Streaming Live Neuronal Simulation Data Into Visualization and Analysis

Simon Oehrl[1,5] [0000−0001−6504−2293], Jan Müller[1] [0000−0002−0126−9293], Jan Schnathmeier[1] [0000−0002−8520−8243], Jochen M. Eppler[2] [0000−0002−3145−3040], Alexander Peyser[2] [0000−0002−3453−310X], Hans Ekkehard Plesser[3,4,6] [0000−0001−7843−5993], Benjamin Weyers[1,5] [0000−0003−4785−708X], Bernd Hentschel[1,5] [0000−0002−2642−9134], Torsten W. Kuhlen[1,5] [0000−0003−2144−4367], and Tom Vierjahn[1,5] [0000−0002−8620−3874]

[1] Virtual Reality and Immersive Visualization, RWTH Aachen University, Germany
oehrl|kuhlen@vr.rwth-aachen.de, tom.vierjahn@acm.org
http://vr.rwth-aachen.de

[2] SimLab Neuroscience, Forschungszentrum Jülich GmbH, Institute for Advanced Simulation, Jülich Supercomputing Centre (JSC), Jülich, Germany
https://www.fz-juelich.de/ias/jsc/slns.html

[3] Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6), Forschungszentrum Jülich GmbH, Germany
http://www.csn.fz-juelich.de

[4] Faculty of Science and Technology
Norwegian University of Life Sciences, Ås, Norway
http://www.nmbu.no/imt

[5] JARA-HPC, Aachen, Germany
http://www.jara.org/en/research/hpc

[6] JARA-BRAIN Institute I, Jülich, Germany
http://www.jara.org/en/research/brain

Abstract. Neuroscientists want to inspect the data their simulations are producing while these are still running. This will on the one hand save them time waiting for results and therefore insight. On the other, it will allow for more efficient use of CPU time if the simulations are being run on supercomputers. If they had access to the data being generated, neuroscientists could monitor it and take counter-actions, e.g., parameter adjustments, should the simulation deviate too much from in-vivo observations or get stuck.

As a first step toward this goal, we devise an in situ pipeline tailored to the neuroscientific use case. It is capable of recording and transferring simulation data to an analysis/visualization process, while the simulation is still running. The developed libraries are made publicly available as open source projects. We provide a proof-of-concept integration, coupling the neuronal simulator NEST to basic 2D and 3D visualization.

Keywords: neuroscientific simulation · in situ visualization

1 Introduction

In order to understand the complex in-vivo processes inside the human brain, neuroscientists work on creating realistic in-silico simulations of subsets of it running on a computer. They create models consisting of individual neurons and their connections. Furthermore, they set the neurons' and the connections' parameters to values that have been determined via prior experiments – either in vivo or in silico. They also use values that they consider reasonable first guesses in order to determine ones that lead to realistic behaviour. A simulation is then run and reveals the behaviour of the neuronal network with the current set of parameters. Due to the complexity of the neuronal networks, the simulations are often run in a massively parallel fashion on supercomputers.

Neuroscientists analyse the acquired data once the simulation has terminated. They compare the simulated data to in-vivo data. For this purpose, they apply statistics and visualizations, for instance, line plots revealing the individual neurons' membrane potentials or raster plots presenting the spikes emitted by the neurons. From observed differences between simulated and the in-vivo data the neuroscientists devise updates to the model and the set of parameters. The updated neuronal model is then again simulated, analysed and re-assessed.

This iterative process of simulation and subsequent analysis (e.g., [20]) requires the neuroscientists to wait for the simulation to be finished before the model and its parametes can be adjusted. Interactive simulations are still very primitive, non standardized, and ad-hoc: means of looking into a running simulation exist (e.g., [16]), but they are limited. Neuroscientists can hardly take counter-actions to simulations that deviate too much from real-life behaviour or that show erratic behaviour beyond recovery. If this happens early during a simulation, scarce CPU time is wasted, without yielding new insight. On the contrary, that time cannot be used for other experiments waiting in the job queue – submitted by other scientists or for simulations using updated parameters.

If the neuroscientists were able to visualize their measurements during a live simulation, they could adjust parameters on the fly. That way, the parameters could be adjusted more frequently, using the allotted compute time more efficiently, providing insight earlier. To date, the only way mimicking such an interactive simulation requires to run it for very short intervals, halt it, analyse/visualize the data, adjust the model and its parameters, and eventually resume the simulation. This is tedious since the neuroscientists have frequently to restart even well-behaving simulations without applying any changes. More importantly, resuming a simulation introduces an overhead due to additional initialization time spent inside the simulator. This overhead will add up, so that currently the neuroscientists trade interactivity for better CPU time usage.

To overcome this, we devise a framework for streaming simulation data from a simulator to a separate analysis/visualization process. The framework is separated into two lightweight parts. The one converts the simulation data into a simulator-agnostic transfer format and provides the analysis/visualization process with means to access that data. The other provides a thin layer for transporting data packets in the transfer format. This separation facilitates adaptation of

the framework to other simulators. Furthermore, we expect that the transport layer will be useful to in situ applications in fields other than neuroscience.

