NEST-5g performance assessment report

Document Information

Reference Number	POP_AR_111 (NEST-5g)
Author	Brian Wylie (JSC)
Contributor(s)	
Date	April 9, 2018

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.*

Contents

1 Background

Applicants Name: Jari Pronold
Institution: Institute of Neuroscience and Medicine (INM-6), Forschungszentrum Jülich
Contributors: Itaru Kitayama & Jun Igarashi (RIKEN), Markus Diesmann, Jakob Jordon, Suzanne Kunkel, Moritz Helias (FZJ)
Application Name: NEST-5g (development version of 2017/05/31 tagged '2aa733a')
Programming Language: C++
Programming Model: MPI+OpenMP (MPI_THREAD_SERIALIZED)
Source Code Available: yes (GPL license: `www.nest-simulator.org`)
Performance study: check (audit)
User description: investigation of simulation scalability on K computer
Application Description: NEST is a simulator for large-scale neuronal networks of simple neuron models with biophysically plausible density of connections.
Input data: `hpc_benchmark.sli`
Testcase Description: 1 MPI process with 8 OpenMP threads per compute node.
Machine Description: K computer at RIKEN AICS, comprising 82,944 compute nodes connected by proprietary Tofu interconnect in a 6D mesh/torus, and with Lustre-based Fujitsu Exabyte File System. Each node has a single Fujitsu SPARC64 VIIIfx 8-core 2.0 GHz processor and runs a Linux-based OS. Fujitsu compilers and MPI (based on OpenMPI) are provided.
Analysis tools: Scalasca/2.3.1 using Score-P/3.1 (environment K-1.2.0-22)

2 Setup

The specified `hpc_benchmark.sli` testcase for weak-scaling executions of typical petascale neuronal network simulation has a balanced random network of 12,500 neurons per MPI process with 11,500 (9000 excitatory plus 2250 inhibitory) incoming synapses per neuron, and with network dynamics simulated for 500ms of biological real time (`Tsim500`).

An instrumented version[1] of NEST-5g for measurement was built on K computer using Score-P with manual user annotation of the key application execution phases combined with OpenMP source construct pre-processing and MPI library interposition. Measurement executions were done on K computer with different numbers of compute nodes, with a single MPI process and eight OpenMP threads per compute node.

3 Application structure and Focus of Analysis

The execution timeline of NEST-5g in Figure 1[2] clearly distinguishes the three primary phases: the initial *connect* phase (aka "build time") to construct all nodes and the postsynaptic part of the connection infrastructure, followed by simulation *prepare* phase (aka "presim time," constructing the presynaptic part of the connection infrastructure and a short pre-simulation where initial transients of the network dynamics subside, dominated by the OpenMP parallel region starting at simulation_manager.cpp line 457) and simulation *run* phase (aka "sim time," dominated by the OpenMP parallel region starting at simulation_manager.cpp line 772), and

[1] pre-release development dated May 2017 tagged '2aa733a'
[2] Vampir display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/vampir_display_quickref.pdf

