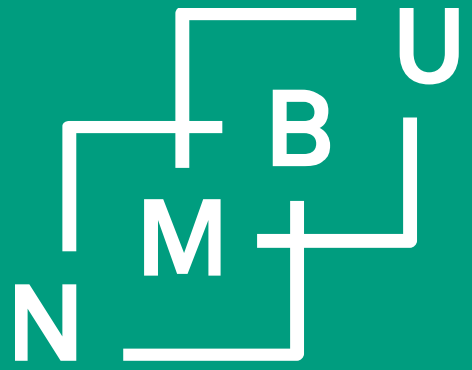


Norwegian University
of Life Sciences

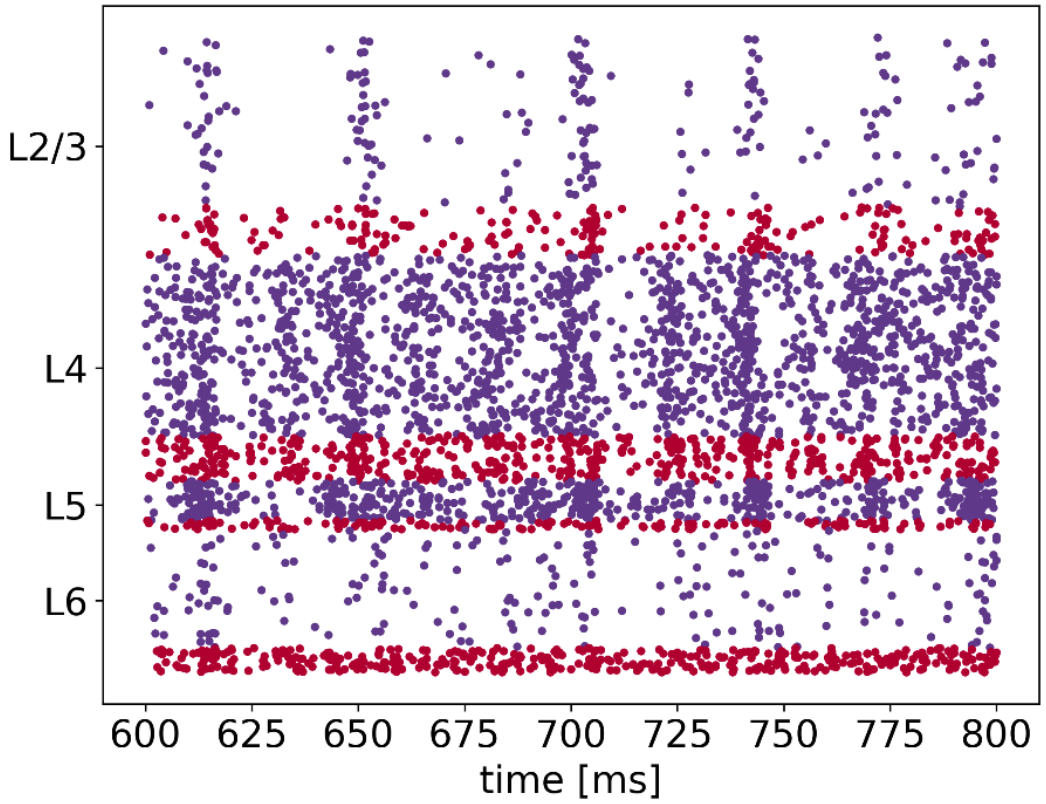
Insights: Successful infrastructure usage

Benchmarking and optimizing the performance of NEST at scale using Piz Daint

3 November 2020

The NEST simulator

- Simulator for spiking neural networks
- Focuses on the dynamics, size and structure of neural systems
  - Not as much on the exact morphology of individual neurons
- Has a Python api and a core written in C++



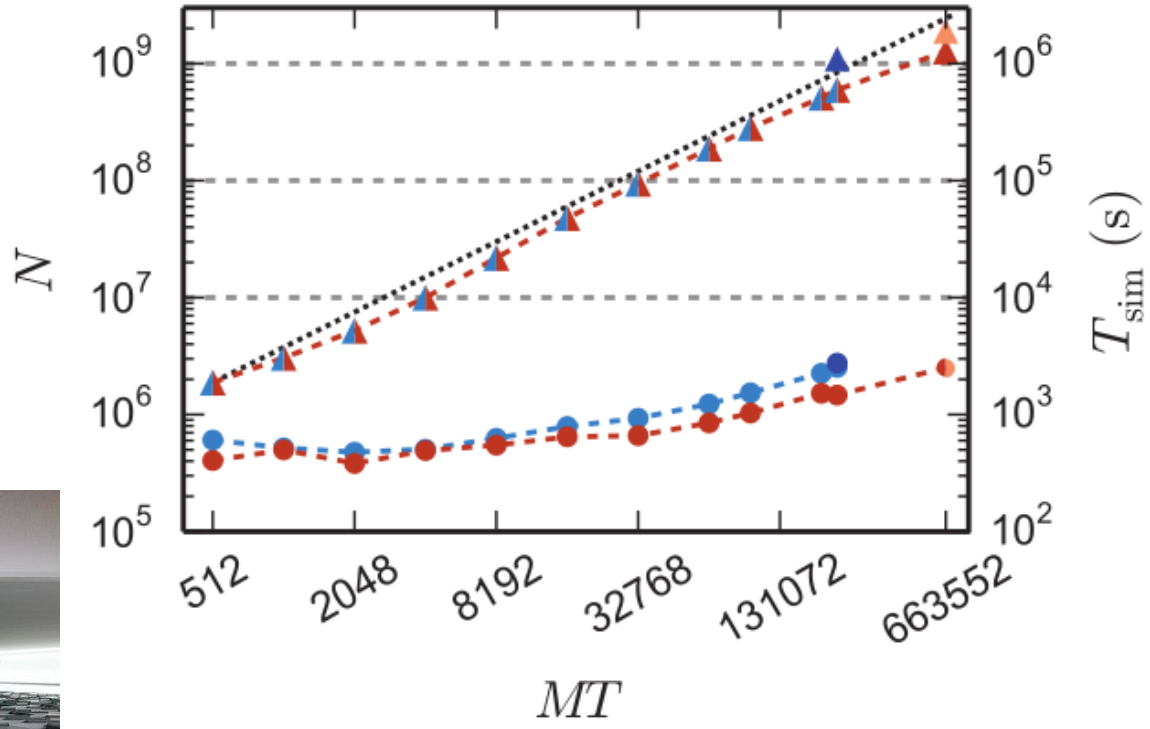
The NEST simulator

- NEST is fast and memory efficient
  - World record simulation (2014):

    1.86×10^9 neurons, 11.1×10^{12} synapses

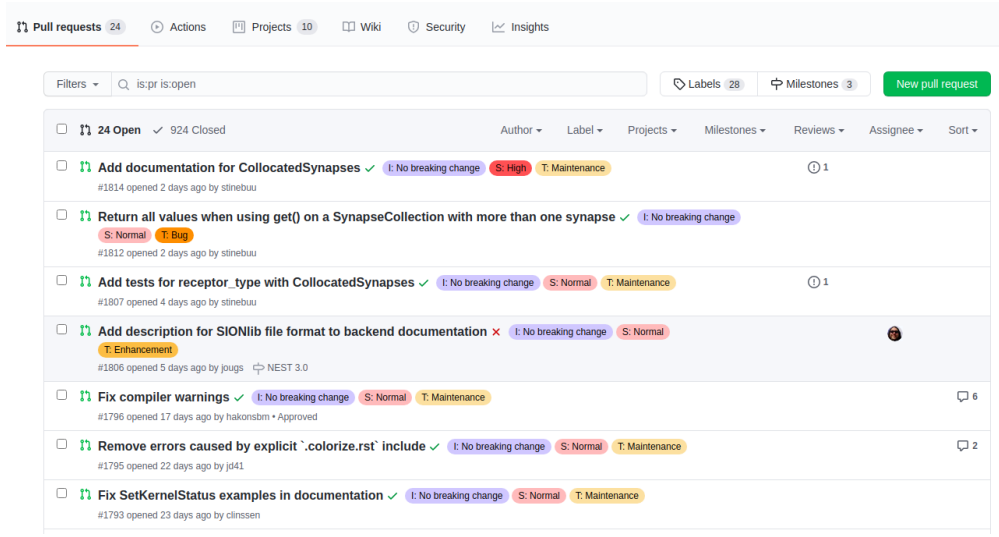
    On the K computer in Japan

By Toshihiro Matsui from Tsukuba and Yokohama, Japan - 京コンピュータ, CC BY 2.0



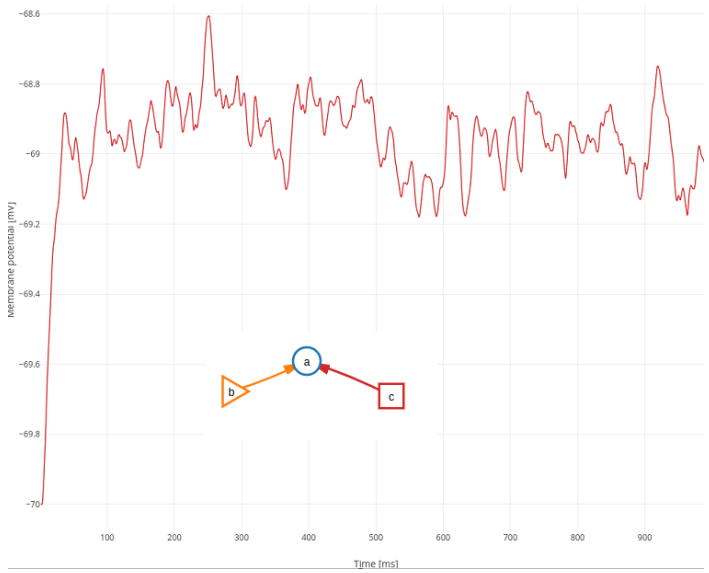
The NEST simulator

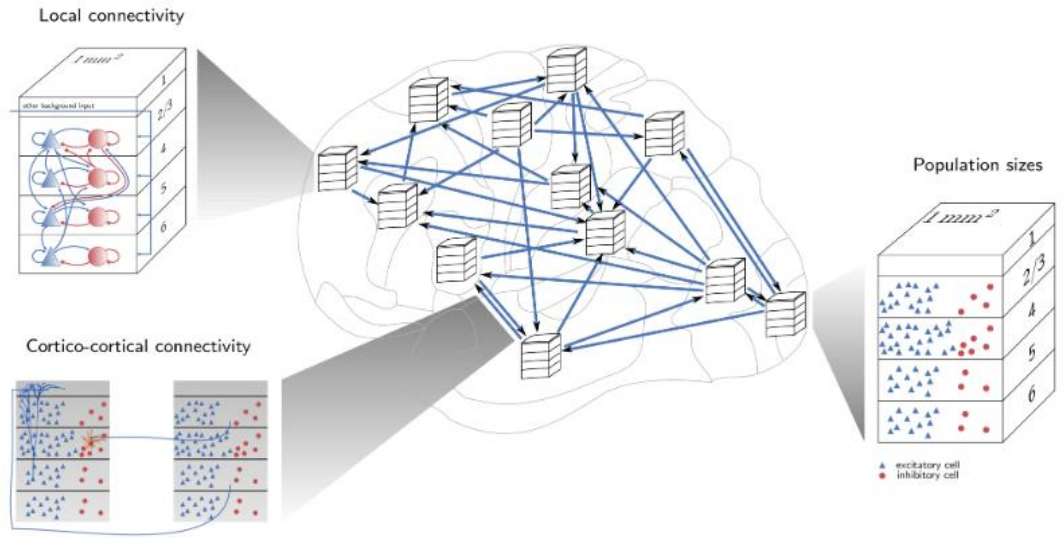
- Developed since 1995
- Active developer community
- New models, connectivity rules, etc. can be added