In the remainder of this paper Section 2 gives an overview of the related work. Section 3 presents the in situ framework we propose, Section 4 outlines our proof-of-concept integration of simulation and visualization, Section 5 evaluates the performance of the proposed framework. Section 6 concludes the paper and gives an outlook on future work.

2 Related Work

Simulating a neuronal network in the computer resembles an electrophysiological experiment. The neuroscientist builds a neuronal system by creating one or more model neurons and connecting them via synapses to form a network. After running the simulation, they analyse the measured data.

The NEST simulator [8] is able to work with large, heterogeneous networks of spiking neurons – either point neurons or neurons with a small number of compartments. NEST can represent spikes in continuous time [14]. It allows for simulating different types of model neurons and synapse models in one network. Arbor [5] is a new multi-compartment neuronal network simulator currently under active development. It is specifically designed for many-core architectures and provides optimized backends for CUDA, KNL, and AVX2. Arbor features asynchronous spike exchange that overlaps compute and communication, and therefore hides latencies. It will enable new scales and classes of morphologically detailed neuronal network simulations on current and future supercomputing architectures. NEURON [9,12] is capable of simulating individual neurons and networks of neurons. The related models are closely linked to experimental data. NEURON supports the neuroscientist in gaining insight without requiring in-depth expertise in numerical methods or programming by providing convenient tools for constructing, exercising, and managing neuron models.

Neuroscientists will look at the simulation results using statistical analysis tools as well as visualizations. They use either file I/O to write the simulation data to disk and read it back, or they use more elaborate transport mechanisms. MUSIC [7] is a standard API for exchanging data between simulators. It provides mechanisms to synchronize the simulations and map their data models to each other. NEST and NEURON provide interfaces to MUSIC. However, MUSIC primarily facilitates inter-operability between neuronal simulators within a parallel computer during runtime. This paper therefore provides a lightweight alternative specifically tailored to coupling simulations and visualizations. `Nett`[7] [17] provides a similar approach in this regard, building on top of ZeroMQ[8]. Furthermore, it can be used to steer simulations [16]. ZeroEQ[9] follows a similar approach.

[7] `https://devhub.vr.rwth-aachen.de/VR-Group/nett`
[8] `http://zeromq.org`
[9] `https://github.com/HBPVis/ZeroEQ`

The scientific visualization (SciVis) community is facing similar in situ visualization challenges as the neuroscience community, albeit on significantly larger scales: while large-scale neuronal network simulations today typically operate on several tens of thousands to a few million neurons with only few recorded attributes, for SciVis the simulation data of a single timestep nowadays easily exceeds hundreds of gigabytes. In order to deal with these amounts of data efficiently, several in situ frameworks have emerged. ParaView Catalyst [2] provides data processing and visualization for in situ analysis and visualization. It works seamlessly with the visualization toolkit VTK [19] and ParaView [1]. LibSim [22] takes a similar approach and facilitates in situ computations within simulation codes. It interfaces VisIt [4] for data analysis and visualization, also building upon VTK. Catalyst and LibSim are pipelines specifically tailored to their visualization applications ParaView and VisIt, respectively. More general pipelines have emerged recently. SENSEI [3] provides a generic in situ interface promoting code portability and reusability, building upon an extended VTK data model. ALPINE [11] is targeting modern supercomputer architectures. It supports Catalyst and LibSim but also provides ALPINE Ascent as its own runtime. Conduit[10] provides a model describing hierarchical scientific data. It provides access to simulation mesh data structures and basic communication functionality. Among others, ALPINE Ascent and VisIt are using Conduit.

All of the above in situ frameworks focus more on the "established" HPC simulations, like computational fluid dynamics or climate simulations, than on the ones used in the field of neuroscience. Therefore, we devise a custom pipeline, developed along use cases emerging from the Human Brain Project [13]. We use Conduit for data description, due to the convenience it provides. We add a lightweight transport layer in order to connect simulation and analysis/visualization. By using Conduit, we expect the pipeline to be compatible to at least some of the existing, mature in situ frameworks, most notably ALPINE. We can therefore later adopt one of them if need be.

3 Method

The devised framework for streaming simulation data from a live simulation to a separate analysis/visualization process consists of two parts. The one (Subsection 3.1) plugs into the neuronal simulation, takes the recorded data and converts it into a simulator-agnostic transfer format; on the analysis/visualization side this part provides access to the stored data. The other part (Subsection 3.2) handles data transport.

