RIKEN
Advanced Institute for
Computational Science

6th International Symposium

# Scalasca collaborations

2016-02-23 |

**Brian J. N. Wylie**
Jülich Supercomputing Centre

b.wylie @ fz-juelich.de

## Overview

Scalasca toolset

- collaborative development

Use of Scalasca and associated tools on K computer

- ABySS execution analysis & tuning
- NEST execution analysis

VI-HPS Tuning Workshops

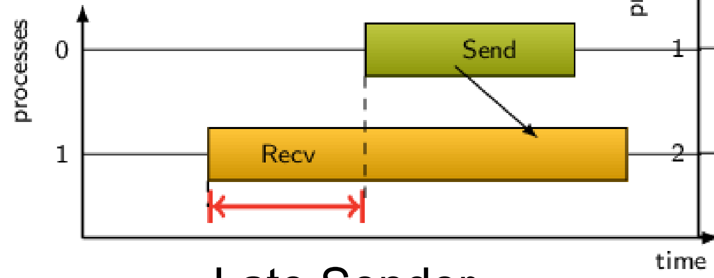
- collaborative training

## Scalasca

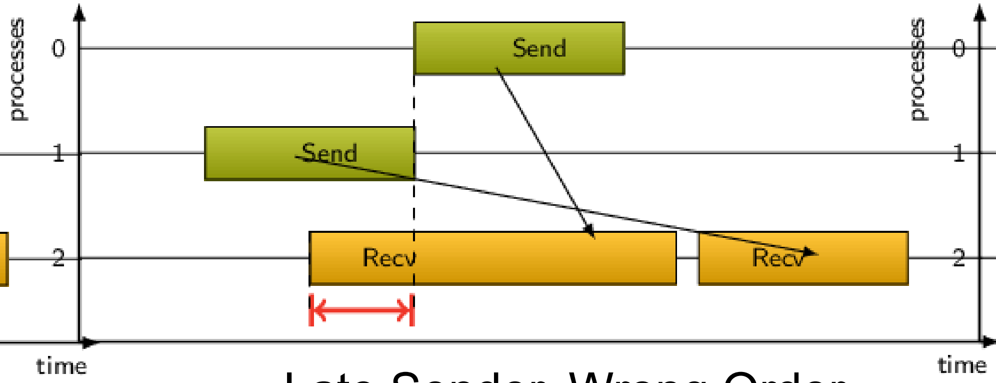
Toolset for scalable performance analysis of large-scale parallel applications

- capable of runtime summary profile and parallel event trace analyses
- for MPI, OpenMP, Pthreads and mixed MPI+threading
  - *successfully used with 1.75M threads and 1.25M processes*
- supporting most popular HPC computer systems
  - *Blue Gene/Q, Cray, Stampede, Tianhe, K computer, etc.*
- with CUBE analysis report explorer and utilities & OPARI source instrumenter components
- using libraries for PAPI hardware counters, SIONlib scalable parallel file I/O, etc.
- available under New BSD open-source license
- http://www.scalasca.org (mailto: scalasca@fz-juelich.de)

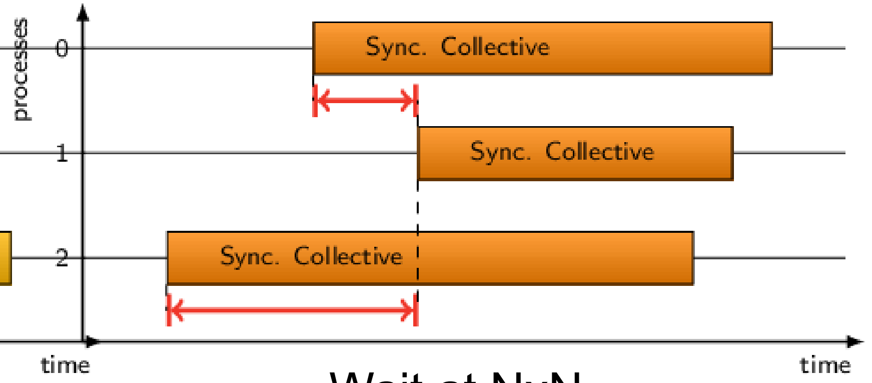
## Parallel execution inefficiency patterns



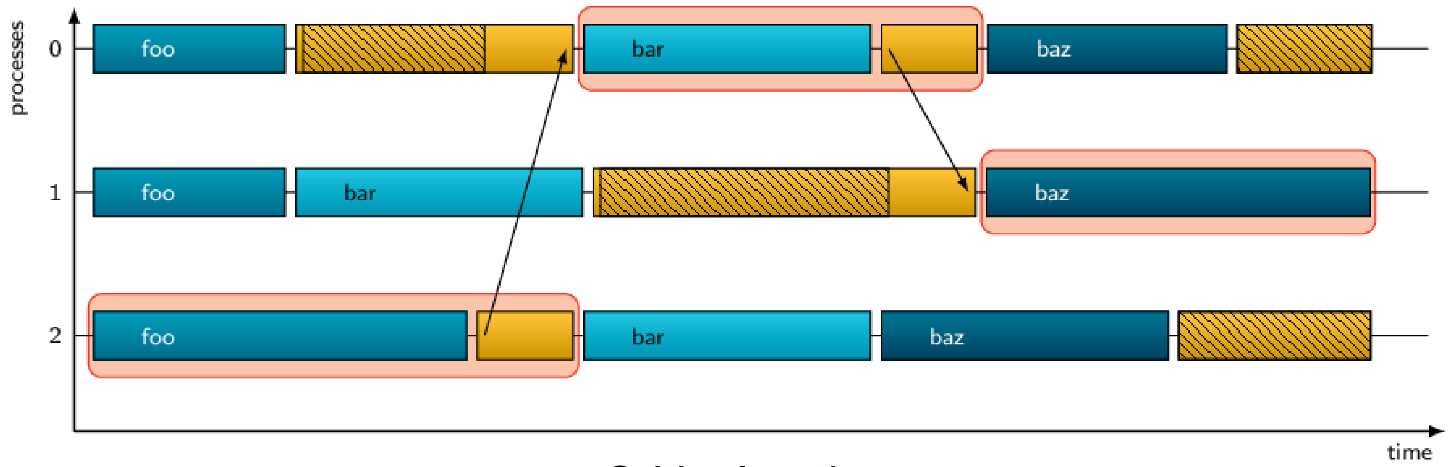
Late Sender



Late Sender, Wrong Order



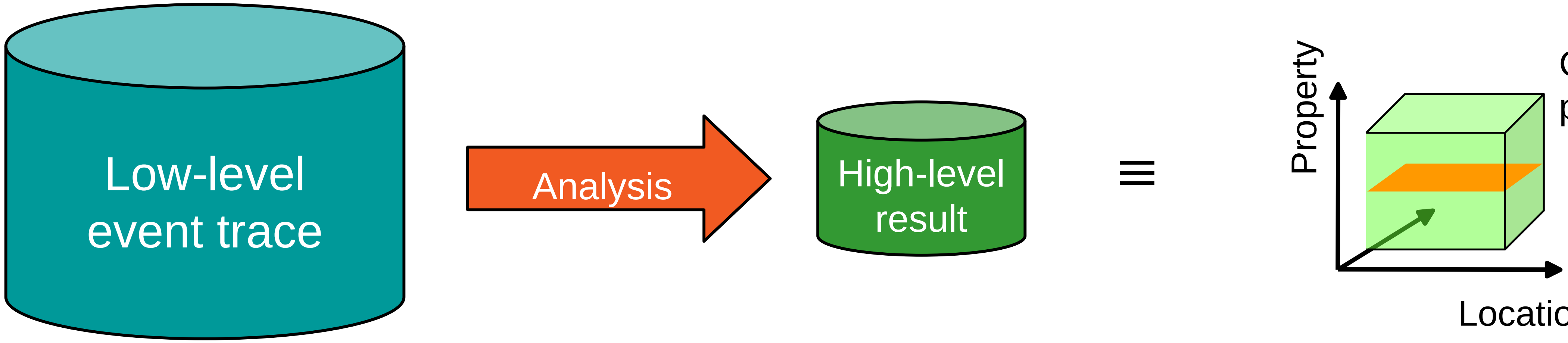
Wait at NxN



Critical path

## Automated execution trace analysis

- automatic search for patterns of inefficient execution behaviour
- classification of behaviour and quantification of significance



- quicker than manual / visual inspection, guaranteed to cover entire event trace
- parallel replay analysis exploits available memory & processors to deliver scalability

## Scalasca development

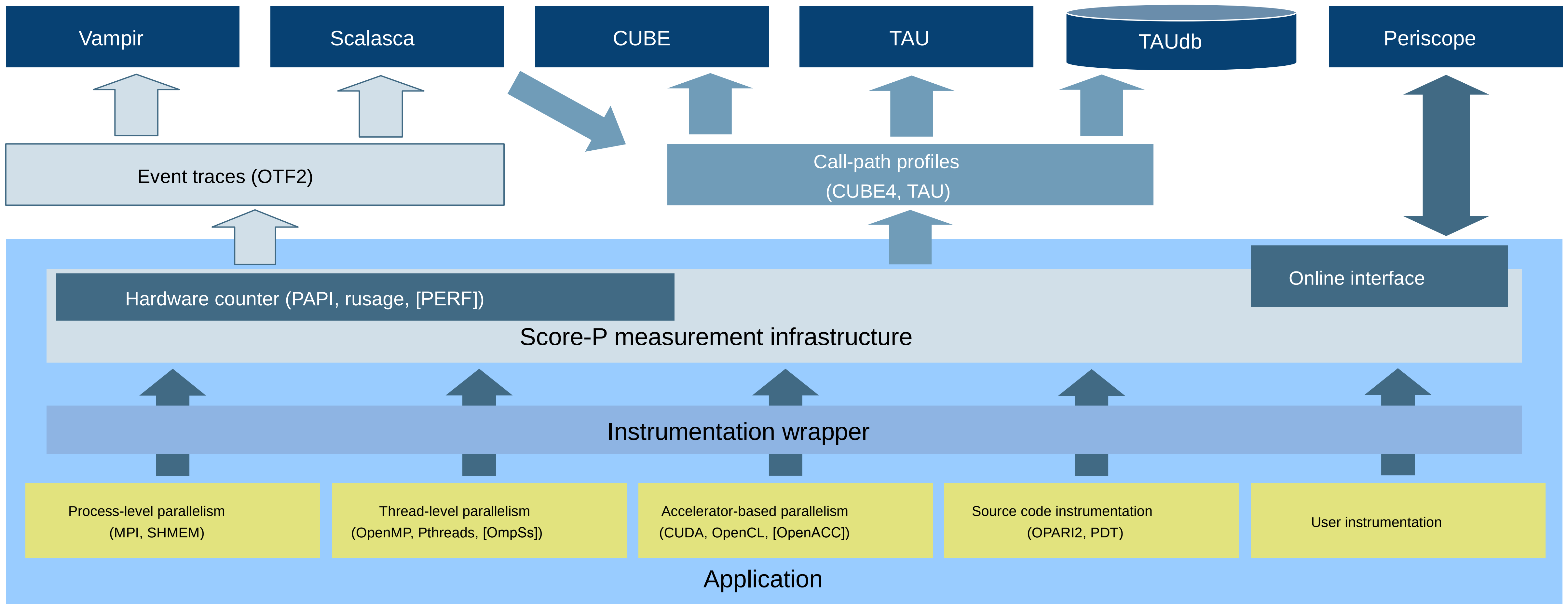
Started 2006 as scalable successor of pioneering KOJAK project (started 1998)