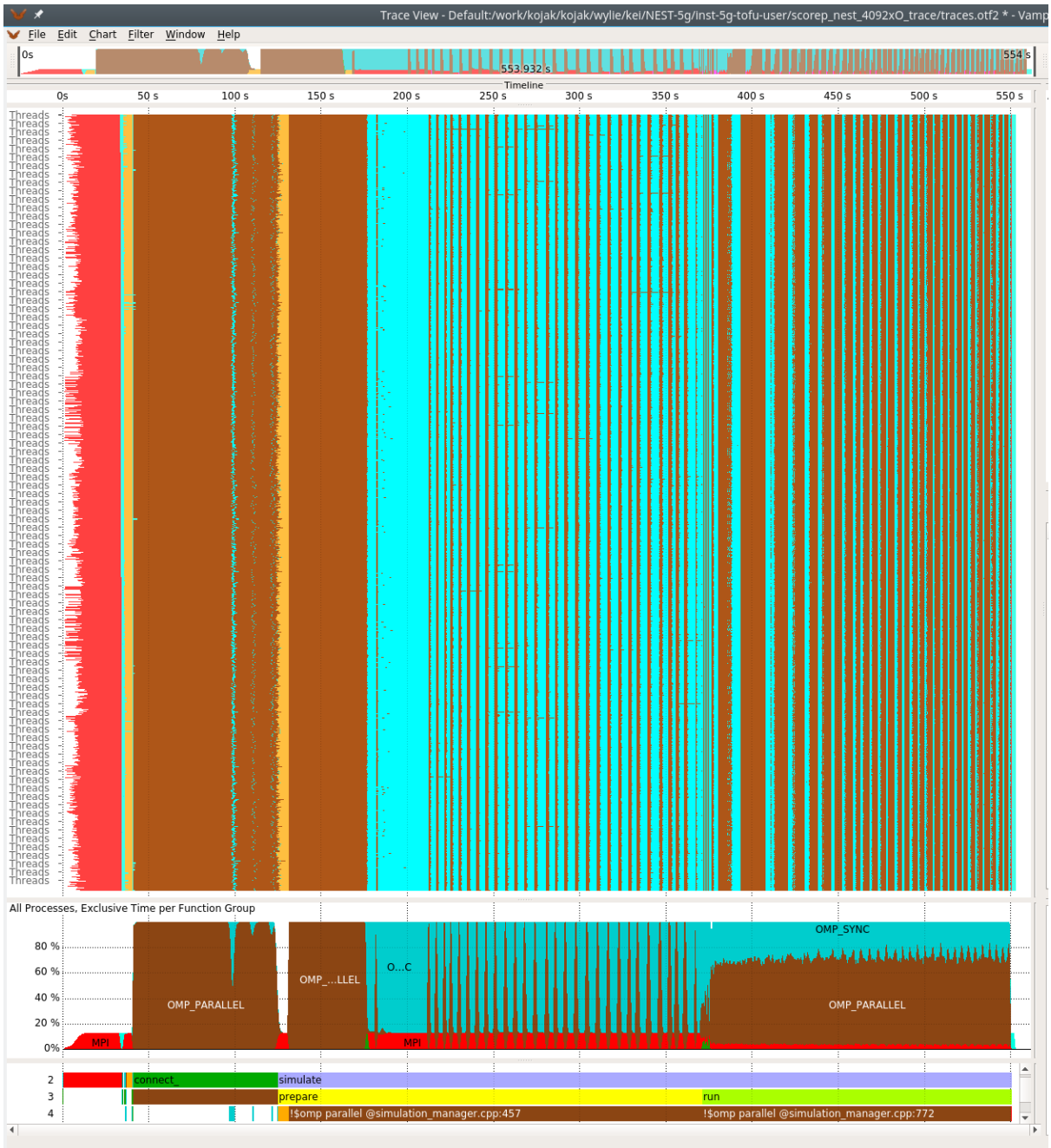

Figure 1: Execution timeline of NEST-5g Tsim500 execution using 4092 MPI processes each with 8 OpenMP threads on K computer.
Thread activity chart of the master thread of MPI rank 0 at bottom showing *connect* phase (in dark green) between 40 and 125 seconds, followed by simulation *prepare* (yellow) phase until 370 seconds and *run* (light green) to completion at 550 seconds.
Above it are the function time summary chart and timeline state chart of all 32,736 threads at top, where OpenMP parallel computation is brown, OpenMP synchronization is cyan, MPI collective communication is orange and other MPI (including `Init` and `Barrier`) is red.
There are 31 MPI `Alltoall` communications in the *prepare* phase, and 355 in the *run* phase.

a final short *cleanup*. Although here the *prepare* phase is longest, consisting almost entirely of MPI and OpenMP synchronization, the *run* phase is the more interesting focus of analysis (FOA) as it would be much longer in real simulations.

Figure 2 shows a NEST-5g execution call-tree extract from a profile measurement on K computer.[3] The manually-annotated *simulate* phase is split into *prepare* (shown collapsed) and the expanded *run* phase with the OpenMP parallel region of simulation_manager.cpp starting at line 772 containing a variety of OpenMP synchronization constructs (of type atomic, critical, single, and both explicit and implicit barriers). MPI collective communication, such as `MPI_Alltoall` occurs in a structured block (sblock) from line 470 to 485 of event_delivery_manager.cpp, requiring `MPI_THREAD_SERIALIZED` thread support to allow it to be executed by the first available thread on each MPI rank. (`MPI_Alltoall` is preceded by explicit synchronization with `MPI_Barrier`.)