3.1 NESCI – Neuronal Simulator Conduit Interface

Neuronal simulators provide means to record the simulated data. NEST [8], for instance, provides among others multimeters and spike detectors as instruments

[10] `https://software.llnl.gov/conduit/`

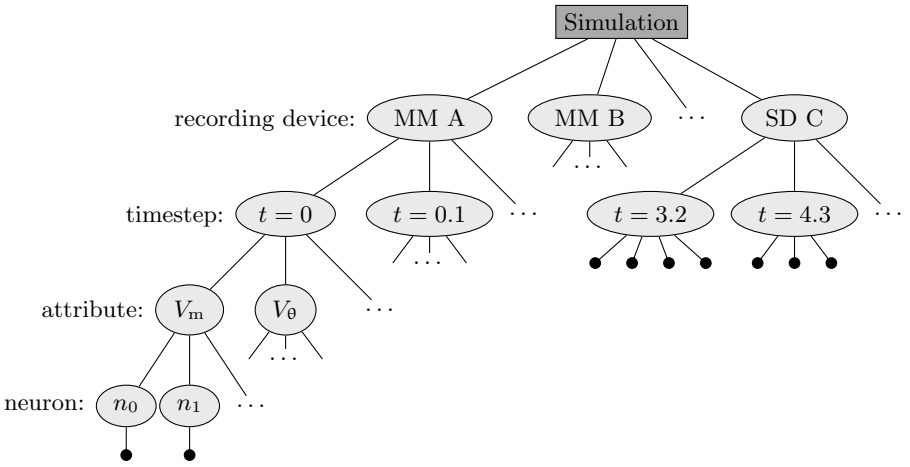

Fig. 1. Hierarchical data layout used for transfer: Each recording device defines a subtree containing data for the stored timesteps. Multimeters (MM A, MM B) store measurements (black circles) of an attribute ($V_\mathrm{m}, V_\theta, \ldots$) for every neuron ($n_0, n_1, \ldots$). Spike detectors (SD C) store a list of neurons that fired in the respective timestep.

that the neuroscientist can connect to either individual neurons or populations of them [21]. Arbor [5] provides similar methods. These instruments are optimized to efficiently work with the respective simulator. Consequently, the interfaces of their associated classes and methods to be called during simulation differ. In order to facilitate connecting any simulator to analysis/visualization, we propose a simulator-agnostic transfer data format.

In this transfer format, data is arranged hierarchically in a tree, since this layout emerges naturally from the concept of data recording in neuronal simulation. Figure 1 presents an overview. The simulation acts as the root node of the data. Each recording device then establishes a subtree immediately underneath the root node. Since all the currently considered recording devices record data that is related to the simulation time, each recording device comprises again several subtrees – one per recorded timestep.

Multimeters record time series data of certain attributes for individual neurons. Those attributes are, for instance, the membrane potential (V_m) or the voltage threshold (V_θ) for firing. Each attribute constitutes a separate subtree underneath the respective time step. For a multimeter, data of each attribute is recorded in every timestep. The data is stored per neuron in the tree's leaves.

Spike detectors record neurons' firing events, i.e., spikes. Typically, these data are sparse: spikes occur only in few timesteps and only few neurons fire in a timestep. Consequently, only those timesteps that contain spikes are stored in the tree. The firing neurons' ids are immediately stored as a list in the leaves of underneath the timestep.

We use Conduit for data handling. Conduit provides intuitive means to write and retrieve hierarchical data. The hierarchical layout of a block of raw memory

can be described to Conduit via JSON. An individual datum is addressed via its path. A valid path consists of the names of the tree's nodes, concatenated along the path from the root node to the respective datum. Furthermore, Conduit provides convenient means to serialize/deserialize the stored data for transport. Finally, Conduit efficiently facilitates updating existing data. The update is encoded as another Conduit tree and fed into the existing one. If a path exists, data are updated. If a path does not yet exists, the respective memory is allocated and data from the update is incorporated into the existing tree. This updating mechanism facilitates implementation of thread safe recording device adapters: each holds its own Conduit tree to record the data, consequently not requiring synchronization. Data is then transported off each device individually, again not requiring synchronization except for the transport layer. The analysis/visualization side in turn holds its own Conduit tree to collect the data to be visualized. It receives the individual trees and gathers them into its tree via Conduit's update mechanism.

On the simulation side, we provide base classes to facilitate implementation of simulator-specific adapters. These then tap into the recording methods of the simulator and feed the data into the hierarchical storage for transfer. On the analysis/visualization side, we provide base classes to facilitate implementation of reader classes that provide access to the stored data. Since each recording device in each simulator may provide different recordings, those reader classes again have to be simulator-specific.

We have implemented the described data layer, converting simulation data to and from the transfer format, as the C++ library `nesci` (pronounce ˈnɛsi) – Neuronal Simulator Conduit Interface. `Nesci` currently contains recording devices for NEST and a prototypic multimeter for Arbor. We provide Python bindings for the analysis/visualization side of `nesci` for convenient integration into existing scripts. `Nesci` is publicly released[11] under the terms and conditions of the Apache v2.0 license.

3.2 CONTRA – Conduit Transport

The data generated by the simulator and recorded via `nesci` has to be transported to the analysis/visualization side. Since `nesci` encoded the data using Conduit into a simulator-agnostic transfer format, we provide a general purpose transport layer for Conduit trees. We expect this transport layer to be therefore also useful for domains other than neuroscience.