The NEST simulator

- A key requirement of NEST is that it should work on models of any size
- Run on laptops and supercomputers





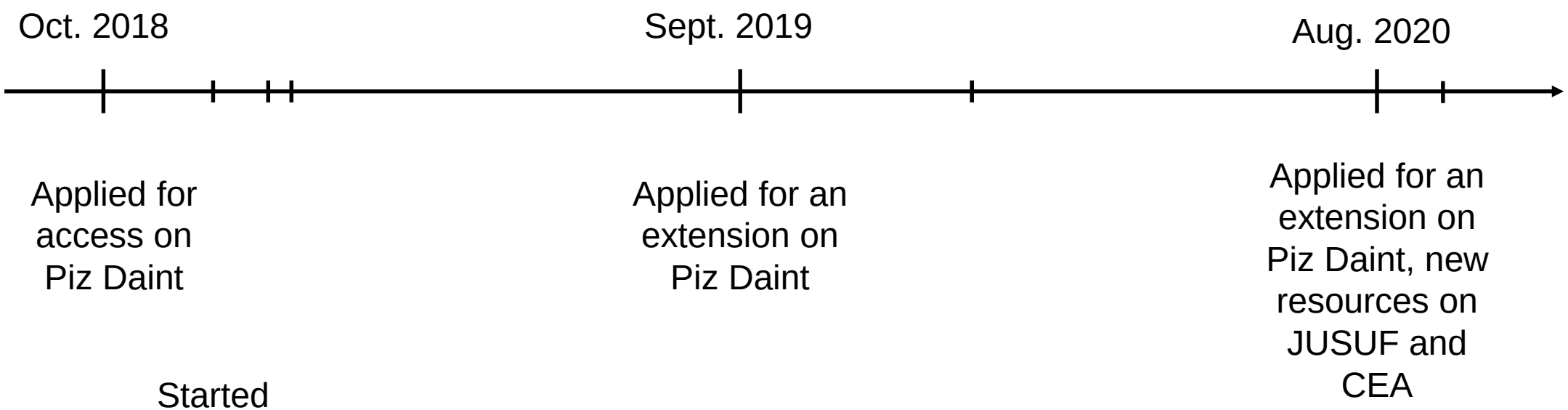
The NEST simulator

- Reliable science: important to test
- Efficient science: Performance counts

```
  test_PushPop_NumPy (nest.tests.test_stack.StackTestCase) ... ok
  test_PushPop_no_NumPy (nest.tests.test_stack.StackTestCase) ... SKIP: M
  test_stack_checker (nest.tests.test_stack.StackTestCase) ... ok
  GetDefaults ... ok
  GetKernelStatus ... ok
  SetDefaults ... ok
  Multiple threads ... ok
  Test plot_network ... SKIP: pydot not found
  OK (SKIP=8)

Phase 8: Running C++ tests (experimental)
-----------------------------------------
  Not running C++ tests because NEST was compiled without Boost.


--------------------------------------------------------------------------------
NEST Testsuite Results
--------------------------------------------------------------------------------
Phase                    Tests   Skipped  Failures    Errors      Time
--------------------------------------------------------------------------------
01 basetests                 6         0         0         0       0.0
02 selftests                 8         4         0         0       0.0
03 unittests               185         1         0         0      56.0
04 regressiontests          94         6         0         0      12.0
05 mpitests                 79         0         0         0       1.0
07 pynesttests             792         8         0         0      96.2
--------------------------------------------------------------------------------
Total                     1164        19         0         0     165.2
--------------------------------------------------------------------------------

The NEST Testsuite passed successfully.

--------------------------------------------------------------------------------
```

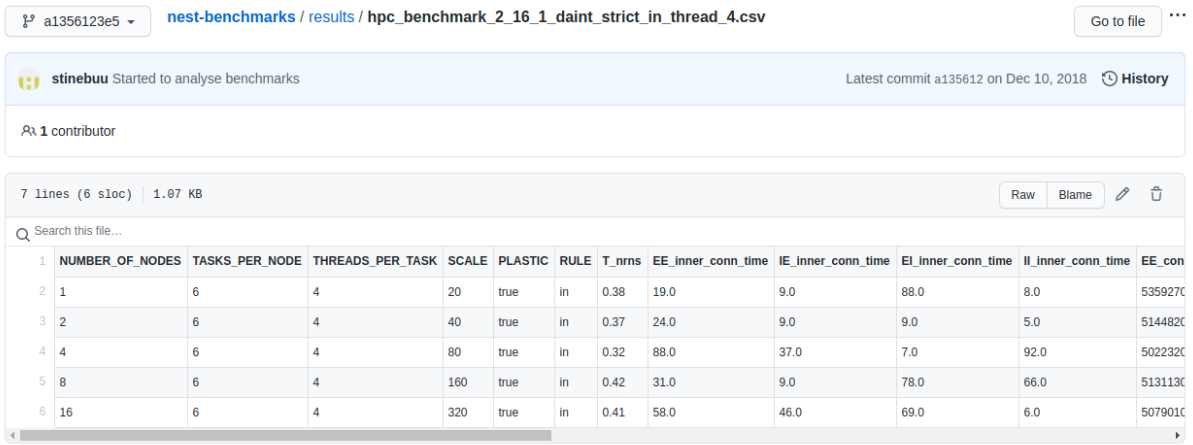
Our use of FENIX, first period

Timeline


Oct. 2018
Sept. 2019
Aug. 2020
Applied for access on Piz Daint
Started benchmarking
Applied for an extension on Piz Daint
Applied for an extension on Piz Daint, new resources on JUSUF and CEA

How we got started

- We sent our first ICEI application for HBP resources on the 23rd of October 2018
- Applied for 25 000 node x hours on Piz Daint
- Received allocations 12th of November 2018
- Got access to Piz Daint on 20th of November 2018



a1356123e5 nest-benchmarks / results / hpc_benchmark_2_16_1_daint_strict_in_thread_4.csv Go to file

stinebuu Started to analyse benchmarks Latest commit a135612 on Dec 10, 2018 History

1 contributor

7 lines (6 sloc) | 1.07 KB Raw Blame

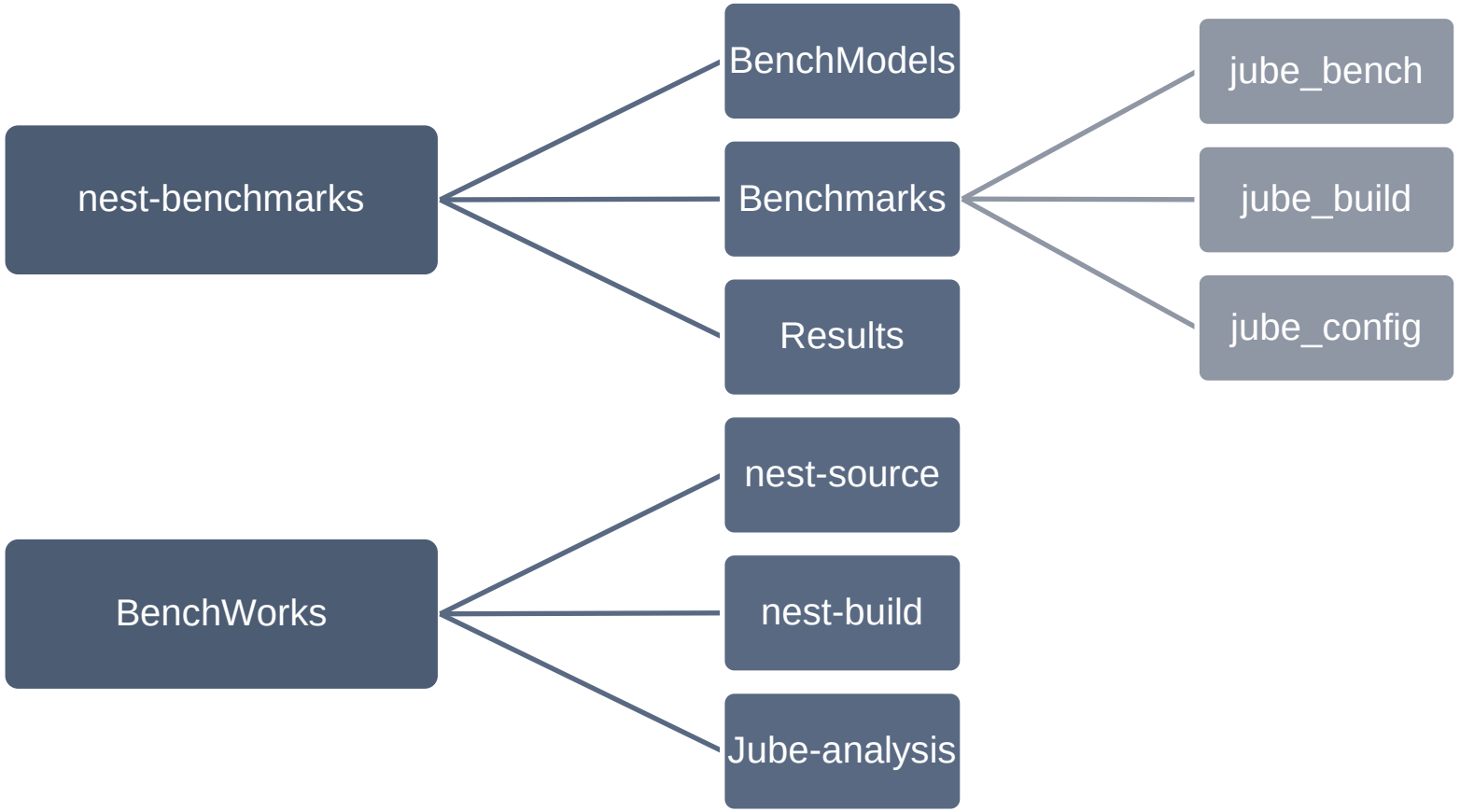
Search this file...