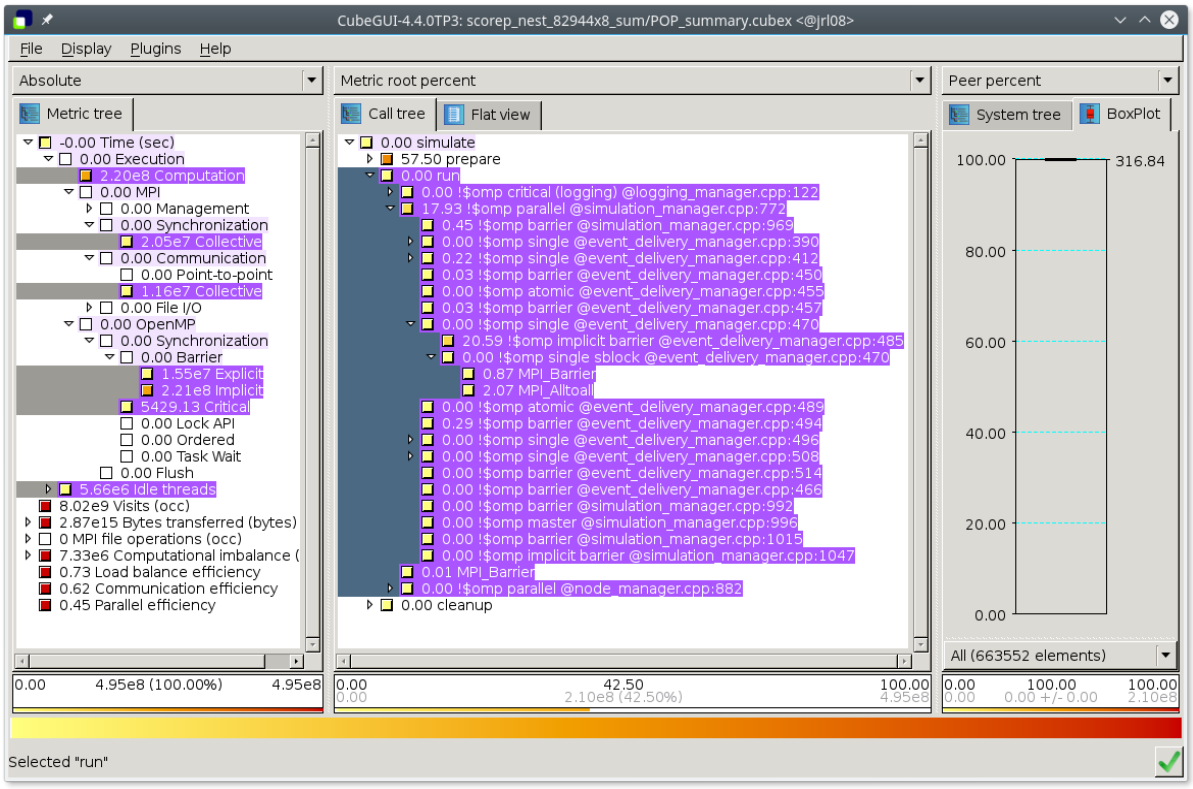

Figure 2: Scalasca analysis report explorer presentation of NEST-5g Tsim500 profile extract from execution on 82,944 compute nodes of K computer. Execution call-tree in the centre pane with simulation *run* phase expanded and selected (highlit), annotated with percentage of total CPU time accumulating to 317 seconds on each of the 663,552 threads.

4 Scalability

Figure 3 shows the weak scaling performance of NEST-5g from 128 to all 82,944 compute nodes of K computer. *connect* (build_edge) time is roughly constant 80 seconds, whereas both *prepare* (presim) and *run* (sim) take exponentially longer with increasing numbers of compute nodes.

[3] Cube display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/cube_display_quickref.pdf