The transport layer provides an abstraction from the specific transport technology, e.g., shared memory or network connections. Inspired by Conduit, we offer a relay that acts as the interface to the data producers and consumers, respectively. The former sends Conduit trees via the relay interface. The latter is provided by the relay with a list of Conduit trees that can then be aggregated into a single data set via Conduit's update mechanism (cf. Subsection 3.1).

[11] `https://devhub.vr.rwth-aachen.de/VR-Group/nesci`

A relay constructs an instance of an accessor to the selected transport technology. On sending, the Conduit tree is serialized into a data packet comprising the tree's data as raw bytes and its schema as a JSON string. The packet is then handed over to the selected transport technology via the accessor. On receiving, the arrived packets are deserialized into individual Conduit trees, one per packet. That way, the consuming application can implement application-specific data aggregation using Conduit's update mechanism.

We provide a relay implementation that can use a variety of transport technologies. For each technology an accessor needs to be implemented. This accessor has to be able to handle the specified data packets by providing suitable send and receive methods. Consequently, environment specific transport technologies can be added in a straightforward way.

We have implemented the described transport layer as the C++ library `contra` – CONduit TRAnsport. `Contra` currently executes data transport via shared memory provided by `boost::interprocess`[12]. That way, `contra` can be used for coupling simulations to analysis/visualization processes both running on a neuroscientist's local workstation. A ZeroMQ-based and a GPRC-based transport will be added in the near future enabling simulation and analysis/visualization to be run on separate machines. MPI-based transport will be added afterwards, possibly relying on Conduit's existing MPI support. We provide Python bindings for the transport interfaces for convenient integration into existing scripts. `Contra` is publicly released[13] under the terms and conditions of the Apache v2.0 license. The LGPL will apply for ZeroMQ-based transport.

4 Application

As a proof-of-concept application we couple a small NEST simulation to both a 2D and a 3D visualization. Both simulation and visualization are supposed to run on a single machine, so that neuroscientists can immediately inspect a running simulation at their desk. This, however, deliberately limits the simulatable network size. Nevertheless, simulations of larger networks to be run on a supercomputer are left for future work. Simulation and visualization are supposed to run in different processes, having simulation and analysis loosely coupled. This split is furthermore expected to simplify later deployment to other platforms using different means of transportation.

4.1 NEST Simulation

The simulated network consists of two layers of neurons containing $4n$ excitatory and n inhibitory neurons, respectively. The neurons are simulated using a leaky integrate-and-fire model with exponential shaped postsynaptic currents [6,10,18]. With a 10 % probability each neuron creates intra- and inter-layer connections to

[12] `https://www.boost.org`
[13] `https://devhub.vr.rwth-aachen.de/VR-Group/contra`

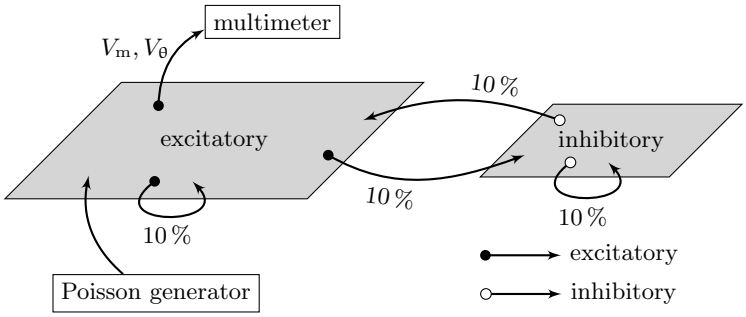

Fig. 2. Neural network used for the proof-of-concept application. A layer of $4n$ excitatory neurons and a layer of n inhibitory neurons are connected with both intra- and inter-layer connections. The excitatory neurons are fed a noise signal. They are connected to a multimeter for visualization.

other neurons. A Poisson generator is connected to each excitatory neuron, feeding them a noise signal, serving as input to the neuronal network. The structure of the network is visualized in Figure 2. A multimeter is connected to the excitatory neurons, recording their membrane potentials V_m and their firing threshold V_θ.

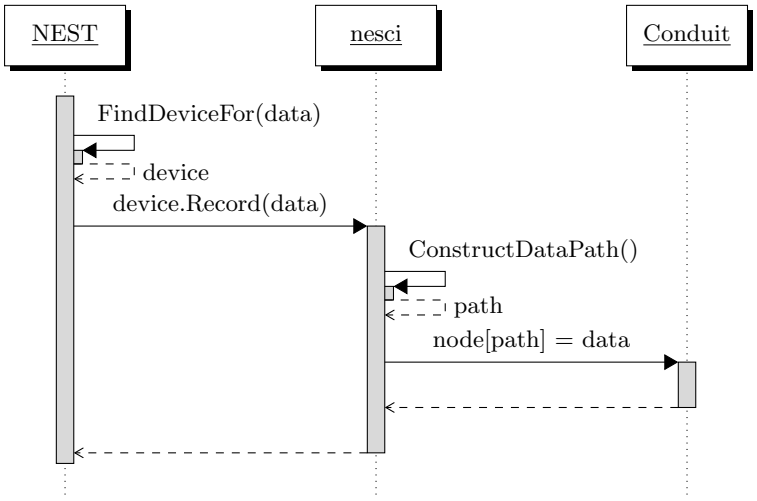

Fig. 3. Data flow for writing data from the NEST simulation to the Conduit node of a nesci device (e.g., a multimeter or spike detector).