- initial collaboration of JSC with University of Tennessee – Knoxville
- now jointly developed with Technische Universität Darmstadt
- support for K computer and Fujitsu FX10/100 contributed by AICS

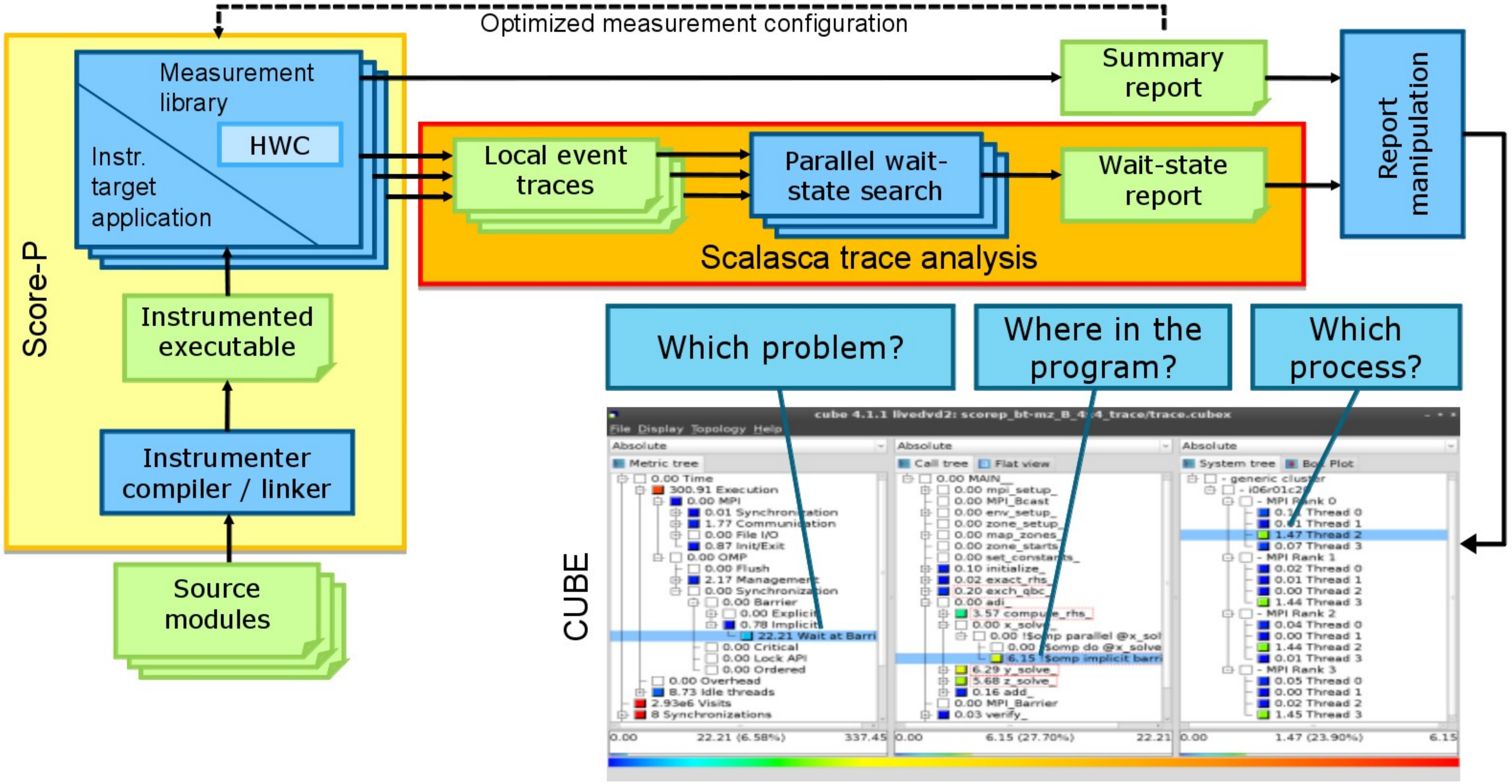
Scalasca2 using community-developed Score-P instrumentation & measurement infrastructure

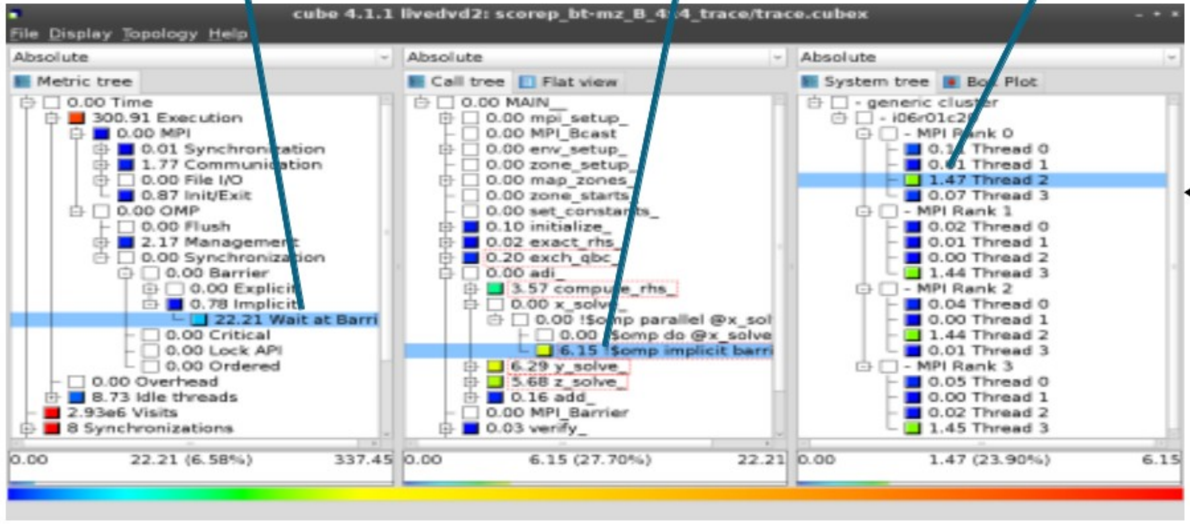
- partnership with GNS, UOregon, RWTH Aachen, TUDarmstadt, TUDresden, TUMunchen
- on-going support and contributions from Fujitsu Ltd

## Score-P architecture overview


Vampir
Scalasca
CUBE
TAU
TAUdb
Periscope
Event traces (OTF2)
Call-path profiles
(CUBE4, TAU)
Hardware counter (PAPI, rusage, [PERF])
Online interface
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism
(MPI, SHMEM)
Thread-level parallelism
(OpenMP, Pthreads, [OmpSs])
Accelerator-based parallelism
(CUDA, OpenCL, [OpenACC])
Source code instrumentation
(OPARI2, PDT)
User instrumentation
Application

## Scalasca workflow




cube 4.1.1 livedvd2: scorep_bt-mz_B_4x4_trace/trace.cubex
File Display Topology Help
Absolute
Metric tree
Call tree
Flat view
System tree
Box Plot

```
0.00 Time
  300.91 Execution
    0.00 MPI
      0.01 Synchronization
      1.77 Communication
      0.00 File I/O
      0.87 Init/Exit
    0.00 OMP
      0.00 Flush
      2.17 Management
      0.00 Synchronization
        0.00 Barrier
          0.00 Explicit
          0.78 Implicit
            22.21 Wait at Barri
        0.00 Critical
        0.00 Lock API
        0.00 Ordered
    0.00 Overhead
    8.73 Idle threads
  2.93e6 Visits
  8 Synchronizations
0.00          22.21 (6.58%)          337.45
```

```
0.00 MAIN__
  0.00 mpi_setup_
  0.00 MPI_Bcast
  0.00 env_setup_
  0.00 zone_setup_
  0.00 map_zones_
  0.00 zone_starts_
  0.00 set_constants_
  0.10 initialize_
  0.02 exact_rhs_
  0.20 exch_qbc_
  0.00 adi_
    3.57 compute_rhs_
    0.00 x_solve_
      0.00 !$omp parallel @x_sol
        0.00 !$omp do @x_solve
        6.15 !$omp implicit barri
    6.29 y_solve_
    5.68 z_solve_
    0.16 add_
  0.00 MPI_Barrier
  0.03 verify_
0.00          6.15 (27.70%)          22.21
```

```
- generic cluster
  - i06r01c20
    - MPI Rank 0
      0.11 Thread 0
      0.01 Thread 1
      1.47 Thread 2
      0.07 Thread 3
    - MPI Rank 1
      0.02 Thread 0
      0.01 Thread 1
      0.00 Thread 2
      1.44 Thread 3
    - MPI Rank 2
      0.04 Thread 0
      0.00 Thread 1
      1.44 Thread 2
      0.01 Thread 3
    - MPI Rank 3
      0.05 Thread 0
      0.00 Thread 1
      0.02 Thread 2
      1.45 Thread 3
0.00          1.47 (23.90%)          6.15
```

# Initial Scalasca performance analyses (recommended strategy)

Rebuild with automatic instrumentation: MPI library + compiler-instrumented routines

Initial runtime summary measurement scored to determine measurement filter

- identifies frequently executed short routines which may need to be eliminated
- to reduce memory requirements (mainly for event trace buffers, but also callpath profiles)
- and to reduce execution dilation of measured routines and entire application

New summary and trace measurements taken using optimised measurement filter

Event traces automatically analysed to identify and quantify MPI wait-state inefficiencies

- producing callpath profiles augmented with additional inefficiency metrics,
  and identifying severest instances

Event traces visualised with Vampir to examine execution time-lines

## ABySS genome sequence assembler

*de novo*, parallel, paired-end genome sequence assembler

- developed by Canada's Michael Smith Genome Sciences Centre

| Sequence assembly algorithm: | Stages: |
|---|---|
| ▪ load short read sequences, breaking each into *k*-mers* | LOADING |
| ▪ find adjacent (overlapping) *k*-mers | GEN_ADJ |
| ▪ remove *k*-mers resulting from read errors | ERODE / TRIM |
| ▪ remove variant sequences (bubbles) | DISCOVER / POP BUBBLES |
| ▪ generate contiguous sequences (contigs) | ASSEMBLE |

ABYSS-P de Bruijn graph assembler implemented in C++ using MPI

- master process directs remaining worker processes

* all possible (ACTG-) base subsequences of length *k* from a read obtained through DNA sequencing

## ABYSS-P scalability on K computer

Evaluation with *E. coli* K12 MG1655 read sequence dataset (6.1 GB)