	NUMBER_OF_NODES	TASKS_PER_NODE	THREADS_PER_TASK	SCALE	PLASTIC	RULE	T_nrns	EE_inner_conn_time	IE_inner_conn_time	EI_inner_conn_time	II_inner_conn_time	EE_con
2	1	6	4	20	true	in	0.38	19.0	9.0	88.0	8.0	535927(
3	2	6	4	40	true	in	0.37	24.0	9.0	9.0	5.0	514482(
4	4	6	4	80	true	in	0.32	88.0	37.0	7.0	92.0	502232(
5	8	6	4	160	true	in	0.42	31.0	9.0	78.0	66.0	513113(
6	16	6	4	320	true	in	0.41	58.0	46.0	69.0	6.0	507901(

Benchmark set-up

- We use the JUBE Benchmarking Environment to run our benchmarks
  - Framework for creating benchmark sets, run the benchmarks on different systems and analyze the results
  - Can set up parameters, shell commands, system dependencies, ++
    - Less error prone benchmark system
  - Versatile and robust
  - Can analyze and extract results from your benchmark output files
  - Developed by the Jülich Supercomputing Centre at FZ Jülich

Layout


nest-benchmarks
BenchModels
Benchmarks
Results
jube_bench
jube_build
jube_config
BenchWorks
nest-source
nest-build
Jube-analysis

Benchmark models

- Need a large variety when testing NEST
  - Models that use PyNEST
  - Models with networks distributed in space
  - HPC benchmarks with long history of benchmarking
  - Realistic models
  - Models that use different connectivity rules
- The focus on the benchmarks has been connectivity

Benchmark models

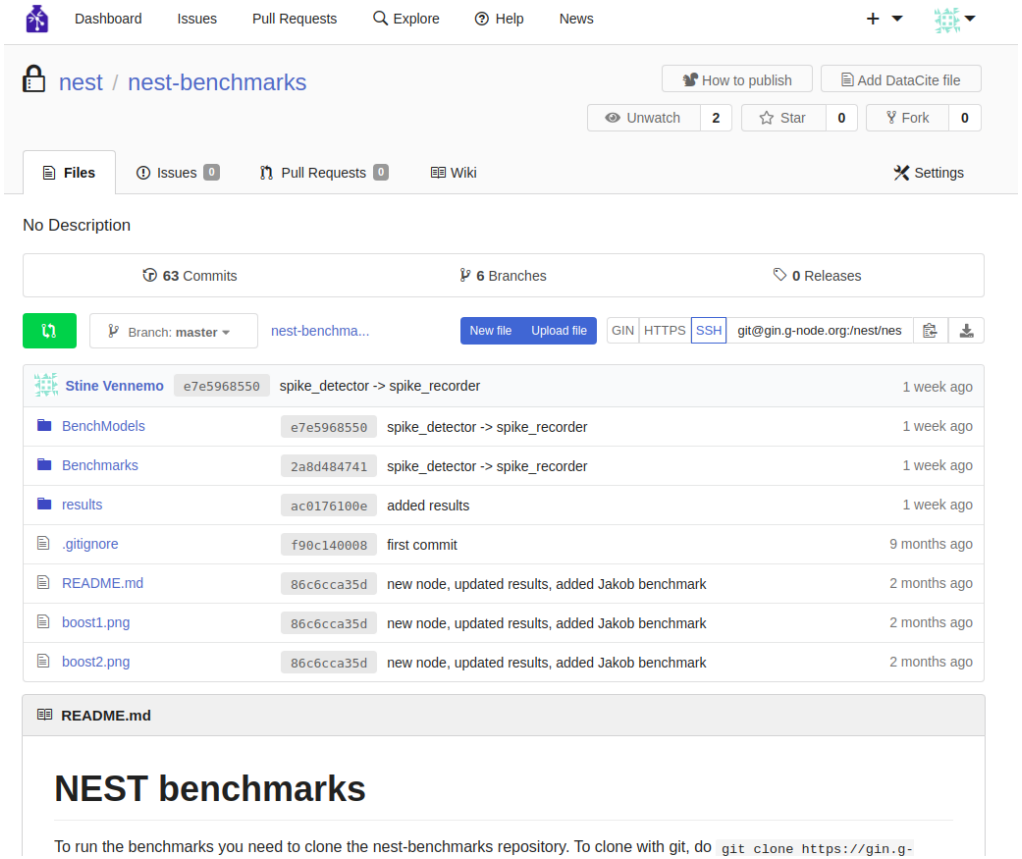
- HPC benchmark model
  - 2 population model
  - Both connected to themselves and each other
    - Every neuron connected to 11,250 other neurons
  - Quite easy to switch connectivity rules
  - Long tradition as benchmark
- Population model
  - Consist of constant x 20 populations
    - Each containing 5000 neurons
  - Every population connects to 100 random populations
  - Fan in of 50 neurons for each projection
    - Each neuron connects to 5000 other neurons

Benchmark models

- Multi-area model
  - Schmidt et al (2018) *Multi-scale account of the network structure of macaque visual cortex Brain Structure and Function* https://doi.org/10.1007/s00429-017-1554-4
  - See also: https://inm-6.github.io/multi-area-model/
  - Represents the visual pathway
  - 32 different areas, each with 8 populations
  - 4.1×10^6 neurons, 2.4×10^{10} connections
- 4x4 mesocircuit model
  - Senk et al (2018) *Reconciliation of weak pair-wise spike-train correlations and highly coherent local field potentials across space* https://arxiv.org/abs/1805.10235
  - 8 populations in 4 cortical layers
    - Distributed in space
  - 1.2×10^6 neurons, 5.5×10^9 connections

Running the benchmarks

- Run the benchmarks from the ${SCRATCH} folder
- Use git (G-node) to transfer the results
  - German neuroinformatics node
  - Developed for neuroscientists
  - Focus on data management



Running the benchmarks

Each benchmark run consists of the following steps:

1. Commit any potential modifications of benchmark scripts to benchmark repository
2. Submit benchmark job

   ```
   jube run <path>/nest-benchmarks/jube_bench/<benchmark-file>.xml
   ```

   noting the JUBE output directory and benchmark run counter of the benchmark
3. Collect benchmark output and condense to report in cvs-format

   ```
   jube analyse <jube_out dir> -i <run #>
   jube result <jube_out dir> -i <run #> > <results dir>/<result-name>.csv
   ```
4. Commit report `<result-name>.csv` to benchmark repository
5. Visualize results using jupyter notebook.

Scaling and system for first allocation

- We focused on weak scaling
- Used 1-32 compute nodes
  - 36 virtual processes per compute node
  - 6 threads per MPI rank (18 core CPUs)
- For each benchmark:
  - Chose a scale *s* for a single compute node
  - Used this base scale to scale up the model relative to number of compute nodes

NUM_NODES	TASKS/NODE	THREADS/TASK	num_tasks	NUM_VPS	SCALE
1	6	6	6	36	20
2	6	6	12	72	40
4	6	6	24	144	80
8	6	6	48	288	160
16	6	6	96	576	320
32	6	6	192	1152	640

Challenges

- Difficult to keep track of all the different runs, versions, results
  - Using commit hashes still makes it difficult to keep track
- Generate a lot of output, what should we keep?
  - Having an enormous number of result files is not helpful when you want to look back

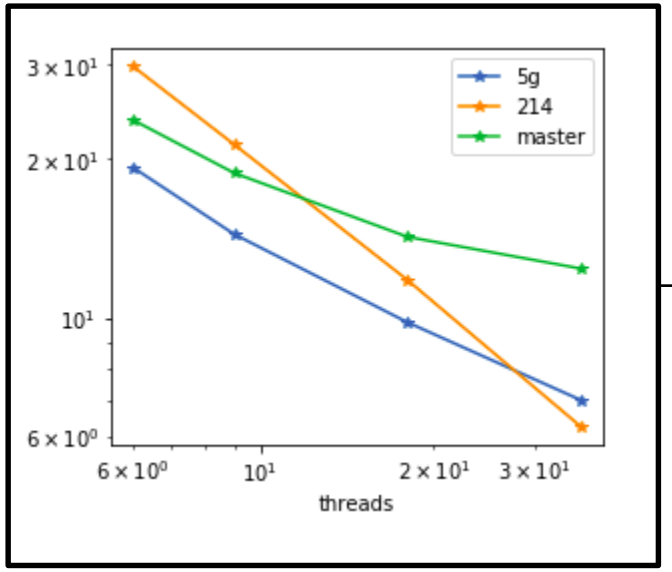

```
k_f104c6d3_daint_strict_fig4_scale_10.csv')
k_42ed8118_daint_strict_fig4_scale_10.csv')
k_02f20d15_daint_strict_fig4_scale_10.csv')
k_c1a600e1_daint_strict_fig4_scale_10.csv')

k_840a86d1_daint_strict_fig4_scale_10.csv') # Denne har litt og
_96bb9d06_daint_strict_fig4_scale_10.csv') # Det er denne
k_96bb9d06-modified_daint_strict_fig4_scale_10.csv')
k_96bb9d06-modified-2_daint_strict_fig4_scale_10.csv')

k_19c4f1b1_daint_strict_fig4_scale_10.csv')
k_3b851ef7_daint_strict_fig4_scale_10.csv') # Denne versionen a
k_0d6165be_daint_strict_fig4_scale_10.csv') # ok
k_47115c6_daint_strict_fig4_scale_10.csv') # ikke ok
_3dae2515_daint_strict_fig4_scale_10.csv') # ikke ok This is th
k_3dae2515-modified_daint_strict_fig4_scale_10.csv')
k_3dae2515-modified-2_daint_strict_fig4_scale_10.csv') # Denne
k_21e89ab2_daint_strict_fig4_scale_10.csv') # ok
```

```
# third jump
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_84483ac4_daint_strict_fig4_scale_10.csv') # Siste fra hackath
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_6f06fa64_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_7df5fe94_daint_strict_fig4_scale_10.csv')


# backwards from yellow
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_18acd78a_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_b6bdad48_daint_strict_fig4_scale_10.csv') # ikke ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_b6bdad48-modified_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_921d34a4_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ecce98ca_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_52038f3a_daint_strict_fig4_scale_10.csv') # Ikke ok, det er d

# Second jump from yellow
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_5ae7dece_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_4f5c3267_daint_strict_fig4_scale_10.csv') # ok? Den er ikke l
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_179025e9_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_33676af8_daint_strict_fig4_scale_10.csv') # ikke ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ef5d7a71_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ef5d7a71-2_daint_strict_fig4_scale_10.csv')

# master right after Block Vector
fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_02114048_daint_strict_fig4_scale_10.csv') # Right after BV
fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_02114048-2_daint_strict_fig4_scale_10.csv')

#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3cacb545-BV-1_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3cacb545-BV_daint_strict_fig4_scale_10.csv') # BV PR merge
```

Help and difficulties

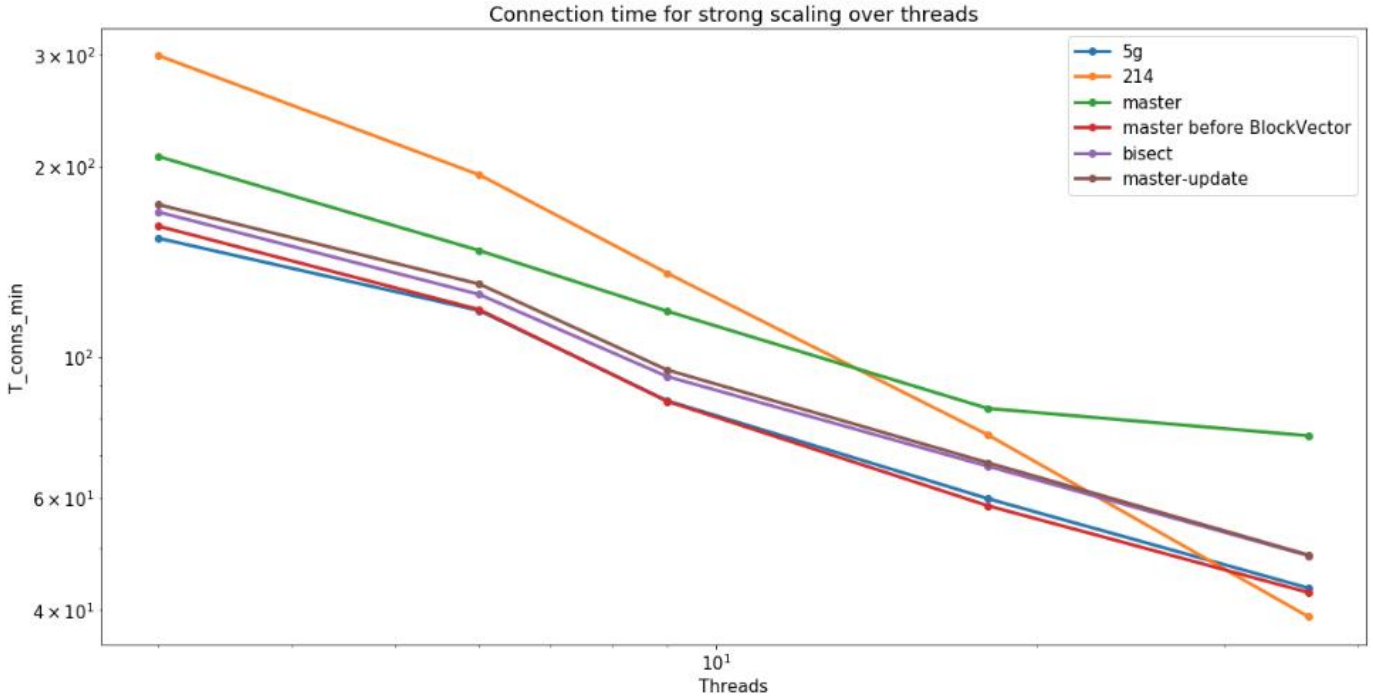
- Received support from help@cscs.ch
  - Fast response times
  - Useful recommendations