In each timestep, NEST issues a call to its recording backend for each recording device and each connected neuron. The measurements for all the neuron's recorded attributes are passed into this call. This data is passed to `nesci` using

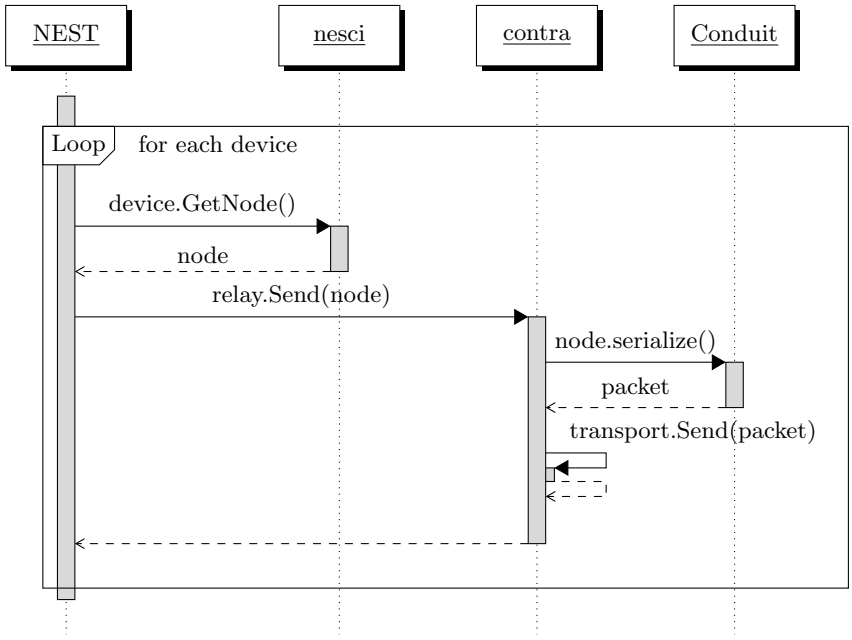

Fig. 4. Data flow for sending data stored in the Conduit nodes of each nesci device via contra.

a convenient interface where it is then stored at the corresponding path in a conduit tree (Figure 3). After some timesteps being recorded – here 10 – the recorded data is sent via `contra`'s transport layer to the analysis/visualization side (Figure 4).

4.2 2D Visualization

The 2D visualization is implemented in Python. It has a Conduit tree to aggregate all the data to be visualized. The visualization is run as a separate process and continuously polls `contra` for new Conduit trees (cf. top of the outer loop in Figure 5). Once new data has been received, each of the new subtrees is used as an update to the visualization's main Conduit tree as described in Subsection 3.1 (cf. inner loop in Figure 5). After having received new data, the 2D plot of the simulated neurons' membrane potentials is updated. For this purpose a list of available timesteps is queried from the multimeter. Afterwards, the neuron ids for which the requested attribute (here the membrane potential) has been recorded during the timesteps is determined. Then, the time-series data is queried for each neuron and passed to Matplotlib for plotting (cf. bottom of the outer loop in Figure 5).

Figure 6, left, shows the resulting plot after having simulated 1000 ms of biological time, using $n = 9$, i.e., 36 excitatory neurons. The plot is continuously