Execution with single MPI rank/node

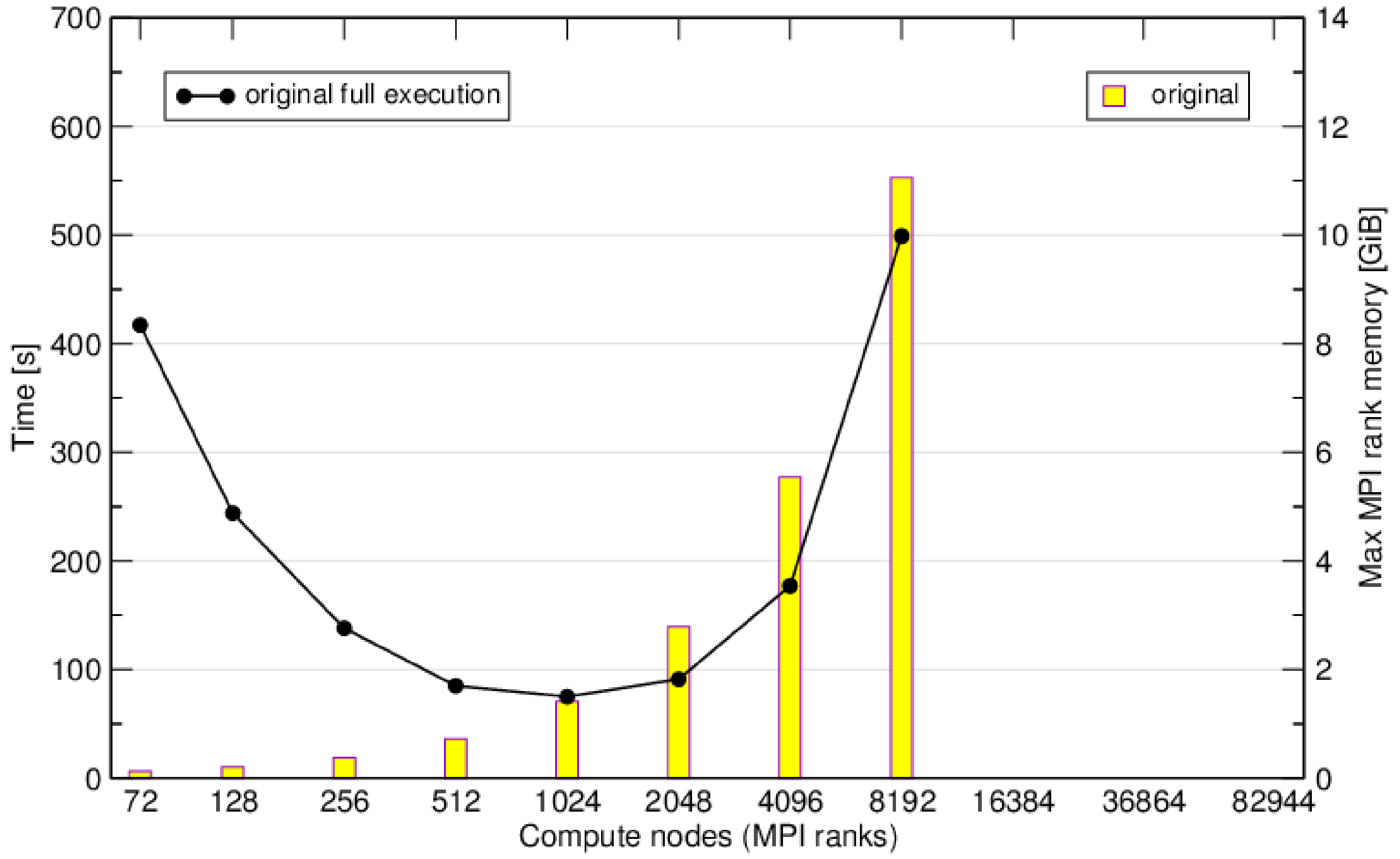
- 14 GB usable node memory

Linear growth of memory for MPI

- exhaustion for 16,384 ranks
- message buffers required for eager mode receipt

Min. execution time for ~1024 nodes

- rapid deterioration thereafter

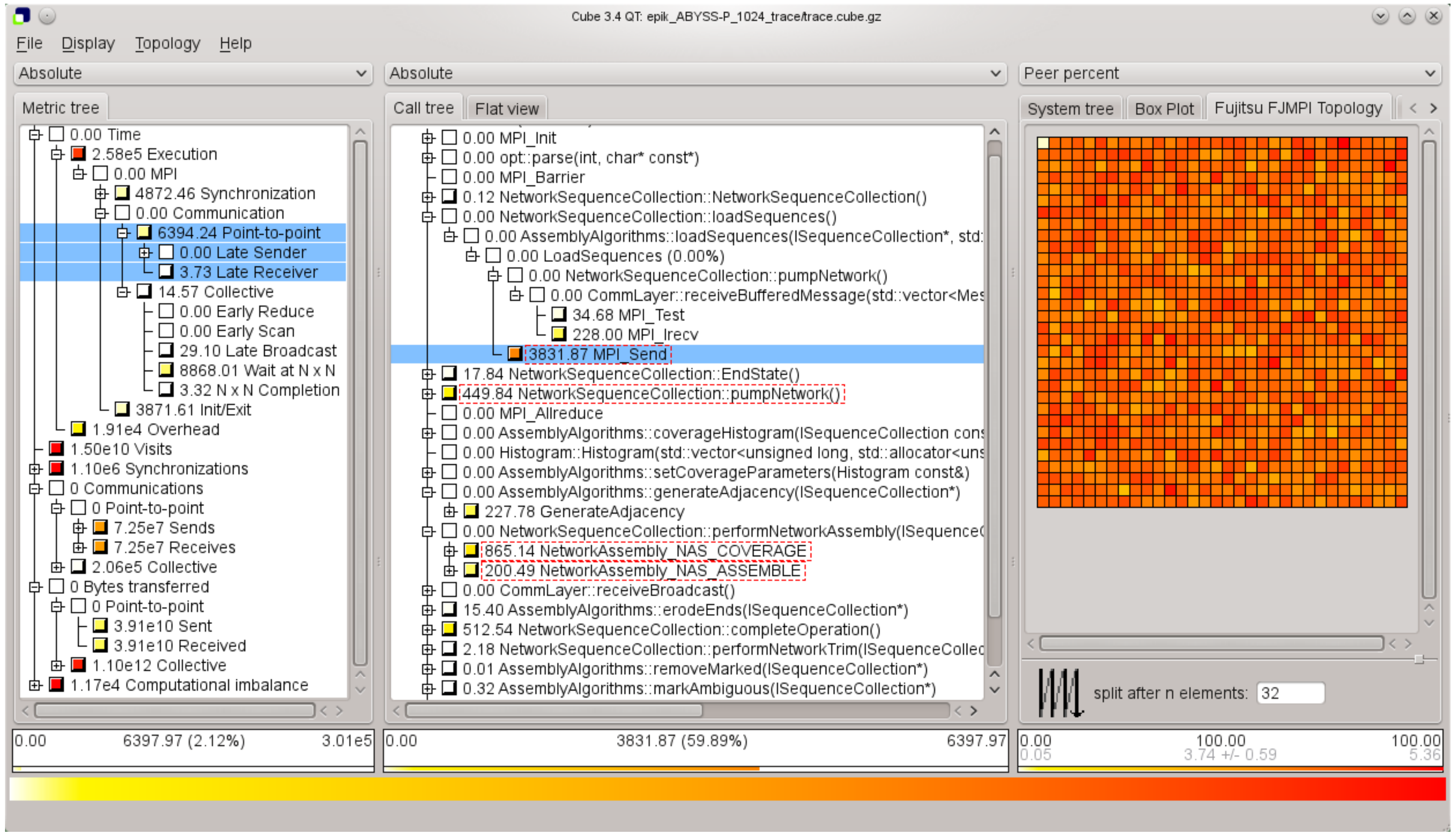


# Scalasca trace analysis (1024 MPI processes)

Callpath profile augmented with additional metrics quantifying wait states calculated in parallel replay analysis of event trace

Presentation by CUBE analysis report explorer

Tree node severity values:
- closed = inclusive,
- open = exclusive;
- coloured according to scale below pane



Cube 3.4 QT: epik_ABYSS-P_1024_trace/trace.cube.gz
File Display Topology Help
Absolute
Absolute
Peer percent
Metric tree
Call tree Flat view
System tree Box Plot Fujitsu FJMPI Topology
split after n elements: 32

```
0.00 Time
  2.58e5 Execution
    0.00 MPI
      4872.46 Synchronization
      0.00 Communication
        6394.24 Point-to-point
          0.00 Late Sender
          3.73 Late Receiver
        14.57 Collective
          0.00 Early Reduce
          0.00 Early Scan
          29.10 Late Broadcast
          8868.01 Wait at N x N
          3.32 N x N Completion
      3871.61 Init/Exit
  1.91e4 Overhead
1.50e10 Visits
1.10e6 Synchronizations
0 Communications
  0 Point-to-point
    7.25e7 Sends
    7.25e7 Receives
  2.06e5 Collective
0 Bytes transferred
  0 Point-to-point
    3.91e10 Sent
    3.91e10 Received
  1.10e12 Collective
1.17e4 Computational imbalance
```

```
0.00 MPI_Init
0.00 opt::parse(int, char* const*)
0.00 MPI_Barrier
0.12 NetworkSequenceCollection::NetworkSequenceCollection()
0.00 NetworkSequenceCollection::loadSequences()
  0.00 AssemblyAlgorithms::loadSequences(ISequenceCollection*, std:
    0.00 LoadSequences (0.00%)
      0.00 NetworkSequenceCollection::pumpNetwork()
        0.00 CommLayer::receiveBufferedMessage(std::vector<Mes
          34.68 MPI_Test
          228.00 MPI_Irecv
      3831.87 MPI_Send
17.84 NetworkSequenceCollection::EndState()
449.84 NetworkSequenceCollection::pumpNetwork()
0.00 MPI_Allreduce
0.00 AssemblyAlgorithms::coverageHistogram(ISequenceCollection cons
0.00 Histogram::Histogram(std::vector<unsigned long, std::allocator<uns
0.00 AssemblyAlgorithms::setCoverageParameters(Histogram const&)
0.00 AssemblyAlgorithms::generateAdjacency(ISequenceCollection*)
  227.78 GenerateAdjacency
0.00 NetworkSequenceCollection::performNetworkAssembly(ISequence(
  865.14 NetworkAssembly_NAS_COVERAGE
  200.49 NetworkAssembly_NAS_ASSEMBLE
0.00 CommLayer::receiveBroadcast()
15.40 AssemblyAlgorithms::erodeEnds(ISequenceCollection*)
512.54 NetworkSequenceCollection::completeOperation()
2.18 NetworkSequenceCollection::performNetworkTrim(ISequenceCollec
0.01 AssemblyAlgorithms::removeMarked(ISequenceCollection*)
0.32 AssemblyAlgorithms::markAmbiguous(ISequenceCollection*)
```

```
0.00        6397.97 (2.12%)      3.01e5
0.00        3831.87 (59.89%)     6397.97
0.00        100.00               100.00
0.05        3.74 +/- 0.59        5.36
```

Hierarchies of metrics

Tree of execution callpaths (values for selected metric)