Bisectioning to locate performance regressions




```
fig4_s10_214['T_bld'] = fig4_s10_214[['T_bld_xn', 'T_bld_nx']].min(axis=1)

# bisect
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_f104c6d3_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_42ed8118_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_02f20d15_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_c1a600e1_daint_strict_fig4_scale_10.csv')
# first jump
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_840a86d1_daint_strict_fig4_scale_10.csv') # Denne har litt og
fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_96bb9d06_daint_strict_fig4_scale_10.csv') # Det er denne
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_96bb9d06-modified_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_96bb9d06-modified-2_daint_strict_fig4_scale_10.csv')


# second jump
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_19c4f1b1_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3b851ef7_daint_strict_fig4_scale_10.csv') # Denne versionen a
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_0d6165be_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_47115c6_daint_strict_fig4_scale_10.csv') # ikke ok
fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3dae2515_daint_strict_fig4_scale_10.csv') # ikke ok This is th
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3dae2515-modified_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_3dae2515-modified-2_daint_strict_fig4_scale_10.csv') # Denne
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_21e89ab2_daint_strict_fig4_scale_10.csv') # ok

# third jump
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_84483ac4_daint_strict_fig4_scale_10.csv') # Siste fra hackath
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_6f06fa64_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_7df5fe94_daint_strict_fig4_scale_10.csv')


# backwards from yellow
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_18acd78a_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_b6bdad48_daint_strict_fig4_scale_10.csv') # ikke ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_b6bdad48-modified_daint_strict_fig4_scale_10.csv')
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_921d34a4_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ecce98ca_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_52038f3a_daint_strict_fig4_scale_10.csv') # Ikke ok, det er d

# Second jump from yellow
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_5ae7dece_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_4f5c3267_daint_strict_fig4_scale_10.csv') # ok? Den er ikke l
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_179025e9_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_33676af8_daint_strict_fig4_scale_10.csv') # ikke ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ef5d7a71_daint_strict_fig4_scale_10.csv') # ok
#fig4_s10_bisect = pd.read_csv('results/hpc_benchmark_ef5d7a71-2_daint_strict_fig4_scale_10.csv')
```

Bisectioning to locate performance regressions



Merged **Improve connection performance over threads** #1119

heplesser merged 8 commits into `nest:master` from `stinebuu:primary_connections` on Mar 22, 2019

stinebuu commented on Feb 8, 2019 — Member

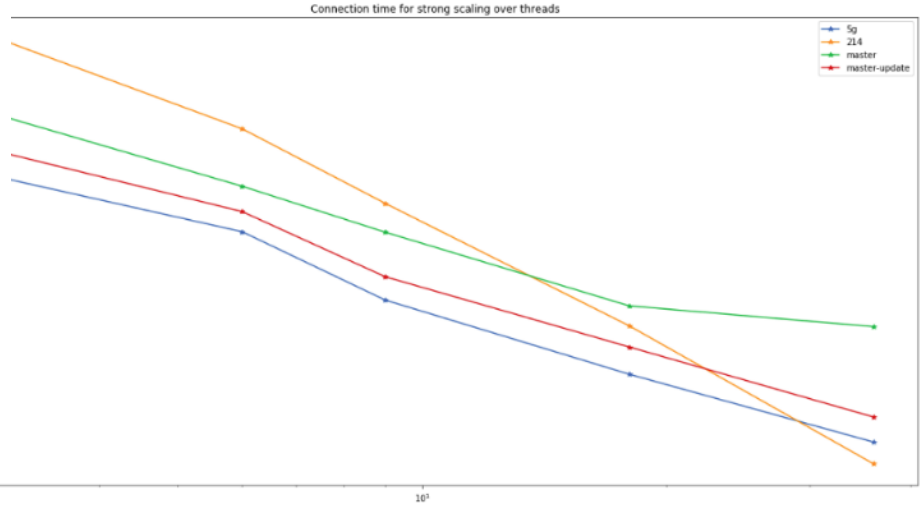
This PR contains two new changes that improve the connection performance, especially over increasing number of threads and for large networks:

all threads writing to `has_primary_connections_` or `secondary_connections_exist_` when we add a new
n, we check if the individual thread has primary or secondary connections, and only write to the variable if the thread
eviously updated it. We also make sure that the threads don't try to write to the variable all at once.

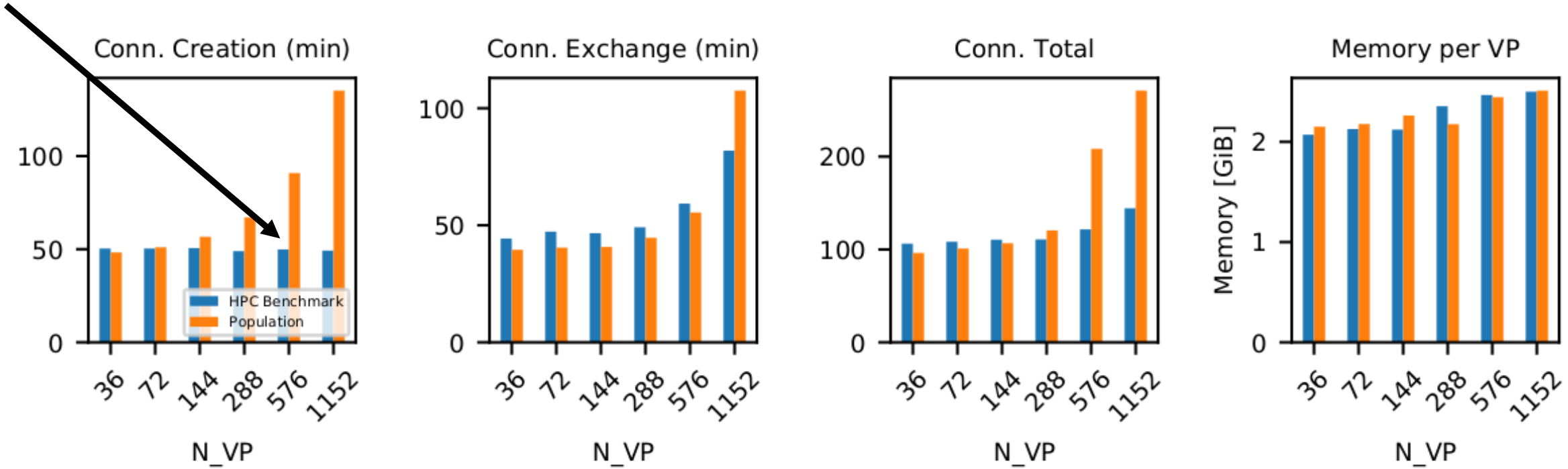
creating a new empty dummy parameter dictionary every time we connect two nodes, we create a static dummy
once, and then use this dummy parameter every time we need an empty dictionary.

s improves the performance of master quite a bit, but we are not on the level with the NEST version used in the 5g
id), see the plots below. The difference between `5g` and this updated master (`master-update`) comes from the

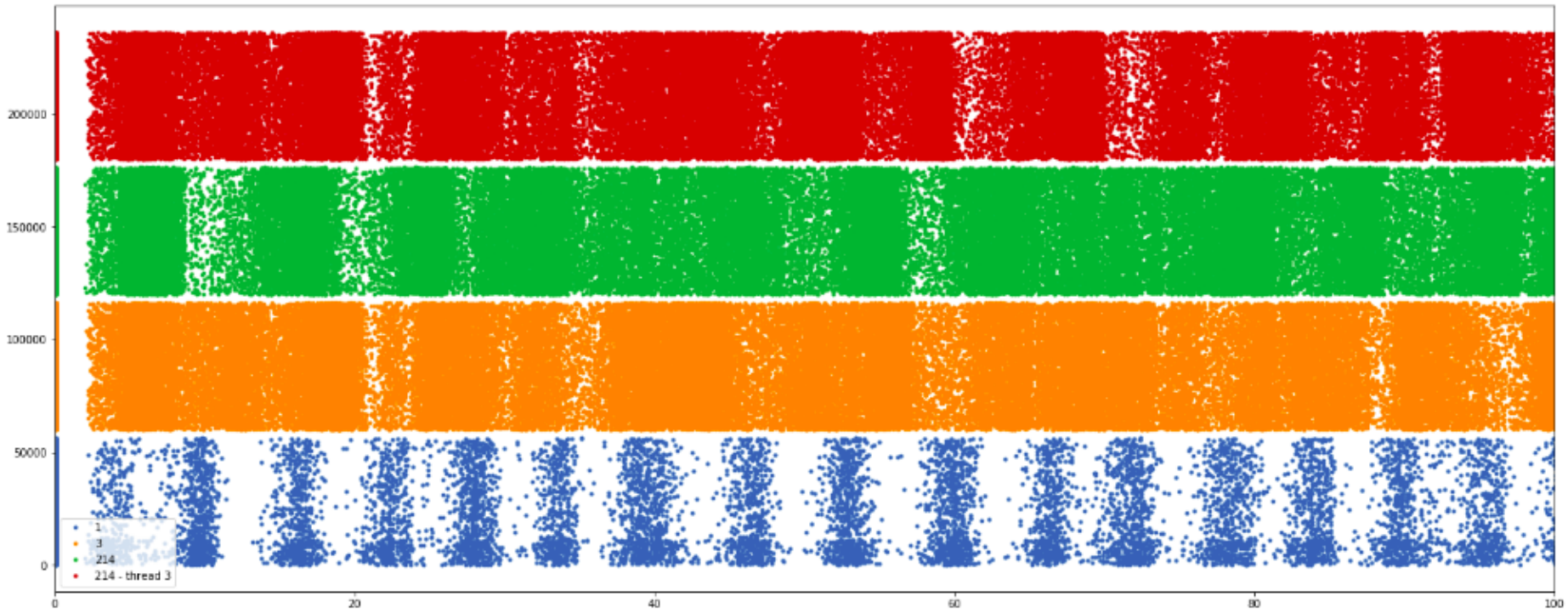
n the `hpc_benchmark` with a scale of 10, which equals 112500 neurons and 1.265738e+09 connections, on 1 node
nt supercomputer, on an increasing number of threads per MPI process (strong scaling over threads). The number of
6, 9, 18, and 36. The figure shows the connection time vs threads.