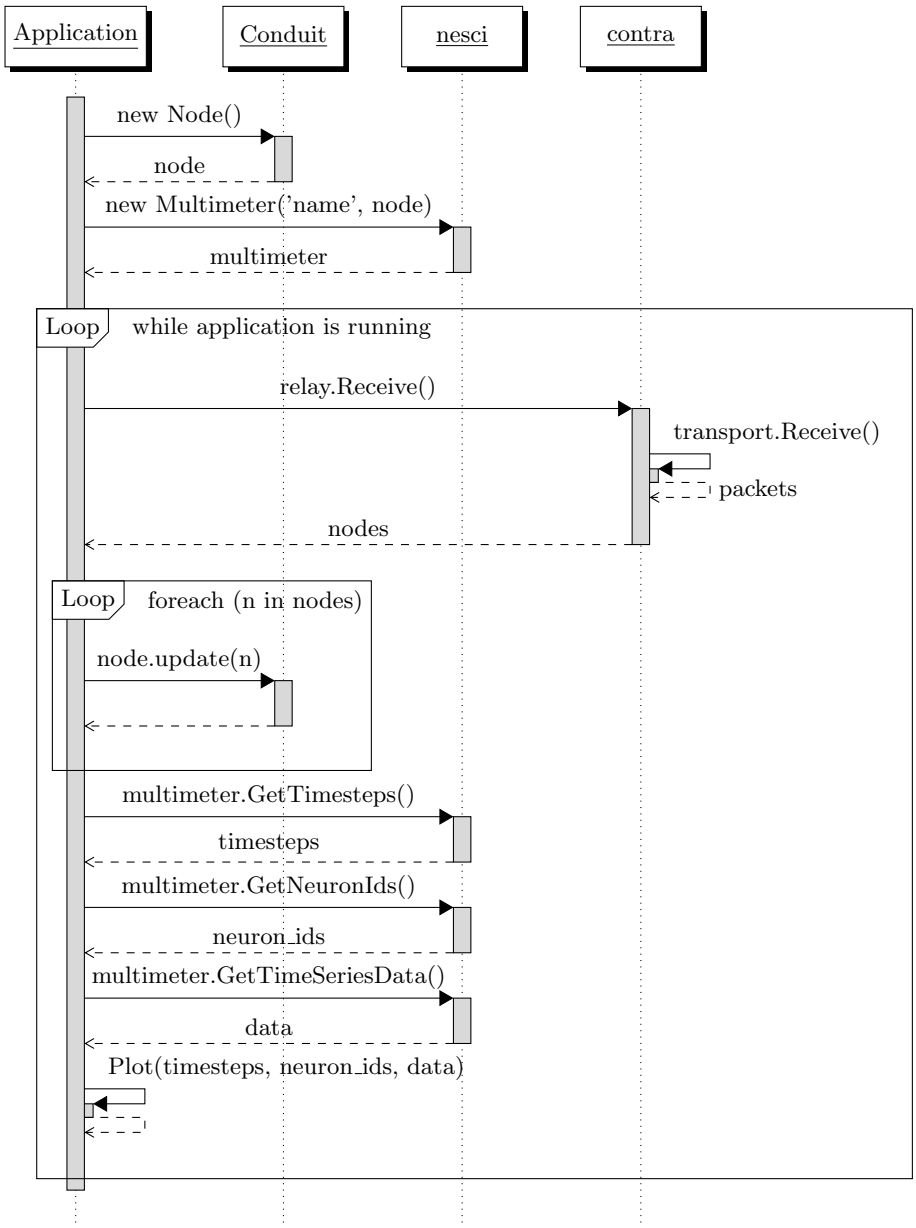

Fig. 5. Structure of the visualization applications. A single nesci multimeter device is created and used to query the data from the main conduit node for the visualization. The relay polls for nodes containing new data via the transport layer (the shared memory) and adds each subtree to the main conduit tree.

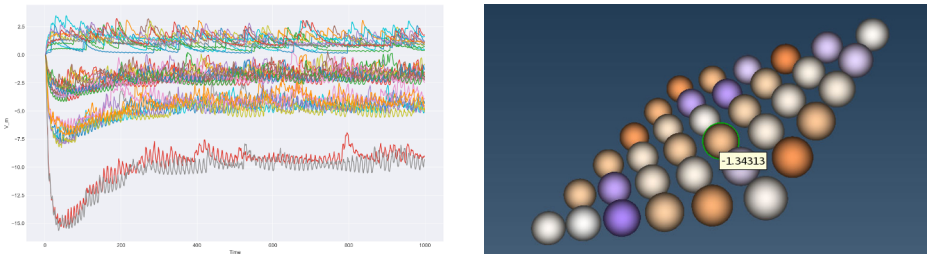

Fig. 6. Visualization of live simulation data generated for the example application using $n = 9$, i.e., 36 excitatory neurons. Left: time-series membrane-potential plot. Right: colour-coded membrane potential mapped to spatially arranged neurons; one neuron is selected with the tooltip showing its membrane potential.

updated during the simulation. The value of n is set only in the simulation script and not hard-coded anywhere in the `nesci-contra` pipeline. Instead, the visualization script automatically determines the number of neurons in the data set in order to plot the correct number of lines – one per neuron.

4.3 3D Visualization

Physically, the neurons in this example are arranged on regular grids. The simulation requires only the connections between the neurons which are modelled probabilistically. The neuroscientists, however, need to see the simulated data arranged in their physically correct way in order to interpret them fully.

For this purpose, we have created a prototypic 3D visualization with the colour-coded membrane potential mapped onto spatially arranged neurons. We have implemented this application in C++ using VTK [19]. Instead of presenting time-series data, this application presents all the membrane potentials for a given time step. The neuroscientists can then navigate the time steps to look at the respective simulation data for all neurons simultaneously in order to evaluate the network's spatial behaviour. Figure 6, right, shows the resulting visualization.

Polling the data from `contra` is done the same way as the previous python implementation (c.f. Figure 5). The retrieved data is then passed to a `vtkPolyData` in order to be rendered.

5 Performance Evaluation

An important question is how much overhead this adds to the simulation when using our proposed pipeline as the benefits of in situ visualization vanish if it slows down the simulation too much. The neuronal network described in the previous section was simulated on a machine equipped with two Intel® Xeon® CPU E5-2695 v3 @ 2.3 GHz for $n = 9$, $n = 25$, and $n = 81$. We connected a simple python client similar to Figure 5 to poll – and thus clear – the shared memory to avoid running to create a realistic scenario. However, the display

of the plots was removed from the application to reduce the impact on the simulation as both run on the same machine. We do not focus on the performance of the visualization but rather the overhead added to the simulation by adding our custom recording backend to the NEST simulator.