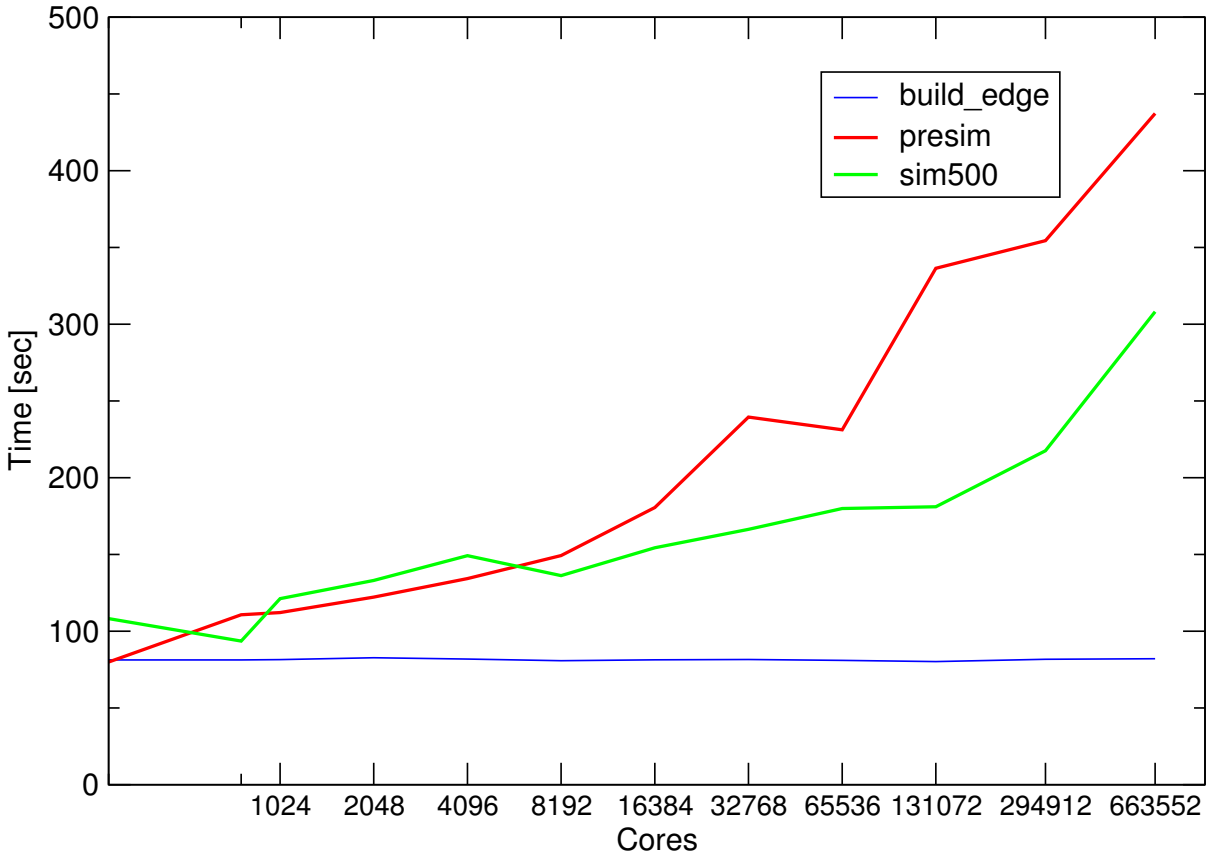

Figure 3: Wallclock execution time of NEST-5g weak-scaling testcase on K computer, distinguishing initial build_edge (*connect*), preparatory presim (*prepare*) and actual simulation (*run*).

5 Parallel efficiency metrics

Basic parallel efficiency metrics[4] are defined where the higher the value (closer to 1.00) then the better is the efficiency. *Load balance* is the ratio of average computation to maximal computation time. *Communication* efficiency is the ratio of maximal computation to maximal executing time. *Parallel efficiency* is the ratio of the average computation time to the maximal executing time which is also the product of *Load balance* and *Communication*. *Computation* efficiency reflects how total computation time scales, here using the 36-node execution as reference. *Global* efficiency is then the product of *Computation* and *Parallel* efficiencies.

Table 1: Efficiency metrics for NEST-5g testcase

Processes/nodes	128	256	512	1024	2048	4096	8192	16384	36864	82944
Cores/threads	1024	2048	4096	8192	16384	32768	65536	131072	294912	663552
Global efficiency	0.68	0.62	0.55	0.60	0.53	0.49	0.45	0.45	0.38	0.27
- Parallel efficiency	0.72	0.70	0.67	0.78	0.73	0.70	0.66	0.67	0.57	0.42
- - Communication eff.	0.99	0.98	0.98	0.90	0.90	0.78	0.92	0.73	0.68	0.45
- - Load balance eff.	0.73	0.71	0.68	0.87	0.81	0.90	0.72	0.92	0.85	0.94
- Computation eff.	0.95	0.89	0.82	0.77	0.73	0.70	0.69	0.67	0.66	0.63

[4]POP standard metrics for parallel performance analysis: https://pop-coe.eu/node/69

Table 1 and Figure 4 give an overview of the scaling efficiency of the NEST-5g Tsim500 weak-scaling testcase executions. There is a general degradation of global efficiency to 0.27 as scale increases, with a gradual reduction of computation efficiency to 0.63 and more erratic reduction of parallel efficiency to 0.45, where the latter has contributory factors of communication and load balance efficiency which seem to vary significantly.