Identifying use cases with poor scaling



Debugging large-network threading error

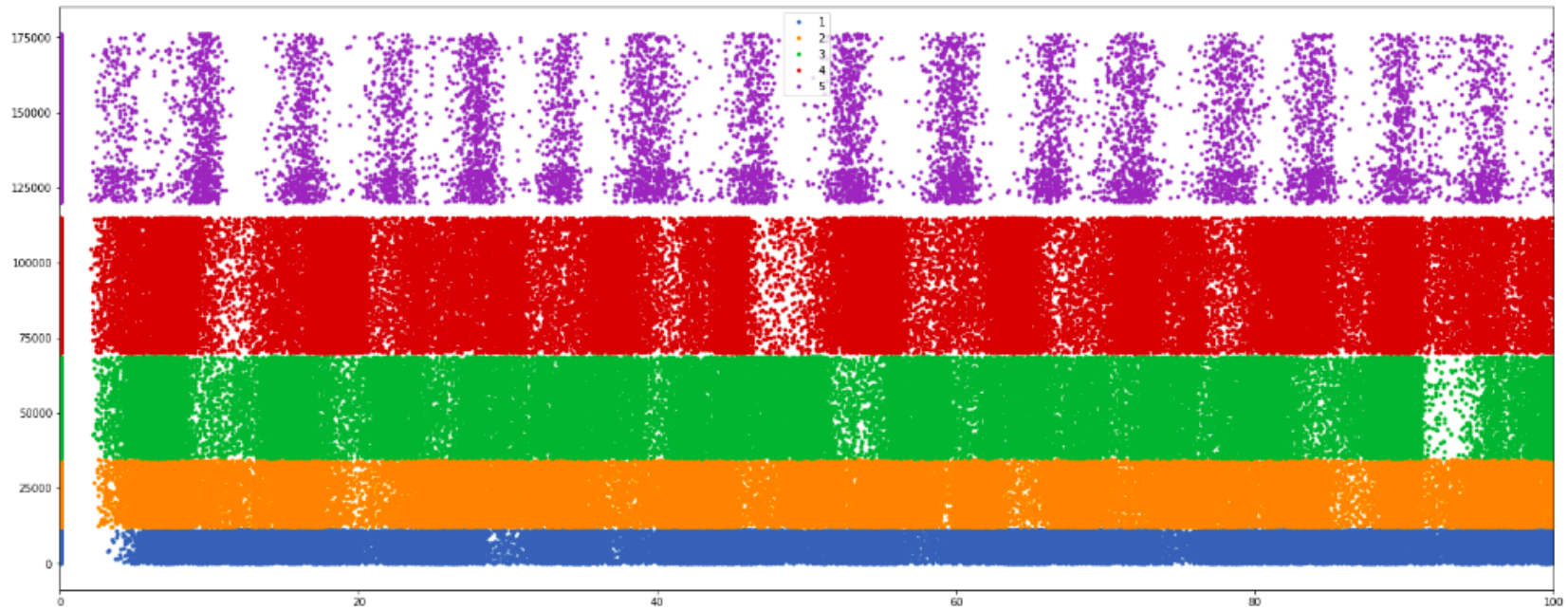


Different scales

In [100]:
```
s1_sw_scale1 = pd.read_csv('all_spikes_mpi1_thread1_switched-scale1.gdf', sep='\t', names=['s', 't'], index_col=Fal
s1_sw_scale2 = pd.read_csv('all_spikes_mpi1_thread1_switched-scale2.gdf', sep='\t', names=['s', 't'], index_col=Fal
s1_sw_scale3 = pd.read_csv('all_spikes_mpi1_thread1_switched-scale3.gdf', sep='\t', names=['s', 't'], index_col=Fal
s1_sw_scale4 = pd.read_csv('all_spikes_mpi1_thread1_switched-scale4.gdf', sep='\t', names=['s', 't'], index_col=Fal
```

In [101]:
```
plt.rcParams['figure.figsize'] = (25, 10)
plt.plot(s1_sw_scale1.t, s1_sw_scale1.s, '.', label='1');
plt.plot(s1_sw_scale2.t, 12000+s1_sw_scale2.s, '.', label='2');
plt.plot(s1_sw_scale3.t, 35000+s1_sw_scale3.s, '.', label='3');
plt.plot(s1_sw_scale4.t, 70000+s1_sw_scale4.s, '.', label='4');
plt.plot(s1_switched.t, 120000+s1_switched.s, '.', label='5');
plt.legend();
plt.xlim(0,100)
```

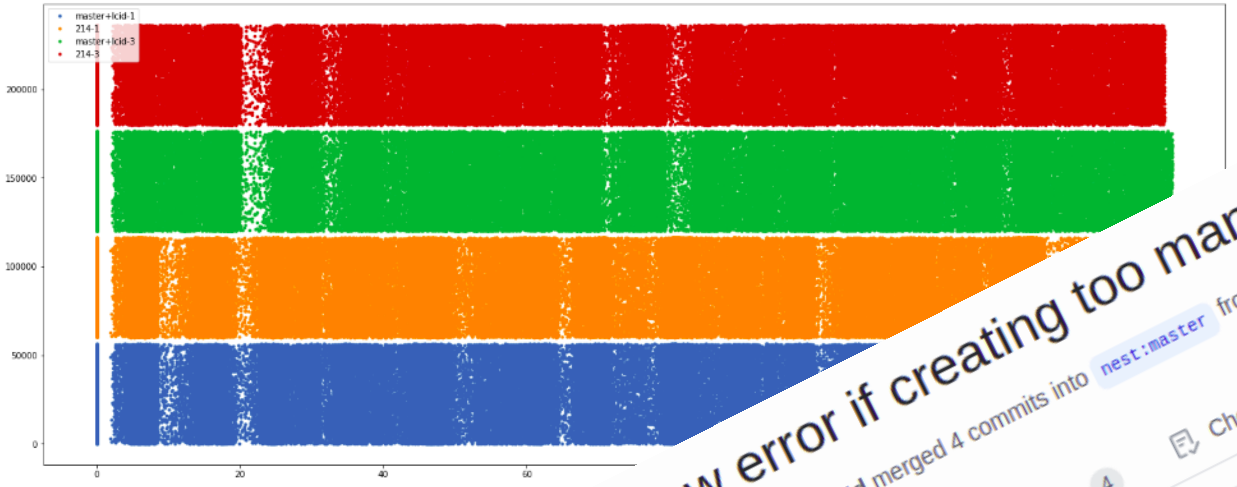
Out[101]: (0, 100)



HIGHER NUMBER OF BITS FOR LCID

In [118]:
```
s1_lcid = pd.read_csv('all_spikes_mpi1_thread1_lcid.gdf', sep='\t', names=['s', 't'], index_col=False)
s1_214_with_num = pd.read_csv('all_spikes_mpi1_thread1_214_with_num.gdf', sep='\t', names=['s', 't'], index_col=Fals
s3_lcid = pd.read_csv('all_spikes_mpi1_thread3_lcid.gdf', sep='\t', names=['s', 't'], index_col=False)
s3_214_with_num = pd.read_csv('all_spikes_mpi1_thread3_214_with_num.gdf', sep='\t', names=['s', 't'], index_col=Fals
```

In [119]:
```
plt.rcParams['figure.figsize'] = (25, 10)
plt.plot(s1_lcid.t, s1_lcid.s, '.', label='master+lcid-1');
plt.plot(s1_214_with_num.t, 60000+s1_214_with_num.s, '.', label='214-1');
plt.plot(s3_lcid.t, 120000+s3_lcid.s, '.', label='master+lcid-3');
plt.plot(s3_214_with_num.t, 180000+s3_214_with_num.s, '.', label='214-3');
plt.legend();
#plt.xlim(-1,100)
```



Throw error if creating too many connections on one thread #1103

terhorstd merged 4 commits into nest:master from hakonsbm:connection_overflow_error on Mar 18, 2019

Merged

Conversation 12 Commits 4 Checks 0 Files changed 1

hakonsbm commented on Jan 14, 2019 Member

There is currently a limit of 2^{27} connections per synapse per thread, due to having 27 bits for the lcid in the Target class. With this PR an exception is thrown if attempting to create more connections than Target can handle.

Fixes #1102.

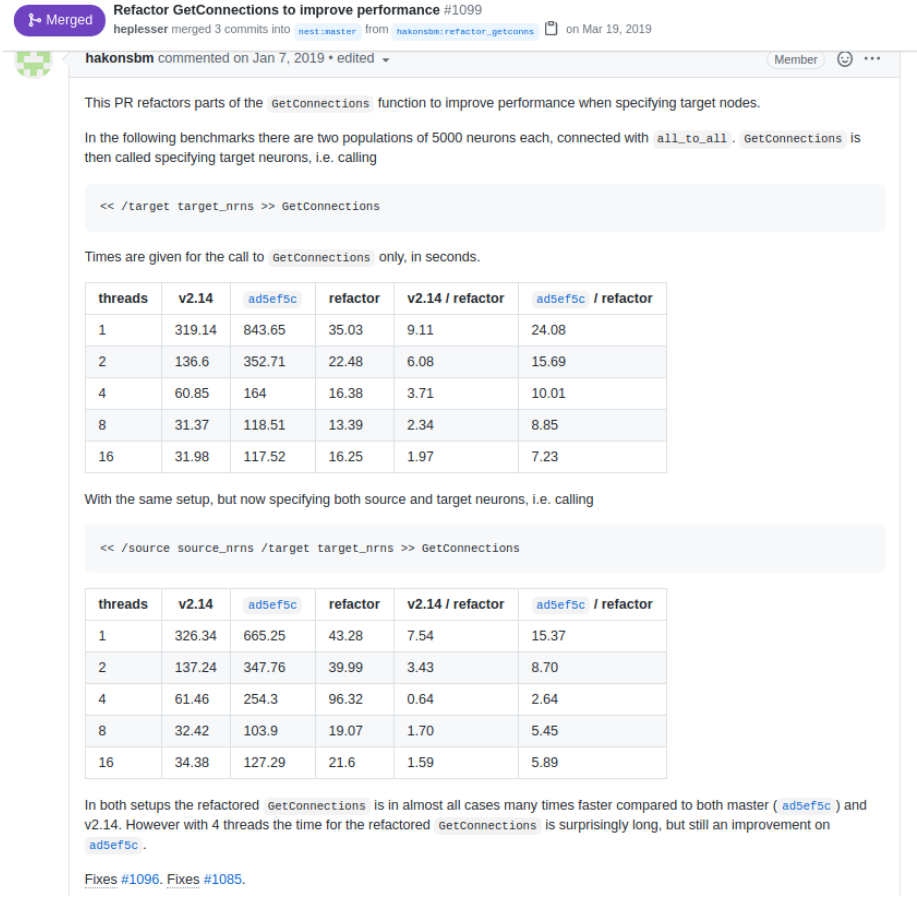
FENIX enabled PRs for first allocation period

#1099: *Refactor GetConnections to improve performance*

#1101: *Take multiplicity into account in local_spike_counter*

#1103: *Throw error if creating too many connections on one thread*

#1105: *Add MPI test for correct number of spikes when multiplicity > 1*



Merged **Refactor GetConnections to improve performance** #1099

heplesser merged 3 commits into `nest:master` from `hakonsbm:refactor_getconns` on Mar 19, 2019

hakonsbm commented on Jan 7, 2019 • edited — Member

This PR refactors parts of the `GetConnections` function to improve performance when specifying target nodes.