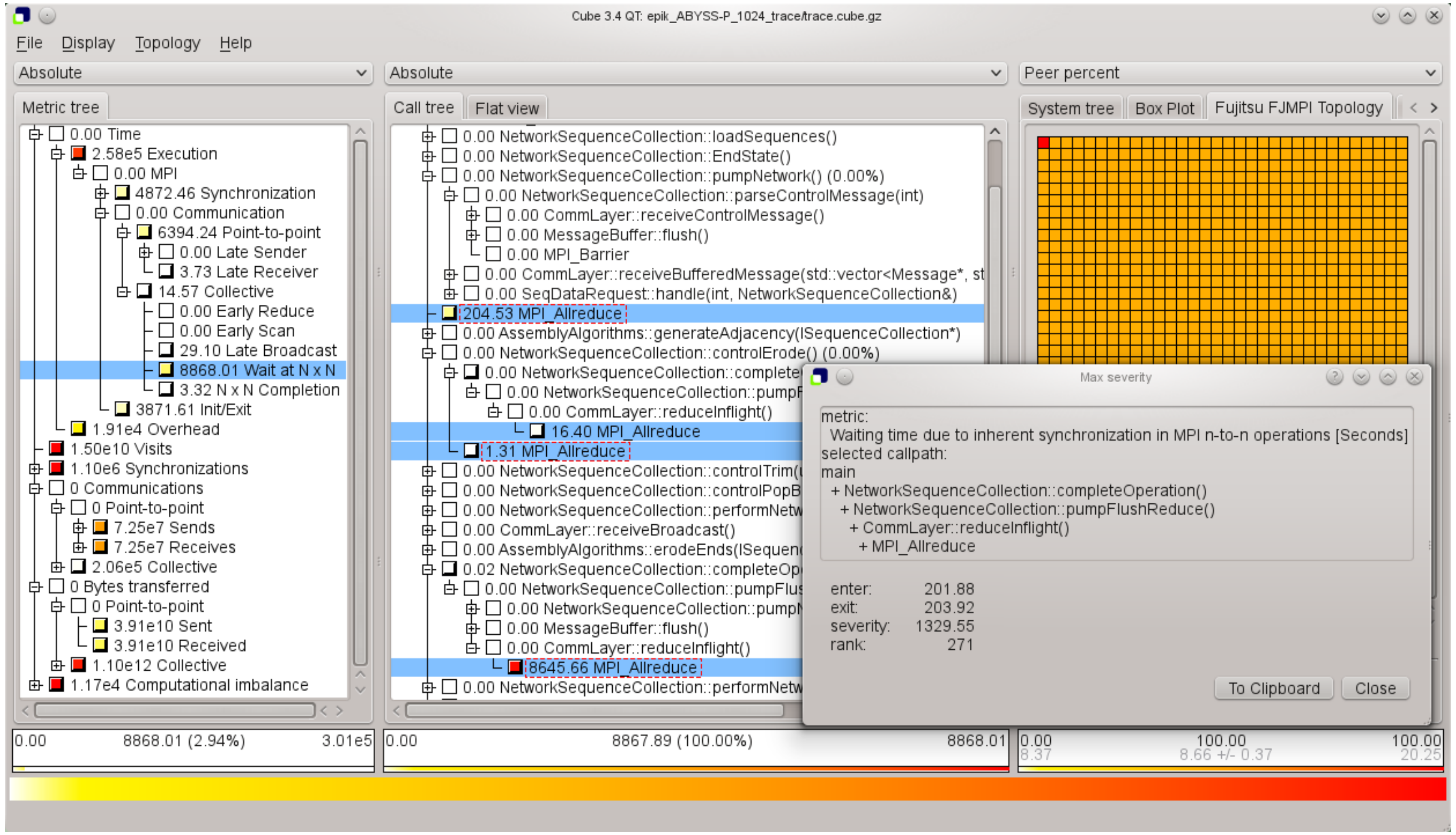
Distribution of MPI rank severity values (rank 0 top-left)

## Scalasca trace analysis identifying severest instance of Wait at NxN in MPI_Allreduce

Lots of point-to-point Send and Irecv communication, with negligible waiting time

Comparatively large waiting time in MPI collectives (Allreduce, Barrier, etc.)

Severest waiting instances averaging 1.3 seconds and with maximum durations of over 2 seconds!


Cube 3.4 QT: epik_ABYSS-P_1024_trace/trace.cube.gz

File Display Topology Help

Absolute | Absolute | Peer percent

Metric tree

```
0.00 Time
  2.58e5 Execution
    0.00 MPI
      4872.46 Synchronization
      0.00 Communication
        6394.24 Point-to-point
          0.00 Late Sender
          3.73 Late Receiver
        14.57 Collective
          0.00 Early Reduce
          0.00 Early Scan
          29.10 Late Broadcast
          8868.01 Wait at N x N
          3.32 N x N Completion
      3871.61 Init/Exit
  1.91e4 Overhead
1.50e10 Visits
1.10e6 Synchronizations
0 Communications
  0 Point-to-point
    7.25e7 Sends
    7.25e7 Receives
  2.06e5 Collective
0 Bytes transferred
  0 Point-to-point
    3.91e10 Sent
    3.91e10 Received
  1.10e12 Collective
1.17e4 Computational imbalance
```

```
0.00          8868.01 (2.94%)          3.01e5
```

Call tree | Flat view

```
0.00 NetworkSequenceCollection::loadSequences()
0.00 NetworkSequenceCollection::EndState()
0.00 NetworkSequenceCollection::pumpNetwork() (0.00%)
  0.00 NetworkSequenceCollection::parseControlMessage(int)
    0.00 CommLayer::receiveControlMessage()
    0.00 MessageBuffer::flush()
    0.00 MPI_Barrier
  0.00 CommLayer::receiveBufferedMessage(std::vector<Message*, st
  0.00 SeqDataRequest::handle(int, NetworkSequenceCollection&)
204.53 MPI_Allreduce
0.00 AssemblyAlgorithms::generateAdjacency(ISequenceCollection*)
0.00 NetworkSequenceCollection::controlErode() (0.00%)
  0.00 NetworkSequenceCollection::complete
    0.00 NetworkSequenceCollection::pumpF
      0.00 CommLayer::reduceInflight()
        16.40 MPI_Allreduce
  1.31 MPI_Allreduce
0.00 NetworkSequenceCollection::controlTrim(
0.00 NetworkSequenceCollection::controlPopB
0.00 NetworkSequenceCollection::performNetw
0.00 CommLayer::receiveBroadcast()
0.00 AssemblyAlgorithms::erodeEnds(ISequen
0.02 NetworkSequenceCollection::completeOp
  0.00 NetworkSequenceCollection::pumpFlus
    0.00 NetworkSequenceCollection::pumpN
    0.00 MessageBuffer::flush()
    0.00 CommLayer::reduceInflight()
      8645.66 MPI_Allreduce
0.00 NetworkSequenceCollection::performNetw
```

```
0.00          8867.89 (100.00%)          8868.01
```

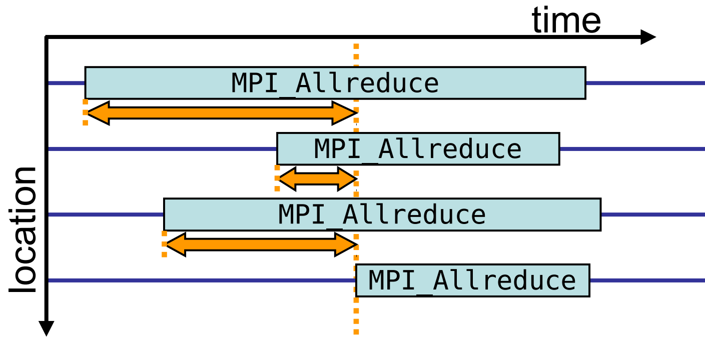
System tree | Box Plot | Fujitsu FJMPI Topology

```
0.00          100.00          100.00
8.37      8.66 +/- 0.37        20.25
```

Max severity

```
metric:
 Waiting time due to inherent synchronization in MPI n-to-n operations [Seconds]
selected callpath:
main
 + NetworkSequenceCollection::completeOperation()
  + NetworkSequenceCollection::pumpFlushReduce()
   + CommLayer::reduceInflight()
    + MPI_Allreduce

 enter:       201.88
 exit:        203.92
 severity:   1329.55
 rank:           271
```

To Clipboard | Close



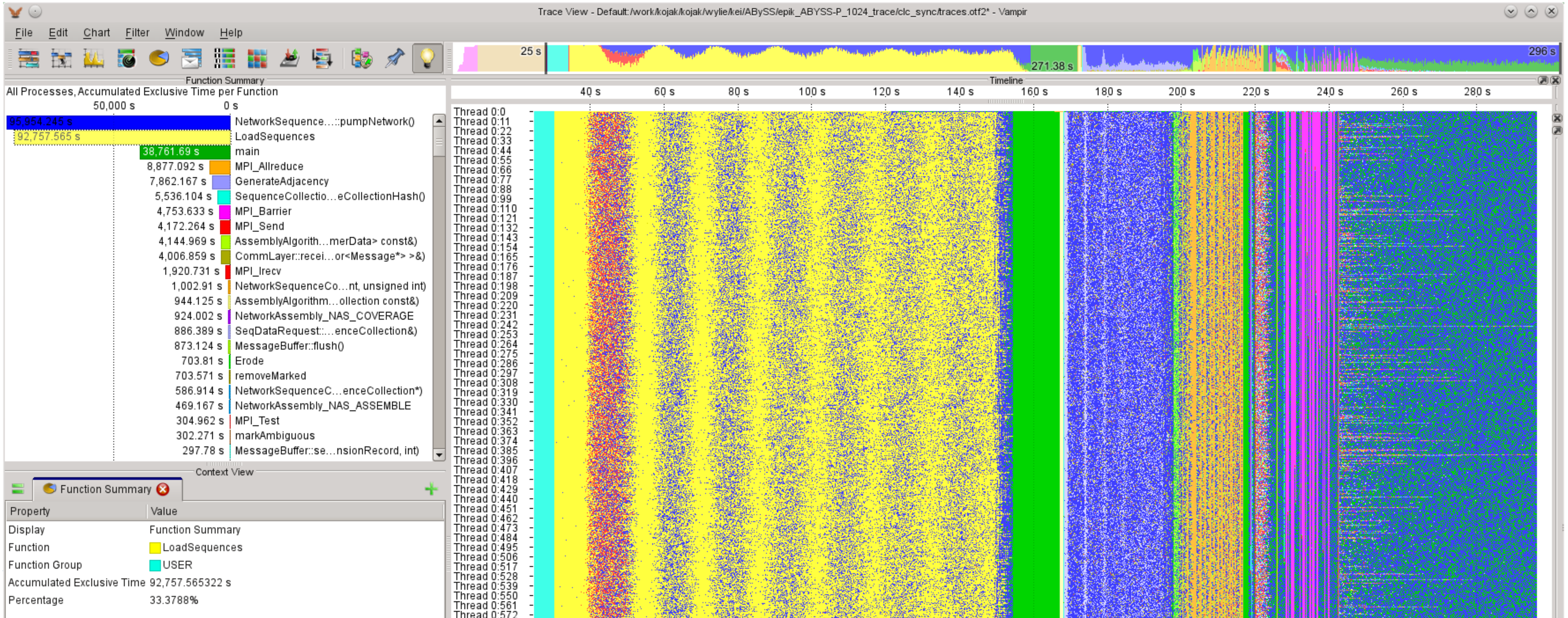
- generally indicative of load-imbalance in preceding computation, however, distribution pattern seems uniform across processes