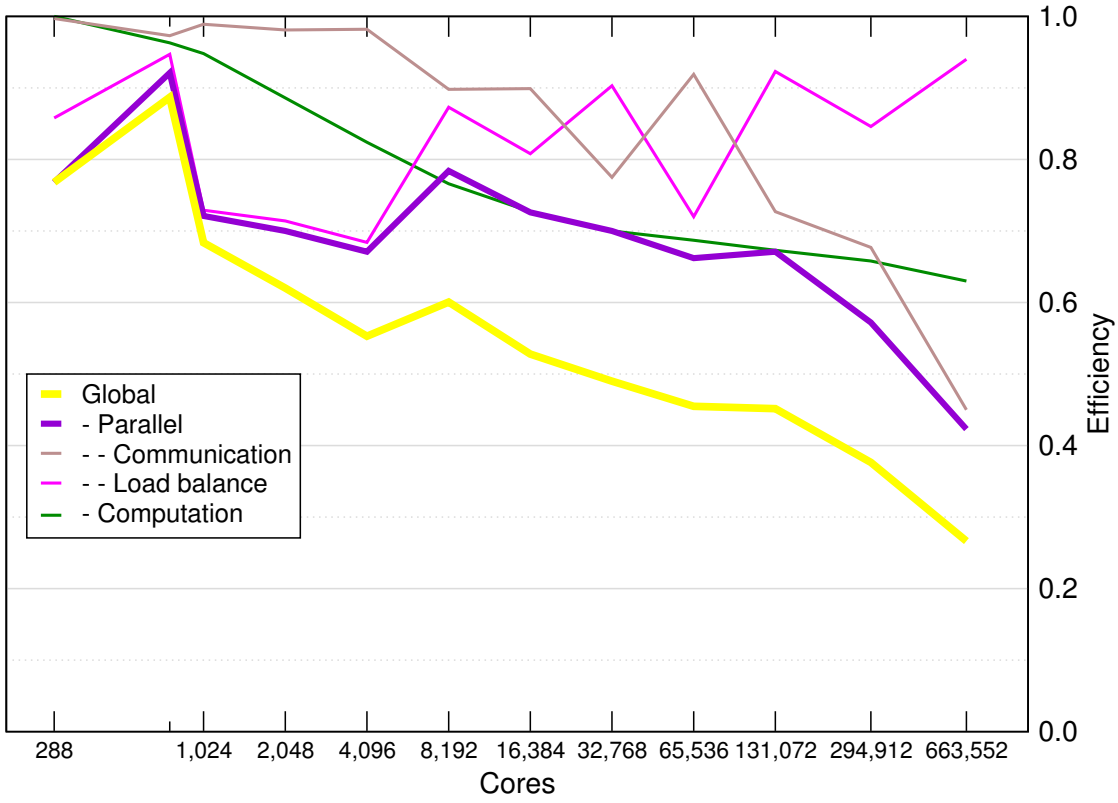

Figure 4: Efficiencies of simulation *run* phase.

6 Load balance

Load balance efficiency is highly variable between 0.68 and 0.94, with remarkably good efficiencies for some of the largest configurations. Despite the very good 0.94 load balance efficiency for 663,552 cores, closer examination in Figure 6 shows thread computation time ranging from 131 to 142 seconds and process computation time ranging from 1051 to 1100 seconds.

Load balance in the *run* phase of the NEST simulation test case relies on the randomly connected network of synapses, and fluctuations as the simulation progresses in the number of spike events generated by each neuron for other neurons in the network.

7 Serial computation

Computation efficiency degrades progressively and only remains above 0.80 until 4,096 cores, after which the rate of degradation diminishes continuing down to 0.63 for 663,552 cores. Mean computation time is also seen to progressively increase in the time breakdown in Figure 5,

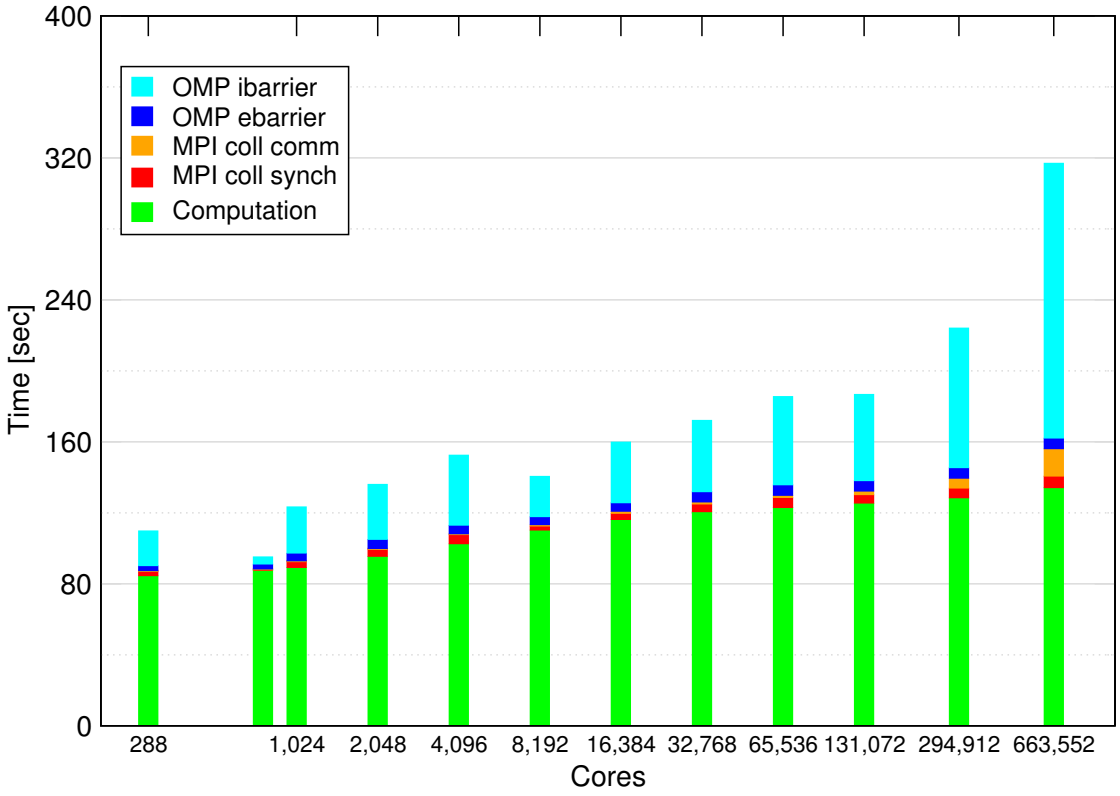

Figure 5: Breakdown of CPU time in simulation *run* phase.

indicating substantially growing computation for each thread as the total number of threads increases. (For perfect weak scaling, local computation is expected to remain constant.)