In the following benchmarks there are two populations of 5000 neurons each, connected with `all_to_all`. `GetConnections` is then called specifying target neurons, i.e. calling

```
<< /target target_nrns >> GetConnections
```

Times are given for the call to `GetConnections` only, in seconds.

threads	v2.14	ad5ef5c	refactor	v2.14 / refactor	ad5ef5c / refactor
1	319.14	843.65	35.03	9.11	24.08
2	136.6	352.71	22.48	6.08	15.69
4	60.85	164	16.38	3.71	10.01
8	31.37	118.51	13.39	2.34	8.85
16	31.98	117.52	16.25	1.97	7.23

With the same setup, but now specifying both source and target neurons, i.e. calling

```
<< /source source_nrns /target target_nrns >> GetConnections
```

threads	v2.14	ad5ef5c	refactor	v2.14 / refactor	ad5ef5c / refactor
1	326.34	665.25	43.28	7.54	15.37
2	137.24	347.76	39.99	3.43	8.70
4	61.46	254.3	96.32	0.64	2.64
8	32.42	103.9	19.07	1.70	5.45
16	34.38	127.29	21.6	1.59	5.89

In both setups the refactored `GetConnections` is in almost all cases many times faster compared to both master (`ad5ef5c`) and v2.14. However with 4 threads the time for the refactored `GetConnections` is surprisingly long, but still an improvement on `ad5ef5c` .

Fixes #1096. Fixes #1085.

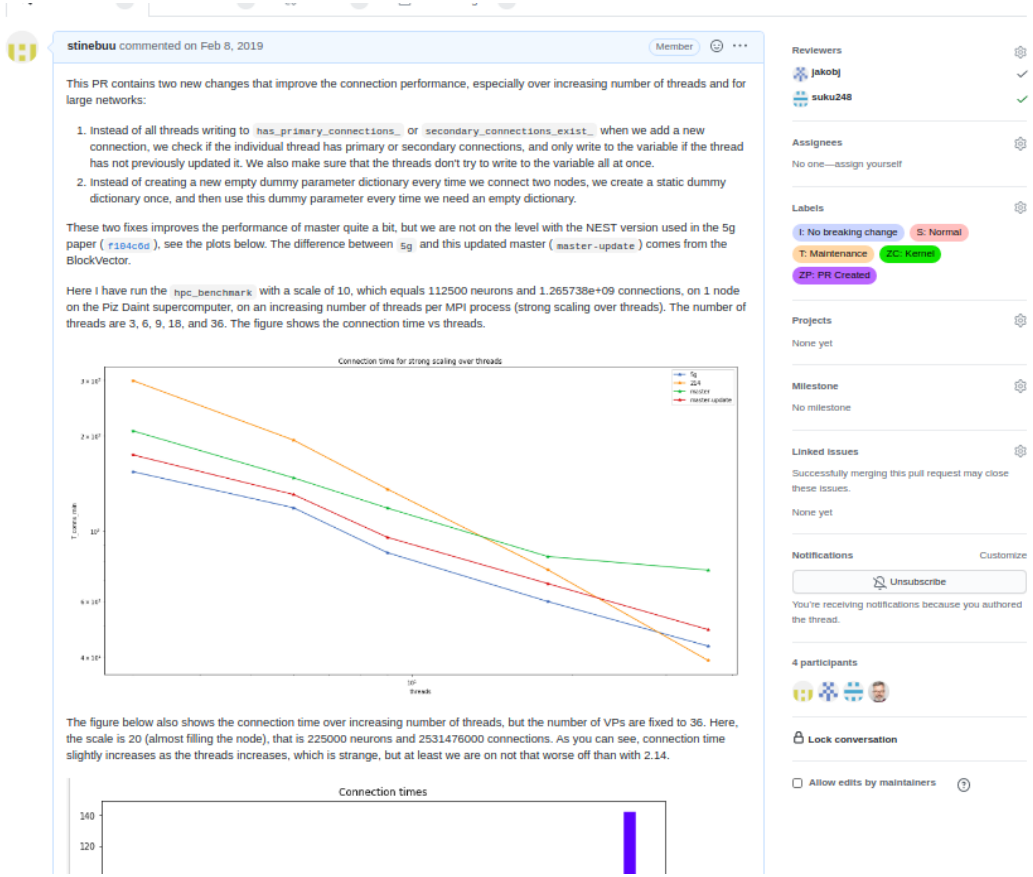
FENIX enabled PRs for first allocation period

#1118: *Connection sorting using Boost's sorting function*

#1119: *Improve connection performance over threads*

#1147: *Fix MPI synchronization problem in presence of very small layers*

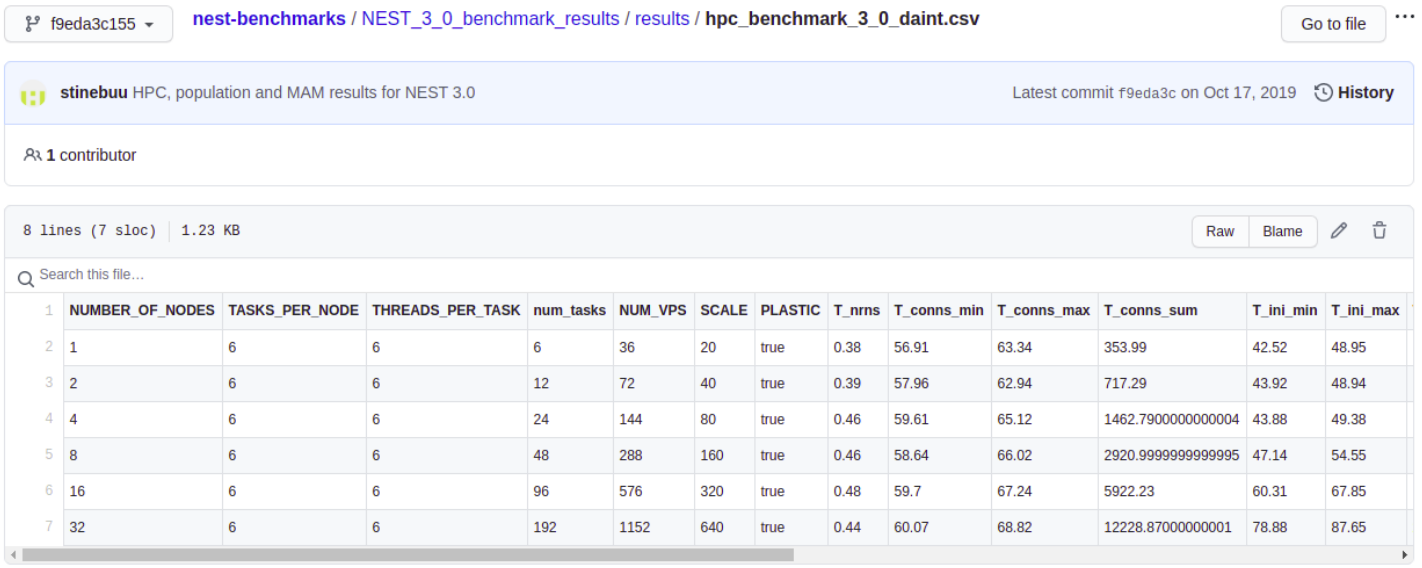
#1170: CMake option and documentation for using the Intel compiler



Second allocation period – towards NEST 3.0

Second allocation period

- We sent our extension ICEI application on the 15th of September 2019
- Received 20 000 node x hours on Piz Daint
- Received allocations on the 15th of October 2019



f9eda3c155 nest-benchmarks / NEST_3_0_benchmark_results / results / hpc_benchmark_3_0_daint.csv Go to file

stinebuu HPC, population and MAM results for NEST 3.0 Latest commit f9eda3c on Oct 17, 2019 History

1 contributor

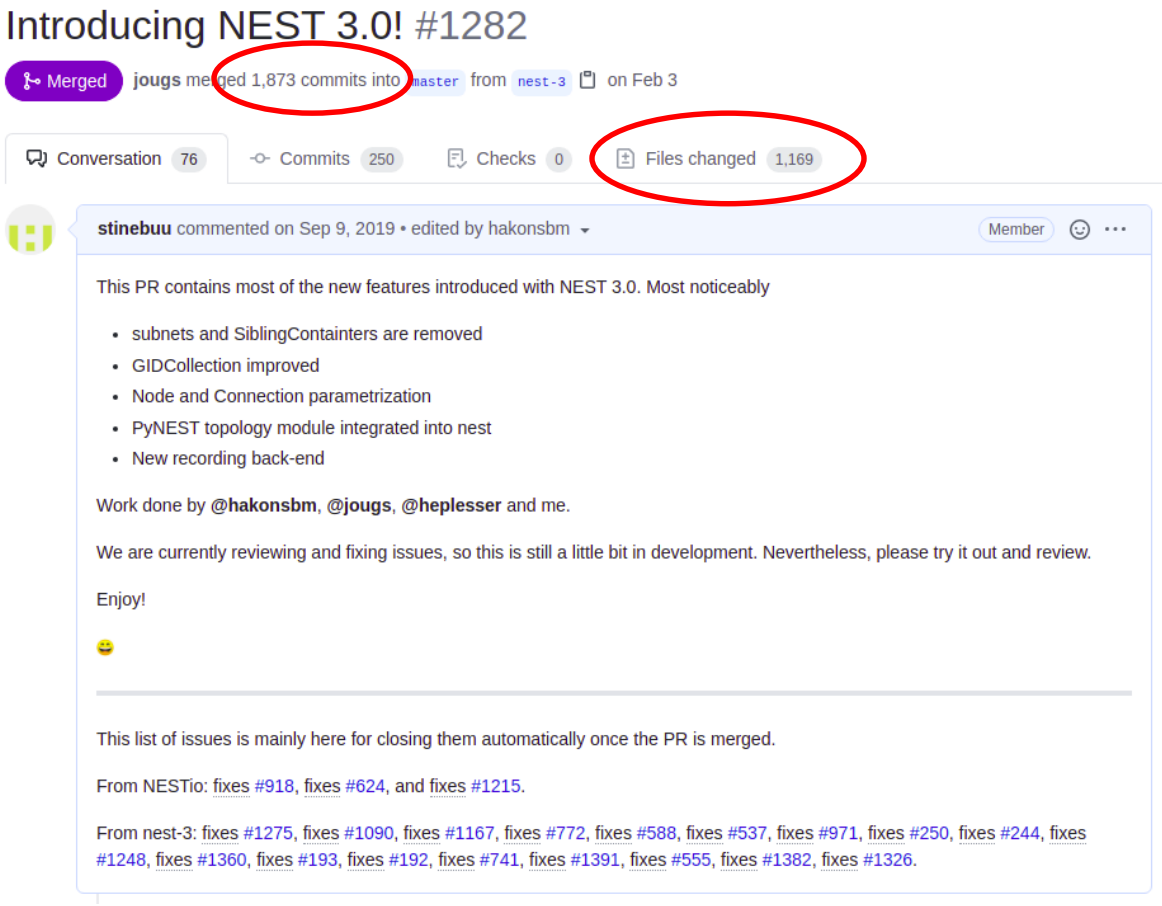
8 lines (7 sloc) 1.23 KB Raw Blame

Search this file...

	NUMBER_OF_NODES	TASKS_PER_NODE	THREADS_PER_TASK	num_tasks	NUM_VPS	SCALE	PLASTIC	T_nrns	T_conns_min	T_conns_max	T_conns_sum	T_ini_min	T_ini_max
2	1	6	6	6	36	20	true	0.38	56.91	63.34	353.99	42.52	48.95
3	2	6	6	12	72	40	true	0.39	57.96	62.94	717.29	43.92	48.94
4	4	6	6	24	144	80	true	0.46	59.61	65.12	1462.7900000000004	43.88	49.38
5	8	6	6	48	288	160	true	0.46	58.64	66.02	2920.9999999999995	47.14	54.55
6	16	6	6	96	576	320	true	0.48	59.7	67.24	5922.23	60.31	67.85
7	32	6	6	192	1152	640	true	0.44	60.07	68.82	12228.87000000001	78.88	87.65

Focus

- PR #1282: *Introducing NEST 3.0*
  - Cumulation of 2 to 3 years work
  - Introduced a lot of changes
  - New ways of representing nodes
    - Touched a lot of code
  - Restructured how we created, connected and communicated
  - Needed extensive benchmarks



Introducing NEST 3.0! #1282

Merged jougs merged 1,873 commits into master from nest-3 on Feb 3

Conversation 76 Commits 250 Checks 0 Files changed 1,169

stinebuu commented on Sep 9, 2019 • edited by hakonsbm — Member

This PR contains most of the new features introduced with NEST 3.0. Most noticeably

- subnets and SiblingContainters are removed
- GIDCollection improved
- Node and Connection parametrization
- PyNEST topology module integrated into nest
- New recording back-end

Work done by **@hakonsbm**, **@jougs**, **@heplesser** and me.