# Vampir visualisation of same event trace

Distinguished phases of LOADING, ERODE/TRIM, ASSEMBLE with “waves” of communication evident within LOADING stage

Graphical (flat) function profile for current time interval

Horizontal time-line of coloured execution states for each MPI process below aggregate state chart provided for navigation when zooming



All Processes, Accumulated Exclusive Time per Function

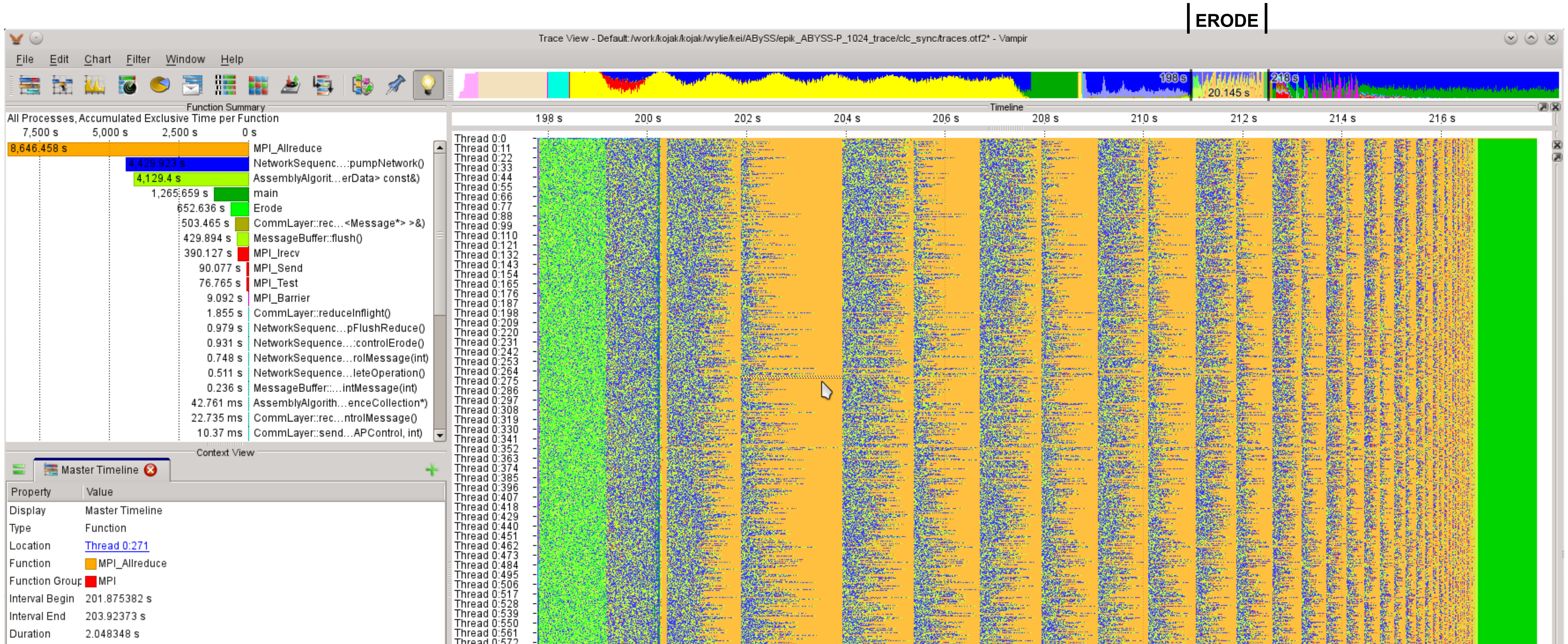
50,000 s    0 s

| | |
|---|---|
| 95,954.245 s | NetworkSequence...::pumpNetwork() |
| 92,757.565 s | LoadSequences |
| 38,761.69 s | main |
| 8,877.092 s | MPI_Allreduce |
| 7,862.167 s | GenerateAdjacency |
| 5,536.104 s | SequenceCollectio...eCollectionHash() |
| 4,753.633 s | MPI_Barrier |
| 4,172.264 s | MPI_Send |
| 4,144.969 s | AssemblyAlgorith...merData> const&) |
| 4,006.859 s | CommLayer::recei...or<Message*> >&) |
| 1,920.731 s | MPI_Irecv |
| 1,002.91 s | NetworkSequenceCo...nt, unsigned int) |
| 944.125 s | AssemblyAlgorithm...ollection const&) |
| 924.002 s | NetworkAssembly_NAS_COVERAGE |
| 886.389 s | SeqDataRequest::...enceCollection&) |
| 873.124 s | MessageBuffer::flush() |
| 703.81 s | Erode |
| 703.571 s | removeMarked |
| 586.914 s | NetworkSequenceC...enceCollection*) |
| 469.167 s | NetworkAssembly_NAS_ASSEMBLE |
| 304.962 s | MPI_Test |
| 302.271 s | markAmbiguous |
| 297.78 s | MessageBuffer::se...nsionRecord, int) |

| Property | Value |
|---|---|
| Display | Function Summary |
| Function | LoadSequences |
| Function Group | USER |
| Accumulated Exclusive Time | 92,757.565322 s |
| Percentage | 33.3788% |

## Vampir display zoomed to time interval of ERODE stage

Sequence of MPI_Allreduce calls within ERODE stage many of the initial ones with huge waiting times (over 2 seconds duration for severest instance on rank 271) resulting from load-imbalance in preceding computation – alternating distribution of severities



Function Summary

All Processes, Accumulated Exclusive Time per Function

| | |
|---|---|
| 8,646.458 s | MPI_Allreduce |
| 4,429.923 s | NetworkSequenc...:pumpNetwork() |
| 4,129.4 s | AssemblyAlgorit...erData> const&) |
| 1,265.659 s | main |
| 652.636 s | Erode |
| 503.465 s | CommLayer::rec...<Message*> >&) |
| 429.894 s | MessageBuffer::flush() |
| 390.127 s | MPI_Irecv |
| 90.077 s | MPI_Send |
| 76.765 s | MPI_Test |
| 9.092 s | MPI_Barrier |
| 1.855 s | CommLayer::reduceInflight() |
| 0.979 s | NetworkSequenc...pFlushReduce() |
| 0.931 s | NetworkSequence...:controlErode() |
| 0.748 s | NetworkSequence...rolMessage(int) |
| 0.511 s | NetworkSequence...leteOperation() |
| 0.236 s | MessageBuffer::...intMessage(int) |
| 42.761 ms | AssemblyAlgorith...enceCollection*) |
| 22.735 ms | CommLayer::rec...ntrolMessage() |
| 10.37 ms | CommLayer::send...APControl, int) |

Context View

| Property | Value |
|---|---|
| Display | Master Timeline |
| Type | Function |
| Location | Thread 0:271 |
| Function | 🟧 MPI_Allreduce |
| Function Group | 🟥 MPI |
| Interval Begin | 201.875382 s |
| Interval End | 203.92373 s |
| Duration | 2.048348 s |

## Assessment of initial ABYSS-P performance analyses with Scalasca

Positive insights

- Complex Master-Worker execution with asynchronous message/work queues
- Complex usage of MPI P2P and collective communication that varies by execution stage
- Increasingly costly communication and load-balancing inefficiencies at larger scale
- Extreme run-to-run execution time variations (likely mostly due to file I/O)

Negative insights

- Compiler-instrumentation prohibitively intrusive (even with extensive filtering)
- Difficult to determine effective filter, due to complex (dual-role) code structure
- Difficult to correlate Master directions with Worker activities
- Difficult to distinguish important execution stages
- Difficult to isolate origin of imbalances, e.g., file I/O

## Revised performance analysis strategy

MPI-only measurements (without compiler instrumentation) don't suffer intrusion

- however, lack of context makes analyses even more difficult to interpret, and traces are still huge

Therefore, manually annotate ABYSS-P sources with user-instrumentation macros

- to distinguish execution stages and other activities of interest (such as file I/O)

Take execution measurements combining MPI + user instrumentation

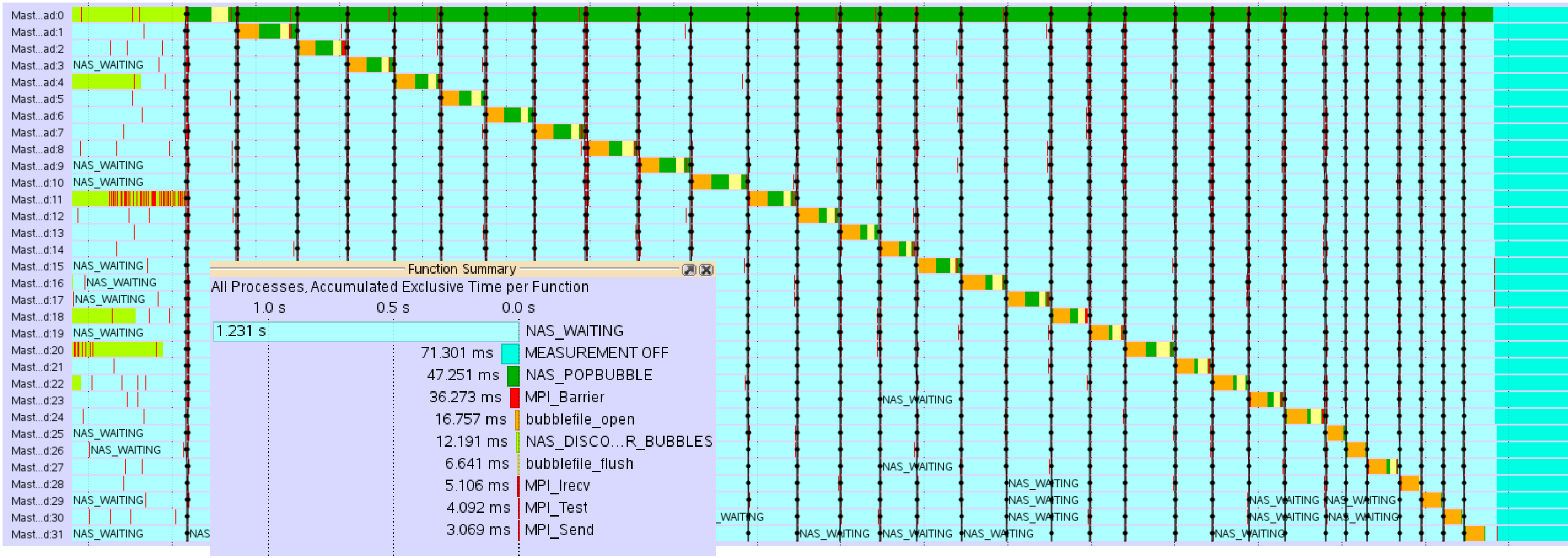
- enable trace collection only for selected execution stage(s) to reduce size