Figure 7 shows a time breakdown of the simulate sub-phases for process rank 0 provided by the application itself, where a large growth in `GatherSpikeData::collocate` dominates and lesser growth in `GatherSpikeData::deliver` are both evident. These sub-phases relate to packing and unpacking the spike event data buffers (before and after exchange between processes in `GatherSpikeData::communicate`). In comparison, the actual neuron computation `Update` increases only slowly and remains relatively small.

Further measurements employing hardware performance counters (such as those for cache misses) would be necessary to investigate potential reasons for this inefficiency.

8 Communication & Synchronization

Figure 7 shows the dramatic increase in time for `GatherSpikeData::communicate` where MPI communication is done by a `single` OpenMP thread in each round of communication exchange.

The Figure 5 breakdown of time in NEST-5g simulation *run* phase shows MPI and OpenMP synchronization times also grow with scale, and are particularly marked at the largest scales: OpenMP synchronization time is 7–9 times the MPI time, as might be expected from 7 threads waiting in OpenMP synchronization while one thread performs MPI operations. *Implicit OpenMP barrier synchronization* after the structured block in the OpenMP single construct used for `MPI_Alltoall` exchange of electrical impulses ("spike data events") is the dominant parallelization overhead, which is partially due to waiting time synchronizing the MPI processes arising from preceding computational imbalance.

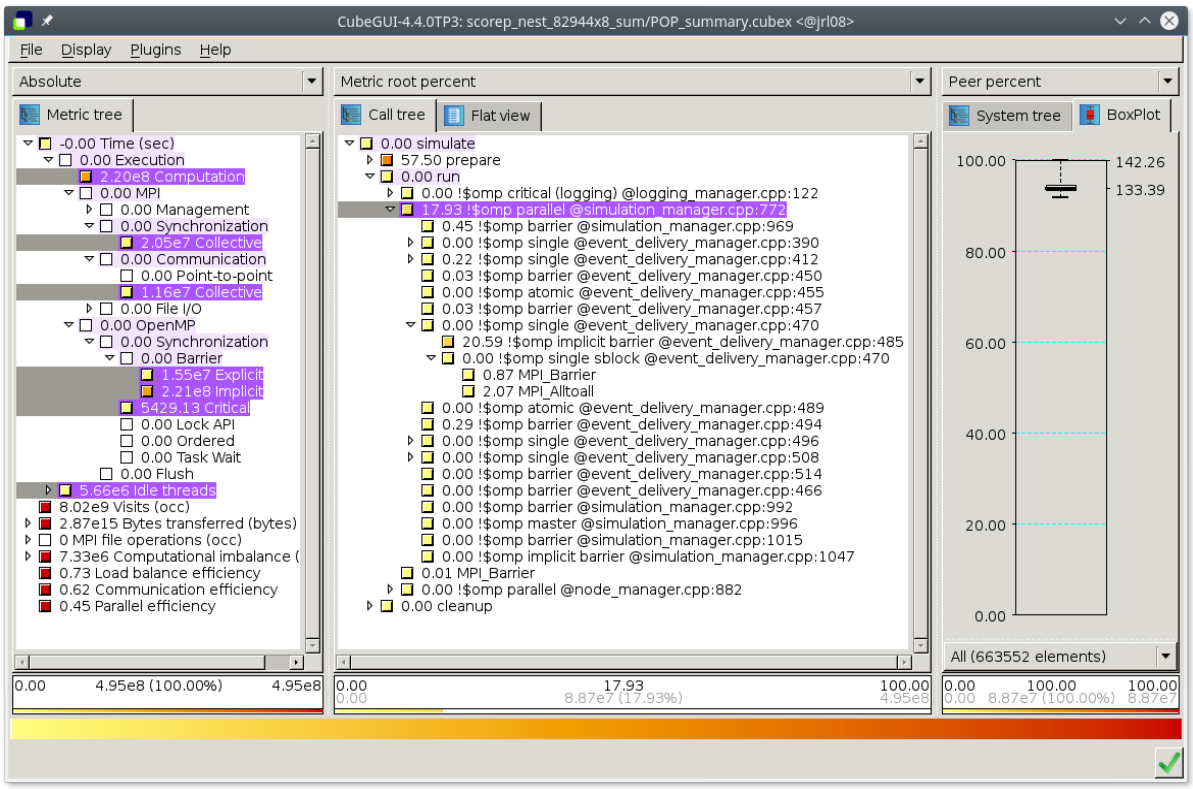

Figure 6: Scalasca analysis report explorer presentation of NEST-5g profile extract from execution on 82,944 compute nodes of K computer showing computation time imbalance in the simulation *run* phase primary parallel loop (simulation_manager.cpp lines 772 to 1047) ranging from 131 to 142 seconds on the 663,552 threads.