We are currently reviewing and fixing issues, so this is still a little bit in development. Nevertheless, please try it out and review.

Enjoy!

😀

This list of issues is mainly here for closing them automatically once the PR is merged.

From NESTio: fixes #918, fixes #624, and fixes #1215.

From nest-3: fixes #1275, fixes #1090, fixes #1167, fixes #772, fixes #588, fixes #537, fixes #971, fixes #250, fixes #244, fixes #1248, fixes #1360, fixes #193, fixes #192, fixes #741, fixes #1391, fixes #555, fixes #1382, fixes #1326.

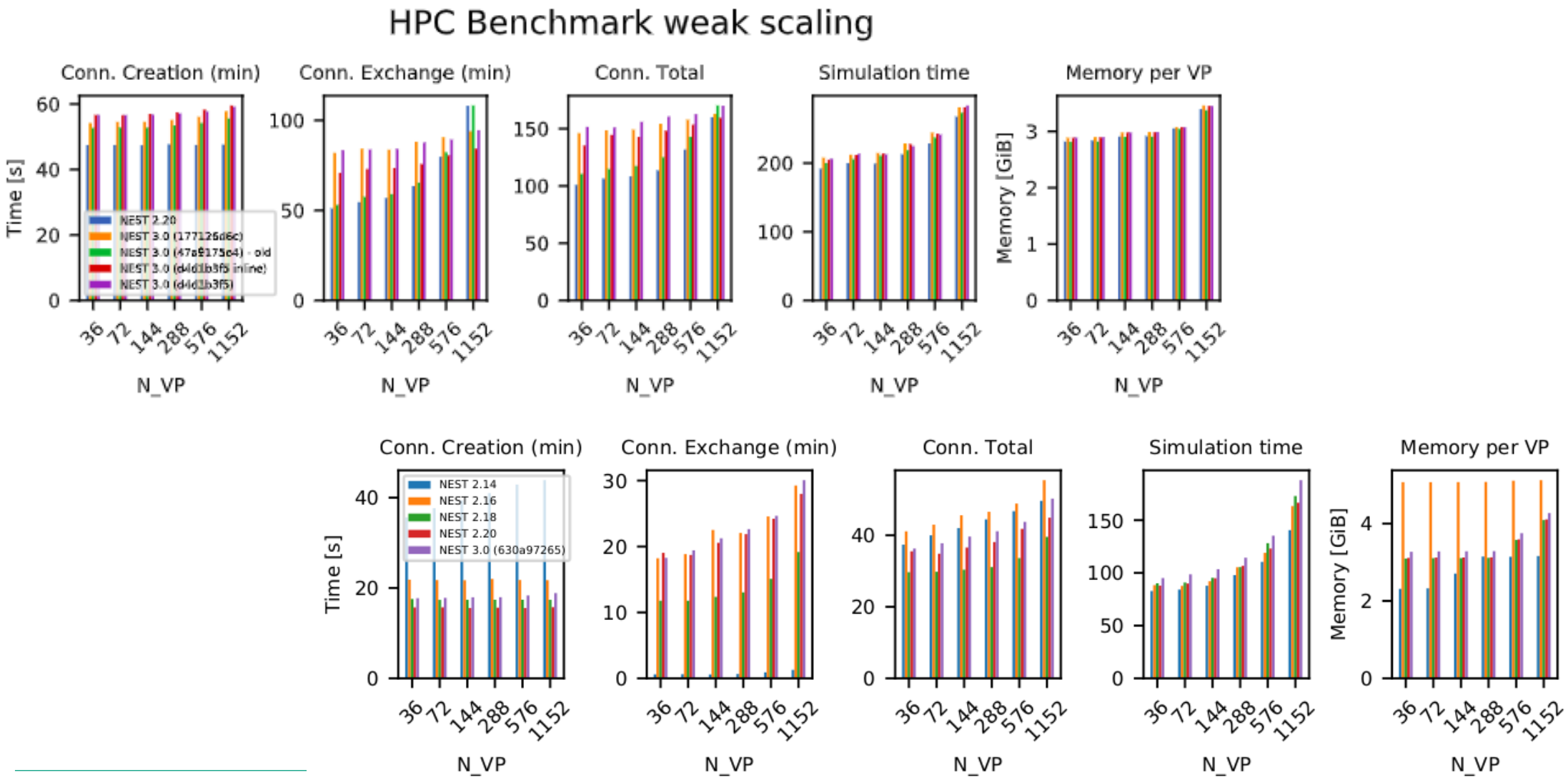
Weak and strong scaling

- Continued with our set-up from the first period:
  - Weak scaling
  - Used 1-32 compute nodes
    - 36 virtual processes per compute node
    - 6 threads per MPI rank
- Also strong scaling experiments
  - Used 1-128 compute nodes

population_model_3_0_630a97265_daint_strong_scaling_in.csv 1.9 KB Permalink History Downlo

```
git,NUMBER_OF_NODES,TASKS_PER_NODE,THREADS_PER_TASK,SCALE,PLASTIC,RULE,NUM_VPS,num_tasks,T_nrns,T_conns_min,T_conns
GIT: (NEST master@630a97265),1,6,6,5,false,in,,6,0.349245548248291,57.80458378791809,58.19709610939026,347.76463890
GIT: (NEST master@630a97265),2,6,6,5,false,in,,12,0.2866785526275635,34.527904748916626,34.84645700454712,415.45768
GIT: (NEST master@630a97265),4,6,6,5,false,in,,24,0.1848762035369873,22.54347062110901,22.779677391052246,543.50093
GIT: (NEST master@630a97265),8,6,6,5,false,in,,48,0.19488310813903809,16.54748249053955,17.054141998291016,798.8794
GIT: (NEST master@630a97265),16,6,6,5,false,in,,96,0.24720335006713867,13.52098298072815,14.03660535812378,1311.736
GIT: (NEST master@630a97265),32,6,6,5,false,in,,192,1.5379769802093506,12.033567905426025,12.666281938552856,2332.1
GIT: (NEST master@630a97265),64,6,6,5,false,in,,384,2.509483575820923,11.228553533554077,11.718058347702026,4349.80
GIT: (NEST master@630a97265),128,6,6,5,false,in,,768,8.277358531951904,10.770785331726074,11.485118865966797,8364.6
```

Benchmarking to avoid performance regressions



Challenges

- NEST API changed a lot
  - Need to update benchmark scripts
  - Different scripts for different versions

Challenges

- Modules available on Piz Daint changed
  - Older modules no longer available
  - Had to upgrade all dependencies
  - Made it difficult to have consistency in benchmarking process

- Run some interactive jobs to try to debug the problems

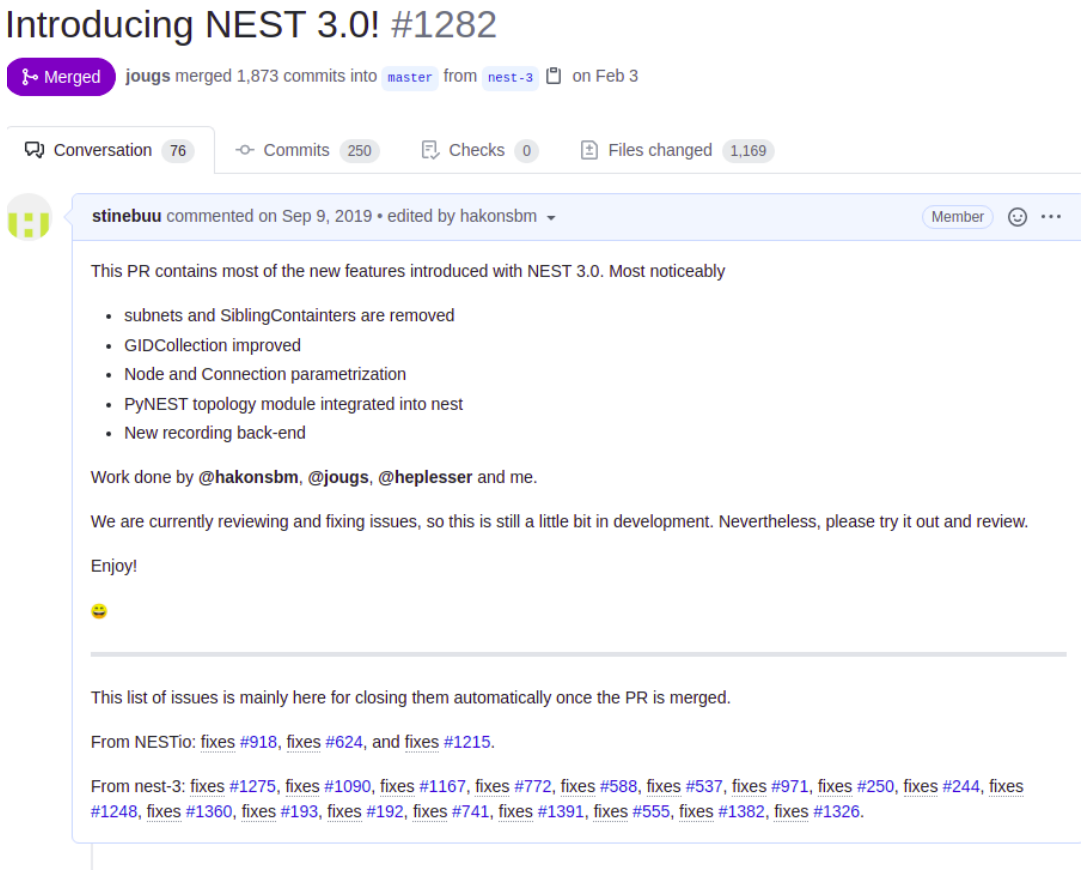
```
salloc -Cmc -pdebug -t15 -N1
```

FENIX enabled PRs for second allocation period

#1276: *Fix threading issue when connecting*

#1282: *Introducing NEST 3.0!* (1169 files!)

#1333: *Remove spurious thread_local_connectors update.*



Introducing NEST 3.0! #1282

Merged **jougs** merged 1,873 commits into `master` from `nest-3` on Feb 3

Conversation 76 · Commits 250 · Checks 0 · Files changed 1,169

stinebuu commented on Sep 9, 2019 • edited by hakonsbm — Member

This PR contains most of the new features introduced with NEST 3.0. Most noticeably

- subnets and SiblingContainters are removed
- GIDCollection improved
- Node and Connection parametrization
- PyNEST topology module integrated into nest
- New recording back-end

Work done by **@hakonsbm**, **@jougs**, **@heplesser** and me.

We are currently reviewing and fixing issues, so this is still a little bit in development. Nevertheless, please try it out and review.

Enjoy!

😀

This list of issues is mainly here for closing them automatically once the PR is merged.

From NESTio: fixes #918, fixes #624, and fixes #1215.

From nest-3: fixes #1275, fixes #1090, fixes #1167, fixes #772, fixes #588, fixes #537, fixes #971, fixes #250, fixes #244, fixes #1248, fixes #1360, fixes #193, fixes #192, fixes #741, fixes #1391, fixes #555, fixes #1382, fixes #1326.