Compare alternative ABYSS-P implementations and execution configurations

- quantify how performance of stages varies with scale

## DISCOVER / POP BUBBLES stage

Each rank in turn executes POPBUBBLE while all others wait, where in addition to local computation POPBUBBLE creates, writes & flushes a separate file before notifying all of its peers



Function Summary

All Processes, Accumulated Exclusive Time per Function

```
1.0 s              0.5 s              0.0 s
1.231 s                                   NAS_WAITING
                  71.301 ms  MEASUREMENT OFF
                  47.251 ms  NAS_POPBUBBLE
                  36.273 ms  MPI_Barrier
                  16.757 ms  bubblefile_open
                  12.191 ms  NAS_DISCO...R_BUBBLES
                   6.641 ms  bubblefile_flush
                   5.106 ms  MPI_Irecv
                   4.092 ms  MPI_Test
                   3.069 ms  MPI_Send
```

## ABYSS-P tuning for K computer

Use separate rank-local directories for files

- avoids highly variable file-system contention for files in common directory

Open (create) files in parallel, in advance of writes

- removed from serialisation within POP_BUBBLE stage
- reduces disruption of computation within ASSEMBLE stage

Employ collective communication instead of eager point-to-point communication

- avoids linearly growing MPI memory requirements, and bottleneck for Master rank 0
- exploits MPI_Reduce and MPI_Alltoall optimised for K computer

## ABYSS-P profile comparison: (original) (improved)

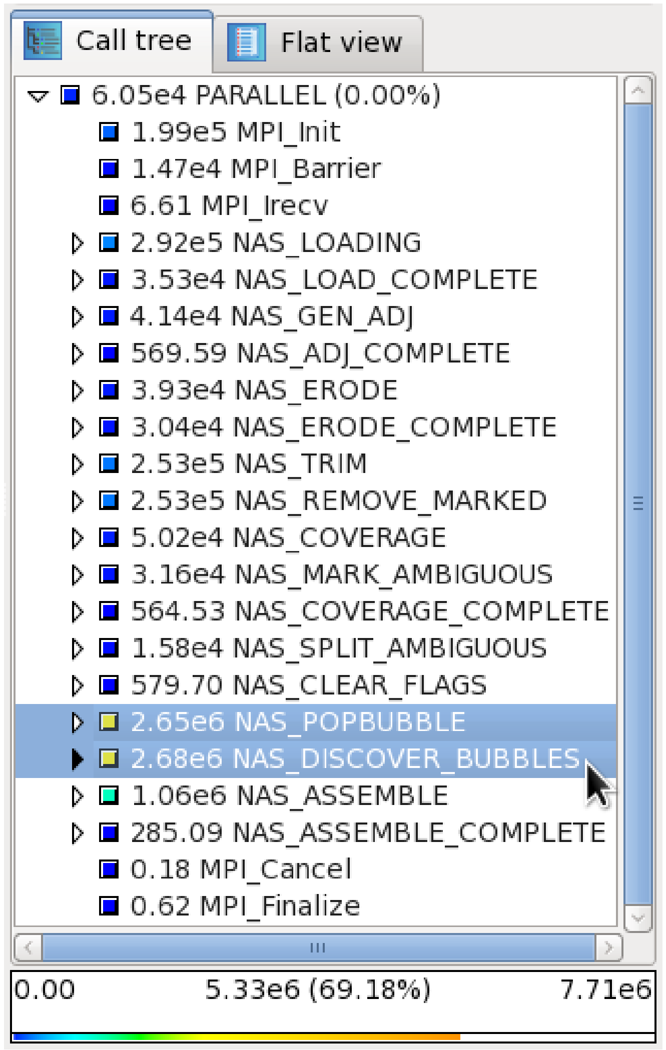
8192 MPI processes on K computer

Original version 941 seconds

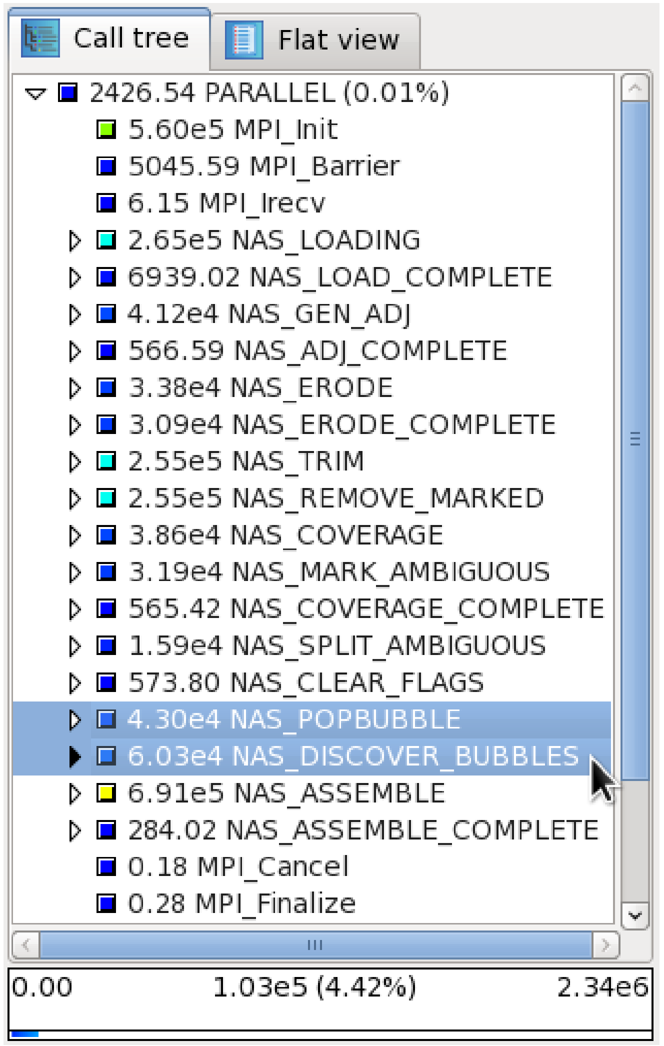
- 69% (651s) in coupled POP/DISCOVER_BUBBLES
- 14% (129s) in ASSEMBLE
- 4.2% (45s) in LOADING

Improved version 293 seconds

- 4.4% (13s) in coupled POP/DISCOVER_BUBBLES
- 30% (84s) in ASSEMBLE
- 12% (33s) in LOADING



```
▽ 6.05e4 PARALLEL (0.00%)
    1.99e5 MPI_Init
    1.47e4 MPI_Barrier
    6.61 MPI_Irecv
  ▷ 2.92e5 NAS_LOADING
  ▷ 3.53e4 NAS_LOAD_COMPLETE
  ▷ 4.14e4 NAS_GEN_ADJ
  ▷ 569.59 NAS_ADJ_COMPLETE
  ▷ 3.93e4 NAS_ERODE
  ▷ 3.04e4 NAS_ERODE_COMPLETE
  ▷ 2.53e5 NAS_TRIM
  ▷ 2.53e5 NAS_REMOVE_MARKED
  ▷ 5.02e4 NAS_COVERAGE
  ▷ 3.16e4 NAS_MARK_AMBIGUOUS
  ▷ 564.53 NAS_COVERAGE_COMPLETE
  ▷ 1.58e4 NAS_SPLIT_AMBIGUOUS
  ▷ 579.70 NAS_CLEAR_FLAGS
  ▷ 2.65e6 NAS_POPBUBBLE
  ▶ 2.68e6 NAS_DISCOVER_BUBBLES
  ▷ 1.06e6 NAS_ASSEMBLE
  ▷ 285.09 NAS_ASSEMBLE_COMPLETE
    0.18 MPI_Cancel
    0.62 MPI_Finalize

0.00          5.33e6 (69.18%)          7.71e6
```



```
▽ 2426.54 PARALLEL (0.01%)
    5.60e5 MPI_Init
    5045.59 MPI_Barrier
    6.15 MPI_Irecv
  ▷ 2.65e5 NAS_LOADING
  ▷ 6939.02 NAS_LOAD_COMPLETE
  ▷ 4.12e4 NAS_GEN_ADJ
  ▷ 566.59 NAS_ADJ_COMPLETE
  ▷ 3.38e4 NAS_ERODE
  ▷ 3.09e4 NAS_ERODE_COMPLETE
  ▷ 2.55e5 NAS_TRIM
  ▷ 2.55e5 NAS_REMOVE_MARKED
  ▷ 3.86e4 NAS_COVERAGE
  ▷ 3.19e4 NAS_MARK_AMBIGUOUS
  ▷ 565.42 NAS_COVERAGE_COMPLETE
  ▷ 1.59e4 NAS_SPLIT_AMBIGUOUS
  ▷ 573.80 NAS_CLEAR_FLAGS
  ▷ 4.30e4 NAS_POPBUBBLE
  ▶ 6.03e4 NAS_DISCOVER_BUBBLES
  ▷ 6.91e5 NAS_ASSEMBLE
  ▷ 284.02 NAS_ASSEMBLE_COMPLETE
    0.18 MPI_Cancel
    0.28 MPI_Finalize

0.00          1.03e5 (4.42%)          2.34e6
```