Figure 8 shows the number of MPI `Alltoall` instances (visits), bytes transfered (where incoming=outgoing) and associated communication time for the simulation *run* and *prepare* phases, for each process. For reference, the spike data buffer size (`buffer_size_spike_data`) is also shown.

The *prepare* phase with 6 communication steps for spike data has a relatively small and slowly increasing number of MPI `Alltoall` instances, with proportional bytes transfered and time. In contrast, for the *run* phase with 340 communication steps for spike data exchange, there is a marked increase from roughly 360 MPI `Alltoall` instances to almost double that for the two largest configurations, with a corresponding increase in time. Notably, the amount of data transfered in and out of each MPI process also increases, but much more erratically, being the product of `buffer_size_spike_data` and the number of MPI `Alltoall` instances.

The doubling of the number of `Alltoall` calls used to deliver spike events for large configurations is due to additional communication rounds necessitated by the limited number of spike events that can be stored in the exchange buffer. While the number of local spike events is roughly constant for each execution configuration, the capacity reserved for each process in the fixed-size buffer progressively diminishes. While a variable-sized `Alltoallv` could be substituted to allow a constant-size spike exchange buffer, it additionally requires vectors of displacements (which also need to be determined), such that it may be substantially slower.

Ideally, the size of the spike events buffer should be sufficiently large for a single communication round, as this will be most efficient. If additional memory can't be allocated for the spike events exchange buffer, diminished simulation efficiency due to additional communication and

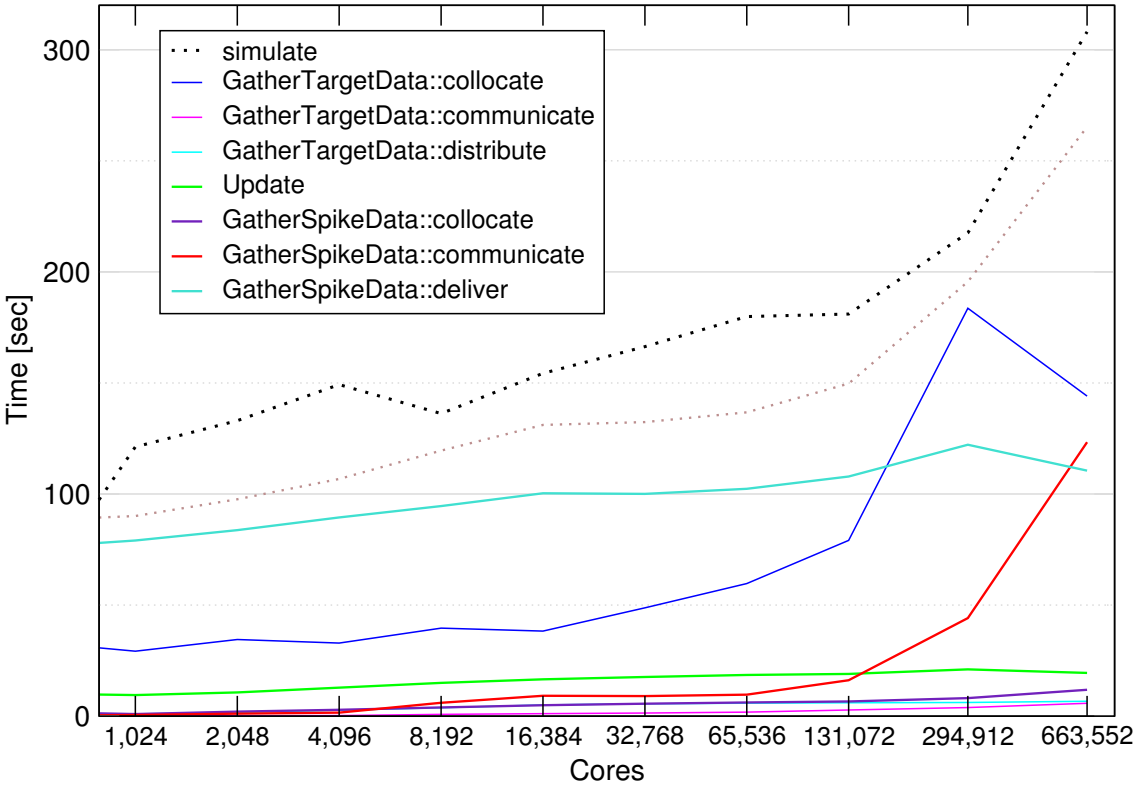

Figure 7: Simulation *run* phase time breakdown for process rank 0.