	Baseline	ASCII	NESCI & CONTRA
$n = 9$	21.5 ms	98.2 ms (457 %)	543.7 ms (2528 %)
$n = 25$	62.2 ms	281.1 ms (452 %)	1559.0 ms (2506 %)
$n = 81$	206.7 ms	940.5 ms (455 %)	5293.2 ms (2560 %)

Table 1. The total time to run the simulation without any recording backend attached, with the ASCII recording backend attached and with our streaming recording backend attached. The percentages in brackets denote the relative duration of the simulation with the corresponding recording backend attached compared to the baseline.

First, we ran the simulation without any recording backend attached to get a baseline of how long the simulation takes on this specific machine. Second, we ran the same simulation attached with the ASCII recording backend that is provided alongside the NEST simulator. It is used to compare the overhead of our streaming backend to other recording backends. Last, we ran the same simulation attached with our custom streaming recording backend that uses nesci and contra. Table 1 shows the total simulation time for the three presented scenarios. For each scenario the network was simulated for 1000 ms of biological time. The simulation time denotes how long it took the simulation to finish. Each test was repeated 5 times only counting the minimum result.

Currently, our custom recording backend adds to much overhead to the simulation for practical usage in real world applications. However, it should be noted that the current implementation is only a proof-of-concept and not a finished product because at this point in time no effort has been put into optimizing the performance of the pipeline. To determine where the performance can be improved the simulation was profiled using the Valgrind instrumentation framework [15].

Profiling revealed two major bottlenecks in the recording backend: writing new data into the Conduit tree as described in Figure 3 and writing the data into the shared memory as described in Figure 4. These two functions contribute to over 90 % of the total simulation time (46 % and 45 % respectively). In the remaining part of this section, we focus on a more detailed analysis of these two functions and suggests possible optimizations.

The main operations that contribute to the bad performance of writing data to the Conduit tree (c.f. Figure 3) are constructing the path to the data (16 % of the total simulation time) and inserting the data into the Conduit tree (22 % of the total simulation time). Constructing the path currently involves numerous

costly `std::stringstream` operations to convert floating point values to strings. The 2017 standard of C++ provides the alternative function `std::to_chars` which is optimized for performance instead of flexibility and usability. The second operation – inserting the data into the tree – involves numerous costly string operations to extract the path and dynamic allocations for the newly added nodes. Conduit supports preallocating a block of memory that can be used to store the data directly. On the one hand, this would drastically reduce the number of allocations when adding new nodes to the tree. On the other hand, the data cen be now written directly to the raw memory by calculating the offset. This saves the costly string operations introduced by parsing the path to the data.

The second major bottleneck is writing the conduit tree to the shared memory (c.f. Figure 4). Like in the previous scenario, two operations take the majority of the time: serializing the Conduit tree (22 % of the total simulation time) and actually writing the data to the shared memory (20 % of the total simulation time). The serialization of the tree is done by Conduit itself so there is no direct way to optimize this step. However, preallocating blocks of memory as discussed before should also accelerate the serialization process as Conduit is able to just copy the block instead of traversing every single node of the tree. In the second limiting operation – writing to the shared memory – almost the whole time is spent acquiring the mutex that synchronizes the writing and the reading process. This could be done in a separate thread so it does not block the simulation.

6 Conclusion and Future Work

We have proposed a lightweight streaming framework for connecting neuronal simulators to analysis/visualization. We have demonstrated its applicability by a proof-of-concept application coupling a simple NEST simulation to 2D and 3D visualizations. Both present the data while it is being simulated without having to continuously pause and resume the simulation. The current state of the pipeline adds too much overhead, but we provided a lot of suggestions to improve its performance in future work.

The two developed libraries (`nesci` and `contra`) work hand in hand, providing concise APIs for integration into neuroscientific workflows. We have demonstrated this property by outlining a typical data flow. Both libraries are publicly available under the terms and conditions of the Apache v2.0 license. While `nesci` is specifically developed for neuroscientific applications, we expect `contra` to be useful in other domains that are using Conduit for describing their data.

We plan to add ZeroMQ-, GRPC-, and MPI-based transport mechanisms to `contra` in the near future. That way, the proposed pipeline will become applicable to large-scale neuronal simulations. Furthermore, we plan to extend `nesci`'s Arbor support. We currently have only a rudimentary implementation available. Working in close collaboration with the Arbor developers will help streamline this implementation and greatly improve performance. In the long run, we will add support for other widely used neuronal simulators.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nos 720270 (HBP SGA1) and 785907 (HBP SGA2), and from the Excellence Initiative of the German federal and state governments.