Third allocation period – multi-system benchmarks

Third allocation period

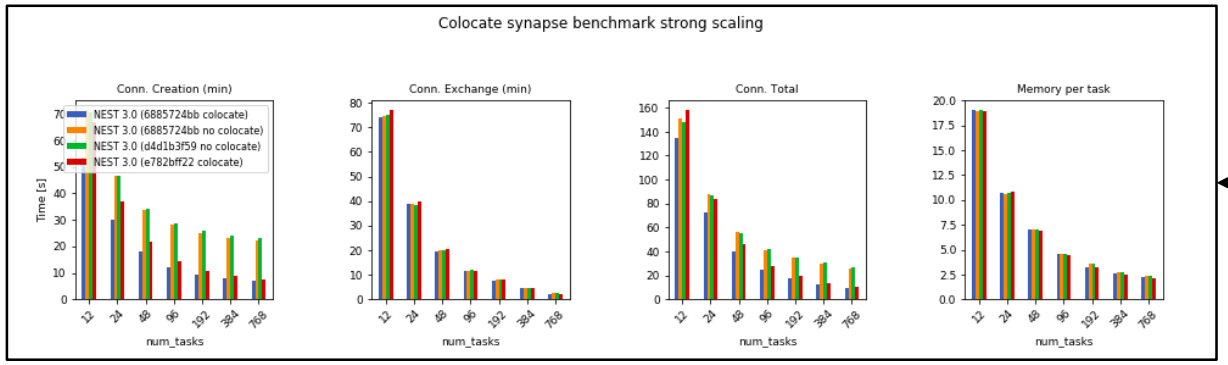
- We sent our second extension ICEI application on the 25th of August 2020
- Applied for 20 000 node x hours on Piz Daint, 10 000 node x hours On JUSUF and 10 000 node x hours at CEA
- Received allocations on the 4th of September 2020

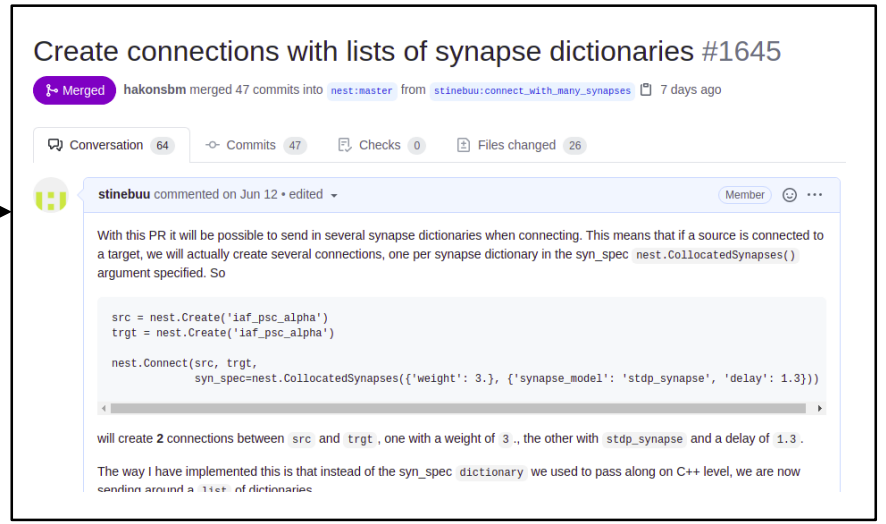
Case	NEST 3.0 (6885724bb colocate)	NEST 3.0 (6885724bb no colocate)	NEST 3.0 (d4d1b3f59 (m) no colocate)	NEST 3.0 (e782bff22 colocate)	NEST 3.0 (e782bff22 no colocate)	NEST 3.0 (6885724bb colocate)	NEST 3.0 (6885724bb no colocate)	NEST 3.0 (d4d1b3f59 (m) no colocate)
N_VP								
36	32.240653	38.559557	37.753368	40.769157	46.634900	49.301378	47.909812	46.932919
72	42.893550	57.209405	56.607433	53.521008	67.448359	59.776954	59.858176	59.697873
144	47.918999	74.364785	74.431536	59.085878	84.872910	60.748362	62.364526	60.365052
288	58.551960	108.635338	110.337347	69.862263	120.850657	62.396903	61.677782	62.177181
576	78.809734	179.284185	183.939633	91.325722	194.748234	64.226048	64.707552	63.842807
1152	120.283371	318.244408	329.092308	135.506494	338.820679	66.992121	67.864405	67.503156

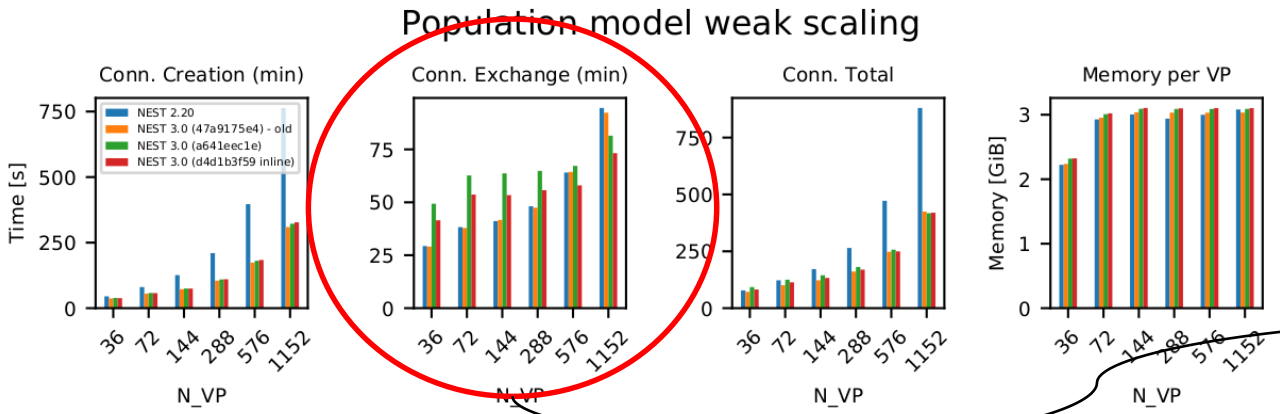
Scaling and system for third allocation

- Weak and strong scale benchmarks
- Resources on JUSUF and CEA
  - Need to find the best set-up so we utilize the systems optimally
  - Need to make sure NEST is optimized for more than one type of machine

First results





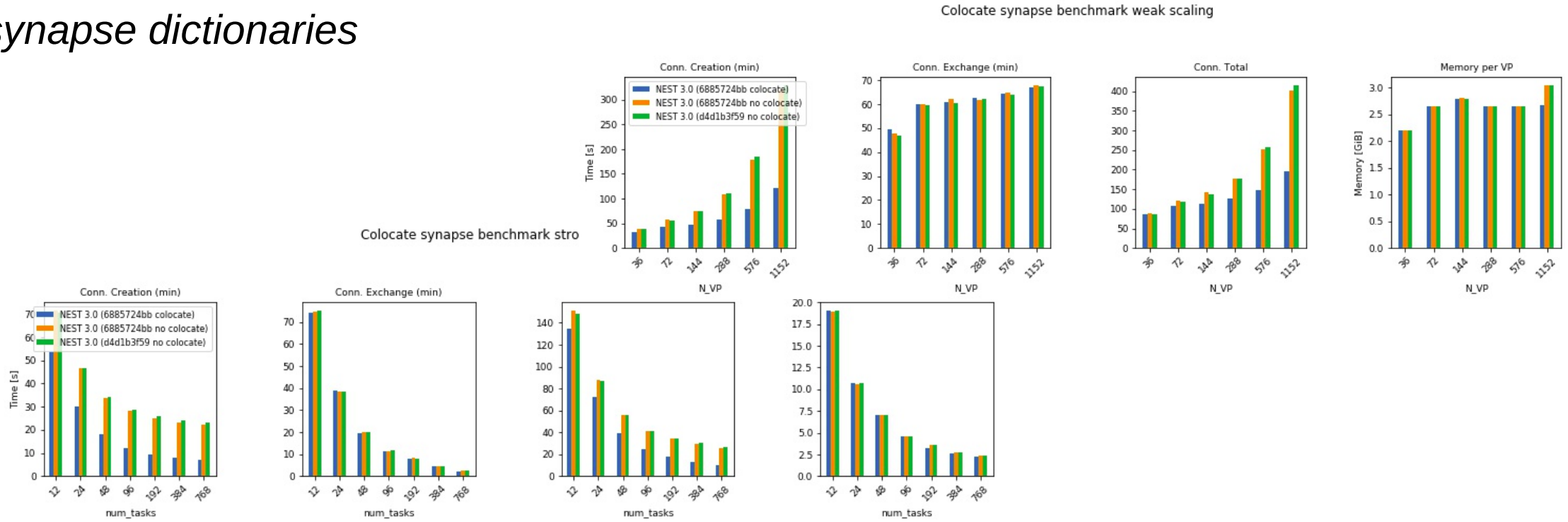


Needs further investigation!

a641eec1e: 16th of October

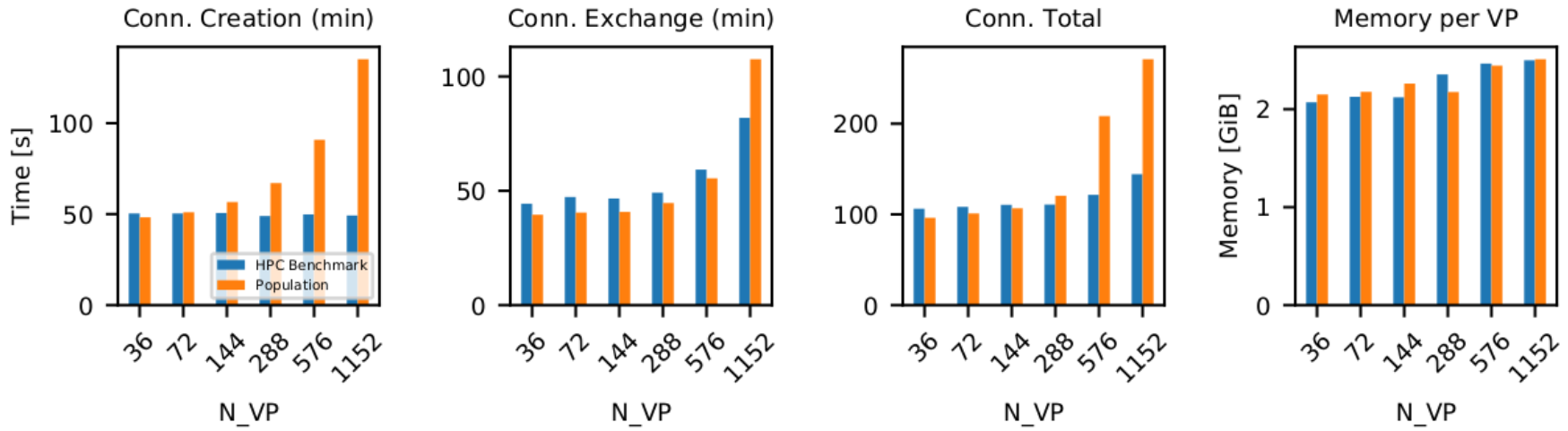
FENIX enabled PRs for third allocation period

#1645: *Create connections with lists of synapse dictionaries*



TODO

- Make the benchmark system more automated
- Set up our benchmarking system on JUSUF and CEA and start testing on multiple systems
- Working on new connectivity scheme that will need to be tested



Thank you!