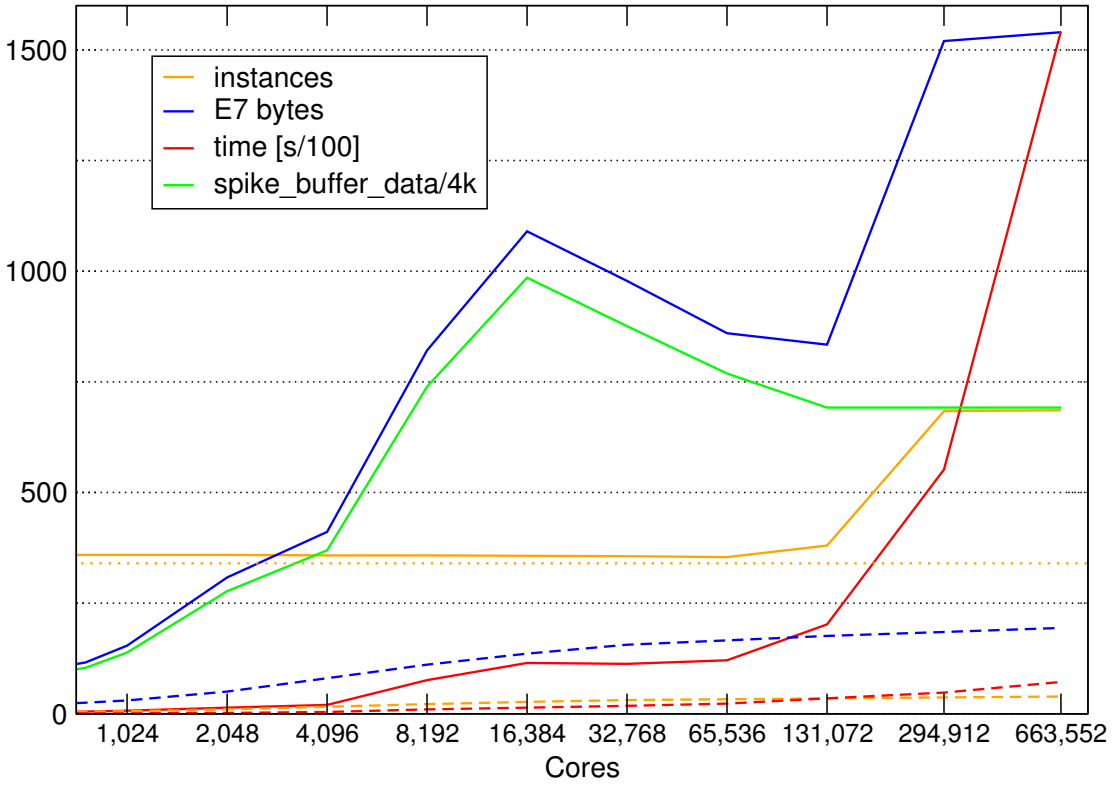

Figure 8: Usage and impact of MPI `Alltoall` in *run* phase (solid) and *prepare* phase (dashed).

longer simulation times will be the consequence. In such cases, it may be preferable to reduce the size of the simulation (i.e., the number of neurons and synapses) to ensure that sufficient spike buffer capacity is provided for single round exchange communication.

9 Summary of observations

Scalability and parallel efficiency of NEST-5g on K computer was investigated with a weak-scaling test configuration up to the full 82,944 compute nodes (663,552 OpenMP threads).

- Weak scaling performance of the simulation *run* phase deteriorates progressively, with a global efficiency of 0.6 to 8,192 cores and below 0.4 for more than 131,072 cores. Both computation and parallel inefficiencies contribute substantially.
- Computation efficiency drops below 0.8 for 8,192 cores, then stabilises somewhat above 0.6 to the largest execution configuration.
- Parallel efficiency of around 0.7 up to 131,072 cores indicates room for improvement of both communication and load balance, which appear to alternate in relative importance at different scales.
- Computation load imbalance leads to significant waiting time for MPI process synchronization in the OpenMP single region where `MPI_Alltoall` is used for event exchange.
- The largest execution configurations require additional rounds of MPI `Alltoall` global communication, resulting in diminished simulation performance.

Recommendations

- Investigate whether a larger spike exchange buffer can be used to avoid additional communication rounds.
- Investigate the origin of the computation load imbalance, perhaps related to the K computer topology, and generally try to improve it.
- Profile the simulation *run* phase in more depth by manually instrumenting the sub-phases for *update* of neurons and population of spike register for target lists, *collocation* of MPI send buffer, and *delivery* of spikes read from MPI buffers via synapses to targets.
- Investigate computation efficiencies of individual cores with hardware counter measurements, e.g., cache performance, vector instructions, branch instructions, etc.
- Compare performance and scalability with other HPC systems, such as IBM Blue Gene/Q.

One or more of these recommendations could be pursued via follow-up POP services, such as a Performance Plan or Proof-of-Concept prototyping.