References

1. Ayachit, U.: The ParaView Guide: A Parallel Visualization Application. Kitware, Inc. (2015)
2. Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., Mauldin, J.: ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In: Proc. 1st Workshop In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization. pp. 25–29 (2015). https://doi.org/10.1145/2828612.2828624
3. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: The SENSEI Generic in Situ Interface. In: Proc. 2nd Workshop In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization. pp. 40–44 (2016). https://doi.org/10.1109/ISAV.2016.13
4. Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In: High Performance Visualization–Enabling Extreme-Scale Scientific Insight, pp. 357–372 (2012)
5. Cumming, B., Yates, S., Klijn, W., Peyser, A., Karakasis, V., Perez, I.M.: Arbor: A morphologically detailed neural network simulator for modern high performance computer architectures. Proc. Neuroscience 2017 (2017), `http://juser.fz-juelich.de/record/840405`
6. Diesmann, M., Gewaltig, M.O., Rotter, S., Aertsen, A.: State space analysis of synchronous spiking in cortical neural networks. Neurocomputing **38-40**, 565–571 (2001). https://doi.org/https://doi.org/10.1016/S0925-2312(01)00409-X
7. Djurfeldt, M., Hjorth, J., Eppler, J.M., Dudani, N., Helias, M., Potjans, T.C., Bhalla, U.S., Diesmann, M., Hellgren Kotaleski, J., Ekeberg, Ö.: Run-Time Interoperability Between Neuronal Network Simulators Based on the MUSIC Framework. Neuroinformatics **8**(1), 43–60 (2010). https://doi.org/10.1007/s12021-010-9064-z
8. Gewaltig, M.O., Diesmann, M.: NEST (NEural Simulation Tool). Scholarpedia **2**(4), 1430 (2007)
9. Hines, M.L., Carnevale, N.T.: Neuron: A Tool for Neuroscientists. Neuroscientist **7**(2), 123–135 (2001). https://doi.org/10.1177/107385840100700207
10. Kobayashi, R., Tsubo, Y., Shinomoto, S.: Made-to-order spiking neuron model equipped with a multi-timescale adaptive threshold. Front Comput Neurosci **3**, 9 (2009). https://doi.org/10.3389/neuro.10.009.2009
11. Larsen, M., Ahrens, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In: Proc. 3rd Workshop In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization. pp. 42–46 (2017). https://doi.org/10.1145/3144769.3144778

12. Lytton, W.W., Seidenstein, A.H., Dura-Bernal, S., McDougal, R.A., Schürmann, F., Hines, M.L.: Simulation Neurotechnologies for Advancing Brain Research: Parallelizing Large Networks in NEURON. Neural Comput **28**(10), 2063–2090 (2016). https://doi.org/10.1162/NECO_a_00876
13. Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., Knoll, A., Sompolinsky, H., Verstreken, K., DeFelipe, J., Grant, S., Changeux, J.P., Saria, A.: Introducing the human brain project. Procedia Computer Science **7**, 39 – 42 (2011). https://doi.org/10.1016/j.procs.2011.12.015
14. Morrison, A., Straube, S., Plesser, H.E., Diesmann, M.: Exact Subthreshold Integration with Continuous Spike Times in Discrete-Time Neural Network Simulations. Neural Comput **19**(1), 47–79 (2007). https://doi.org/10.1162/neco.2007.19.1.47
15. Nethercote, N., Seward, J.: Valgrind: A framework for heavyweight dynamic binary instrumentation. SIGPLAN Not. **42**(6), 89–100 (Jun 2007). https://doi.org/10.1145/1273442.1250746, http://doi.acm.org/10.1145/1273442.1250746
16. Nowke, C., Diaz Pier, S., Weyers, B., Hentschel, B., Morrison, A., Kuhlen, T.W., Peyser, A.: Toward rigorous parameterization of underconstrained neural network models through interactive visualization and steering of connectivity generation. Front Neuroinform (2018), provisionally accepted
17. Nowke, C., Zielasko, D., Weyers, B., Peyser, A., Hentschel, B., Kuhlen, T.W.: Integrating Visualizations into Modeling NEST Simulations. Front Neuroinform **9**, 29 (2015). https://doi.org/10.3389/fninf.2015.00029
18. Rotter, S., Diesmann, M.: Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biol Cybern **81**(5), 381–402 (1999). https://doi.org/10.1007/s004220050570
19. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics. Kitware, Inc., 4th edn. (2006)
20. Schuecker, J., Schmidt, M., van Albada, S.J., Diesmann, M., Helias, M.: Fundamental activity constraints lead to specific interpretations of the connectome. PLoS Comput Biol **13**(2), 1–25 (2017). https://doi.org/10.1371/journal.pcbi.1005179
21. Schumann, T., Frings, W., Peyser, A., Schenck, W., Thust, K., Eppler, J.M.: Modeling the I/O behavior of the NEST simulator using a proxy. In: Proc. 3rd ECCOMAS Young Investigators Conf. (2015), http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:82-rwth-2015-039806
22. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In: EG Symp. Parallel Graphics and Visualization (2011). https://doi.org/10.2312/EGPGV/EGPGV11/101-109