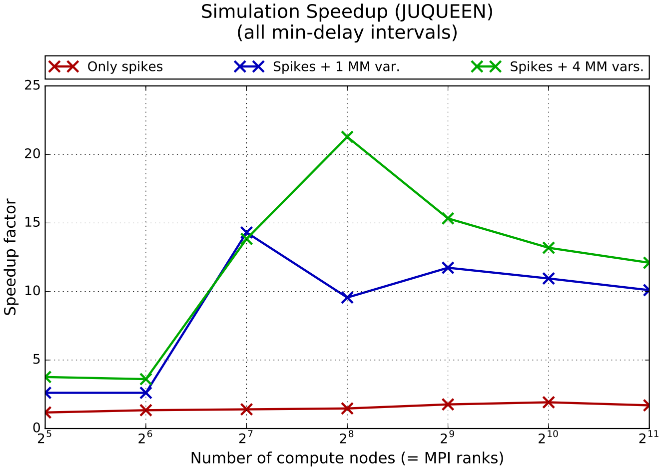
Overall simulation time (spikes + one state var.)



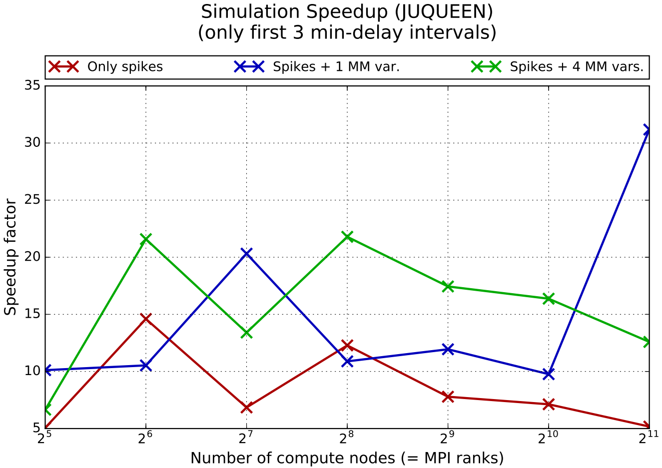
Overall simulation time (spikes + four state vars.)



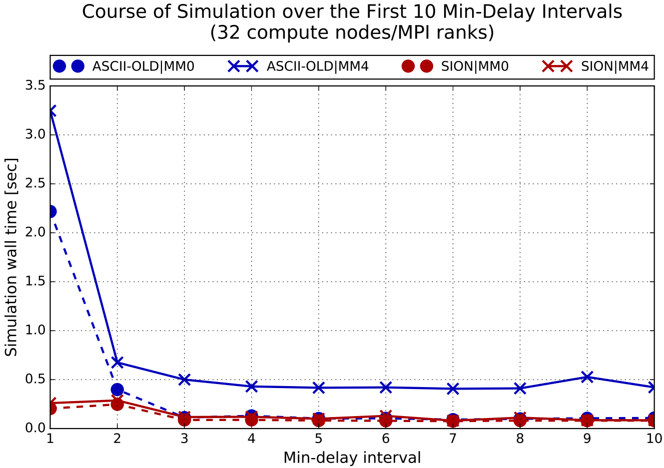
Overall simulation speedup



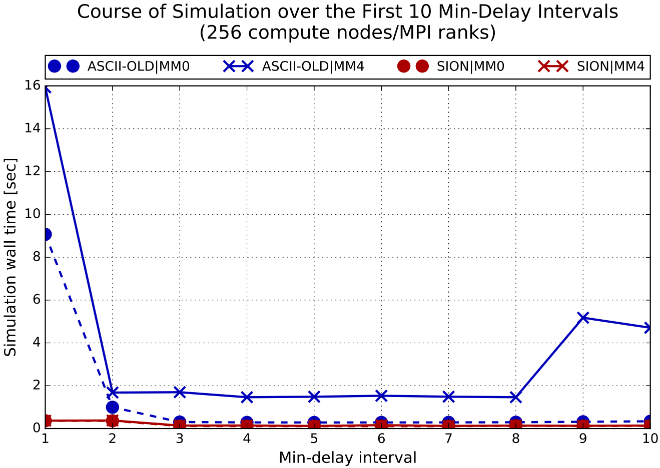
Simulation speedup (first 3 min-delay intervals)



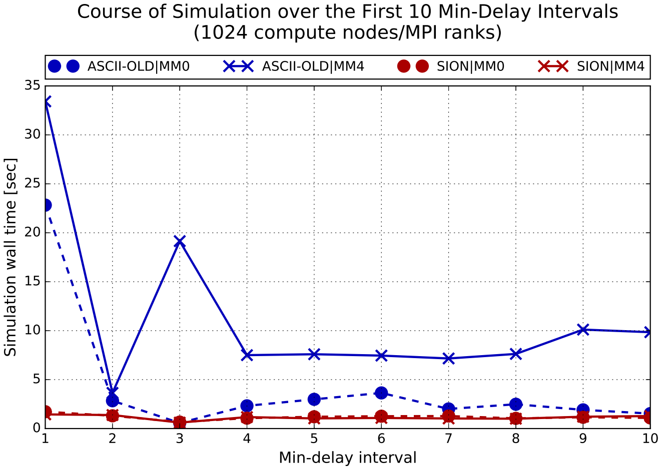
Simulation time course (first 10 min-delay int.) (1)



Simulation time course (first 10 min-delay int.) (2)



Simulation time course (first 10 min-delay int.) (3)



Benchmarks and performance: Summary

- **Surprise:** New SIONLIB backend faster than old recording backend writing to `/dev/null`
- **Only spikes:** Speedup by factor two for all simulation sizes (compared to old ASCII backend)
  - Most likely getting larger the more spike detectors used
- **With multimeters:**
  Speedup between 10 and 20 for M >= 128
- **During the first three min-delay intervals:**
  Even larger speedups
  - Reason: File creation in the very beginning of the simulation (metadata overkill when operating on single files)

Future work and outlook

Status: Code is mostly ready. Some devices are not supported yet and some cleanup needs to happen.

- Finish the benchmarks to make sure NESTio performs well
- Clean up the code and start a PR against NEST master
- Provide a backend for Neo to read the binary SIONlib files
- Write documentation and examples
- Make the **i** in NESTio a reality