## ABYSS-P scalability on K computer (comparison)

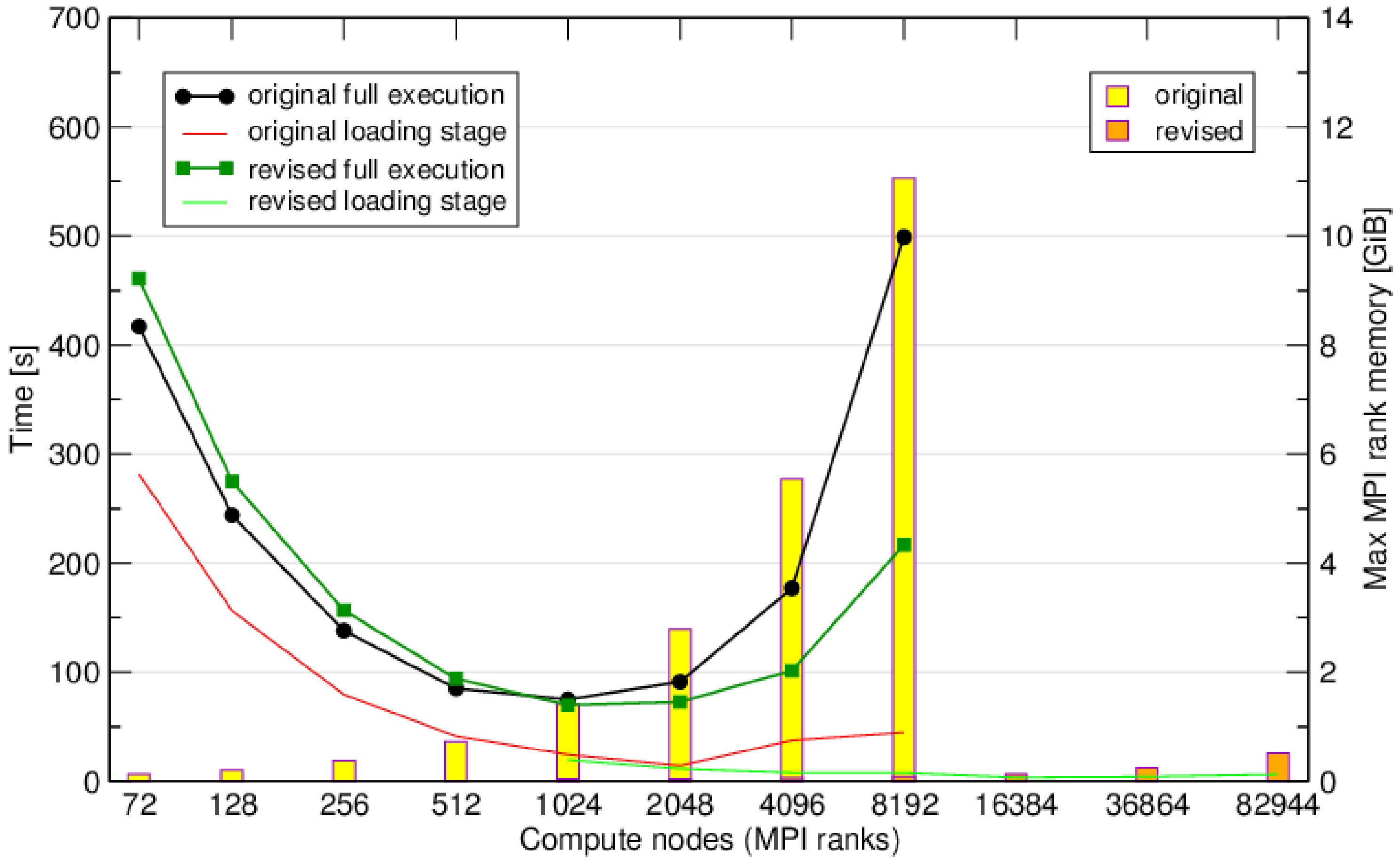
Improved execution time at scale

- ASSEMBLE now dominates
- DISCOVER/POP BUBBLES 50 times faster for 8,192
- LOADING scales well and reduced to a few seconds

Much less memory required for MPI

- only 0.5 GB for 82,944 ranks

Rework of Master-Worker coordination communication needed for remaining ABYSS-P stages



## ABySS execution analysis & tuning review

ABYSS-P executions on the K computer suffered critical performance and scaling issues

- slowdown with more than 1024 compute nodes, excessive MPI memory requirements

Analysis using Scalasca/Score-P/Vampir helped identify & quantify execution inefficiencies

- file I/O variability, serial file creation, rank 0 coordination bottleneck, ...
- automatic instrumentation provided a convenient starting point,
  but needed to be complemented with manual instrumentation
  - *Master/Worker execution stages and file I/O*

ABYSS-P remediation

- rank-local directories, reorganised file handling, exploiting efficient MPI collectives
- execution (and Scalasca measurement) now possible on full 82,944 compute nodes
  - *only initial execution stages completed*

## NEST neural simulation tool

Neuronal spiking network simulation tool [www.nest-simulator.org]

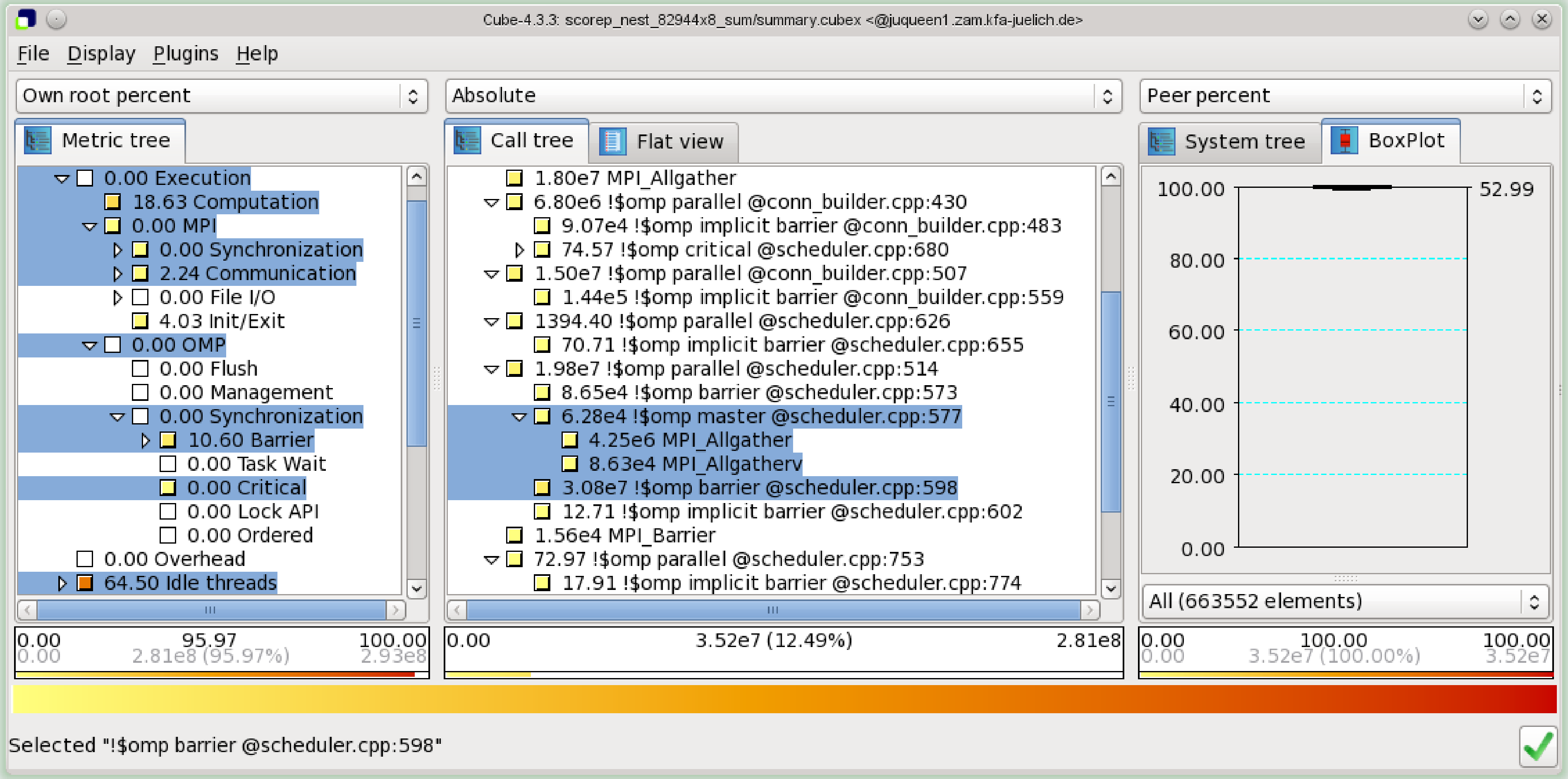
- C++ implementation using MPI and OpenMP parallelisation
- neurons distributed to MPI processes
- synapses and associated connection information stored on post-synaptic process
- exchange during simulation update based on MPI_Allgather(v) by master threads

Experiments

- empty-gap random connectivity simulation on K computer
  - *gather_events performance governed by MPI&OMP computational load imbalances*
- HDF5 import module for large-scale data-driven simulation on JUQUEEN BG/Q
  - *1.9 TB connectivity map from high-resolution biological data*
  - *failure to use collective MPI File I/O significantly diminishes reading performance*

# NEST 82944x8 measurement on K computer: simulation gather_events

53s to gather spiking events from firing synapses



Cube-4.3.3: scorep_nest_82944x8_sum/summary.cubex <@juqueen1.zam.kfa-juelich.de>

File Display Plugins Help

Own root percent | Absolute | Peer percent

Metric tree:

```
▽ ☐ 0.00 Execution
    ■ 18.63 Computation
  ▽ ■ 0.00 MPI
    ▷ ■ 0.00 Synchronization
    ▷ ■ 2.24 Communication
    ▷ ☐ 0.00 File I/O
      ■ 4.03 Init/Exit
  ▽ ☐ 0.00 OMP
      ☐ 0.00 Flush
      ☐ 0.00 Management
    ▽ ☐ 0.00 Synchronization
      ▷ ■ 10.60 Barrier
        ☐ 0.00 Task Wait
        ■ 0.00 Critical
        ☐ 0.00 Lock API
        ☐ 0.00 Ordered
  ☐ 0.00 Overhead
▷ ■ 64.50 Idle threads
```

0.00 | 95.97 | 100.00
0.00 | 2.81e8 (95.97%) | 2.93e8

Call tree / Flat view:

```
  ■ 1.80e7 MPI_Allgather
▽ ■ 6.80e6 !$omp parallel @conn_builder.cpp:430
    ■ 9.07e4 !$omp implicit barrier @conn_builder.cpp:483
  ▷ ■ 74.57 !$omp critical @scheduler.cpp:680
▽ ■ 1.50e7 !$omp parallel @conn_builder.cpp:507
    ■ 1.44e5 !$omp implicit barrier @conn_builder.cpp:559
▽ ■ 1394.40 !$omp parallel @scheduler.cpp:626
    ■ 70.71 !$omp implicit barrier @scheduler.cpp:655
▽ ■ 1.98e7 !$omp parallel @scheduler.cpp:514
    ■ 8.65e4 !$omp barrier @scheduler.cpp:573
  ▽ ■ 6.28e4 !$omp master @scheduler.cpp:577
      ■ 4.25e6 MPI_Allgather
      ■ 8.63e4 MPI_Allgatherv
    ■ 3.08e7 !$omp barrier @scheduler.cpp:598
    ■ 12.71 !$omp implicit barrier @scheduler.cpp:602
  ■ 1.56e4 MPI_Barrier
▽ ■ 72.97 !$omp parallel @scheduler.cpp:753
    ■ 17.91 !$omp implicit barrier @scheduler.cpp:774
```

0.00 | 3.52e7 (12.49%) | 2.81e8

System tree / BoxPlot:

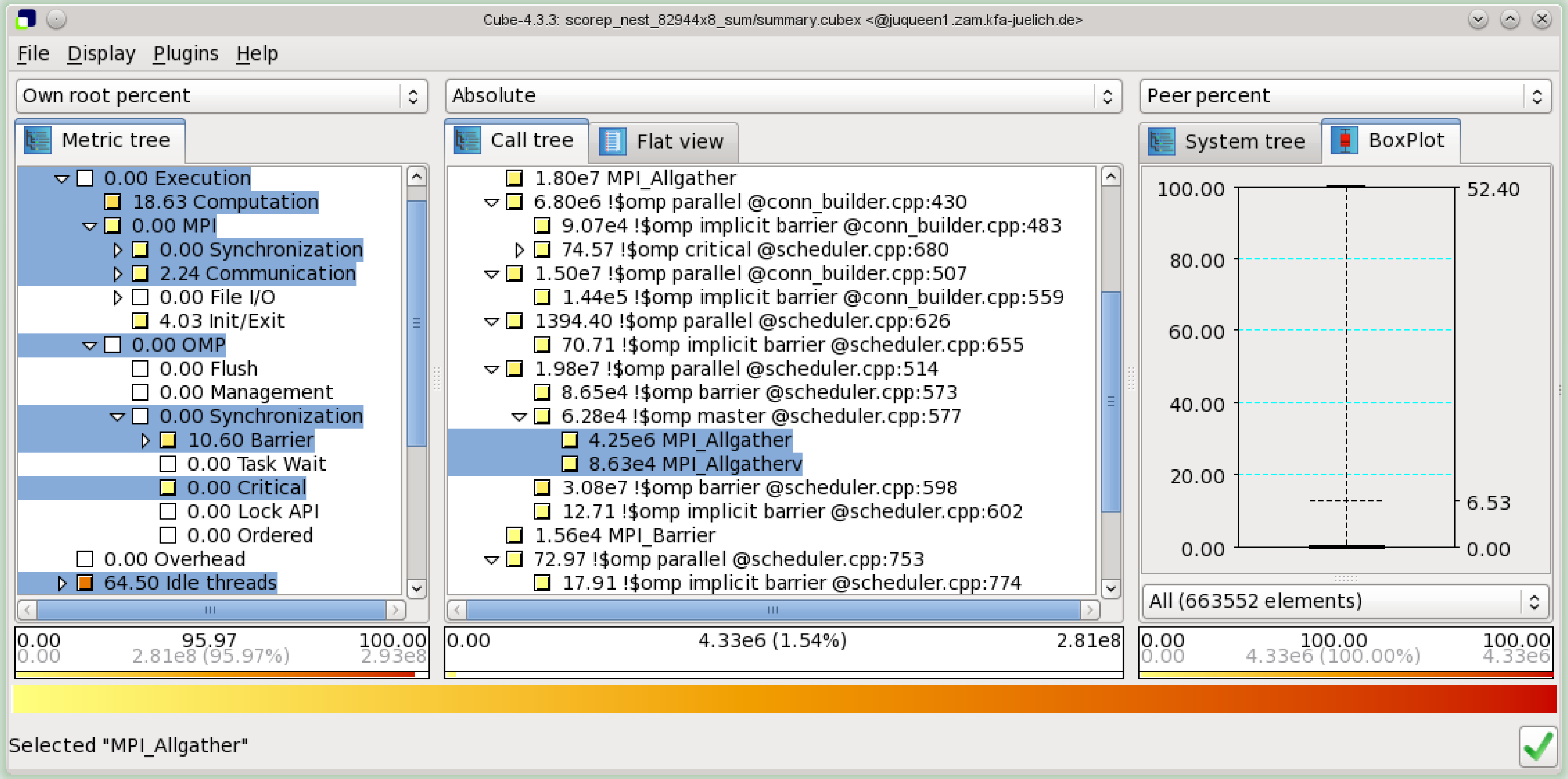
All (663552 elements)

0.00 | 100.00 | 100.00
0.00 | 3.52e7 (100.00%) | 3.52e7

Selected "!$omp barrier @scheduler.cpp:598"

# NEST 82944x8 measurement on K computer: simulation gather_events

Following OpenMP barrier shows imbalance after MPI collective communication



Cube-4.3.3: scorep_nest_82944x8_sum/summary.cubex <@juqueen1.zam.kfa-juelich.de>

File Display Plugins Help

Own root percent | Absolute | Peer percent

Metric tree

```
▽ ☐ 0.00 Execution
      18.63 Computation
  ▽ ☐ 0.00 MPI
    ▷ ☐ 0.00 Synchronization
    ▷ ☐ 2.24 Communication
    ▷ ☐ 0.00 File I/O
      ☐ 4.03 Init/Exit
  ▽ ☐ 0.00 OMP
      ☐ 0.00 Flush
      ☐ 0.00 Management
    ▽ ☐ 0.00 Synchronization
      ▷ ☐ 10.60 Barrier
        ☐ 0.00 Task Wait
        ☐ 0.00 Critical
        ☐ 0.00 Lock API
        ☐ 0.00 Ordered
  ☐ 0.00 Overhead
▷ ☐ 64.50 Idle threads
```

Call tree | Flat view

```
    ☐ 1.80e7 MPI_Allgather
▽ ☐ 6.80e6 !$omp parallel @conn_builder.cpp:430
      ☐ 9.07e4 !$omp implicit barrier @conn_builder.cpp:483
    ▷ ☐ 74.57 !$omp critical @scheduler.cpp:680
▽ ☐ 1.50e7 !$omp parallel @conn_builder.cpp:507
      ☐ 1.44e5 !$omp implicit barrier @conn_builder.cpp:559
▽ ☐ 1394.40 !$omp parallel @scheduler.cpp:626
      ☐ 70.71 !$omp implicit barrier @scheduler.cpp:655
▽ ☐ 1.98e7 !$omp parallel @scheduler.cpp:514
      ☐ 8.65e4 !$omp barrier @scheduler.cpp:573
    ▽ ☐ 6.28e4 !$omp master @scheduler.cpp:577
        ☐ 4.25e6 MPI_Allgather
        ☐ 8.63e4 MPI_Allgatherv
      ☐ 3.08e7 !$omp barrier @scheduler.cpp:598
      ☐ 12.71 !$omp implicit barrier @scheduler.cpp:602
    ☐ 1.56e4 MPI_Barrier
▽ ☐ 72.97 !$omp parallel @scheduler.cpp:753
      ☐ 17.91 !$omp implicit barrier @scheduler.cpp:774
```

System tree | BoxPlot

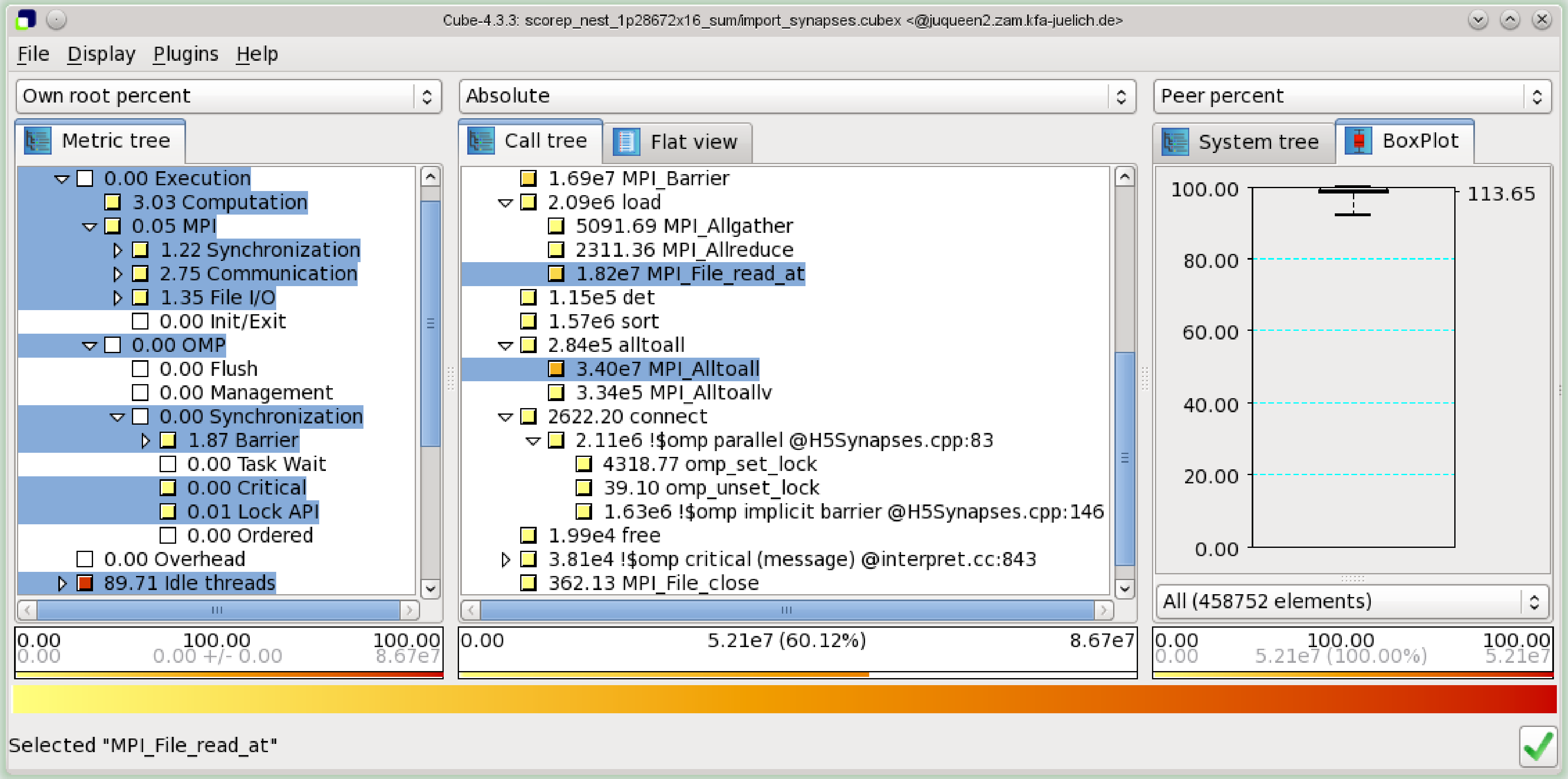
All (663552 elements)

```
0.00        95.97                100.00
0.00   2.81e8 (95.97%)           2.93e8

0.00        4.33e6 (1.54%)       2.81e8

0.00        100.00               100.00
0.00   4.33e6 (100.00%)          4.33e6
```

Selected "MPI_Allgather"

## NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Roughly 2 minutes to load and redistribute synapse data



Cube-4.3.3: scorep_nest_1p28672x16_sum/import_synapses.cubex <@juqueen2.zam.kfa-juelich.de>

File Display Plugins Help

Own root percent | Absolute | Peer percent

Metric tree

```
▽ 0.00 Execution
    3.03 Computation
  ▽ 0.05 MPI
    ▷ 1.22 Synchronization
    ▷ 2.75 Communication
    ▷ 1.35 File I/O
      0.00 Init/Exit
  ▽ 0.00 OMP
      0.00 Flush
      0.00 Management
    ▽ 0.00 Synchronization
      ▷ 1.87 Barrier
        0.00 Task Wait
        0.00 Critical
        0.00 Lock API
        0.00 Ordered
  0.00 Overhead
▷ 89.71 Idle threads
```

Call tree | Flat view

```
  1.69e7 MPI_Barrier
▽ 2.09e6 load
    5091.69 MPI_Allgather
    2311.36 MPI_Allreduce
    1.82e7 MPI_File_read_at
  1.15e5 det
  1.57e6 sort
▽ 2.84e5 alltoall
    3.40e7 MPI_Alltoall
    3.34e5 MPI_Alltoallv
▽ 2622.20 connect
  ▽ 2.11e6 !$omp parallel @H5Synapses.cpp:83
      4318.77 omp_set_lock
      39.10 omp_unset_lock
      1.63e6 !$omp implicit barrier @H5Synapses.cpp:146
  1.99e4 free
▷ 3.81e4 !$omp critical (message) @interpret.cc:843
  362.13 MPI_File_close
```

System tree | BoxPlot

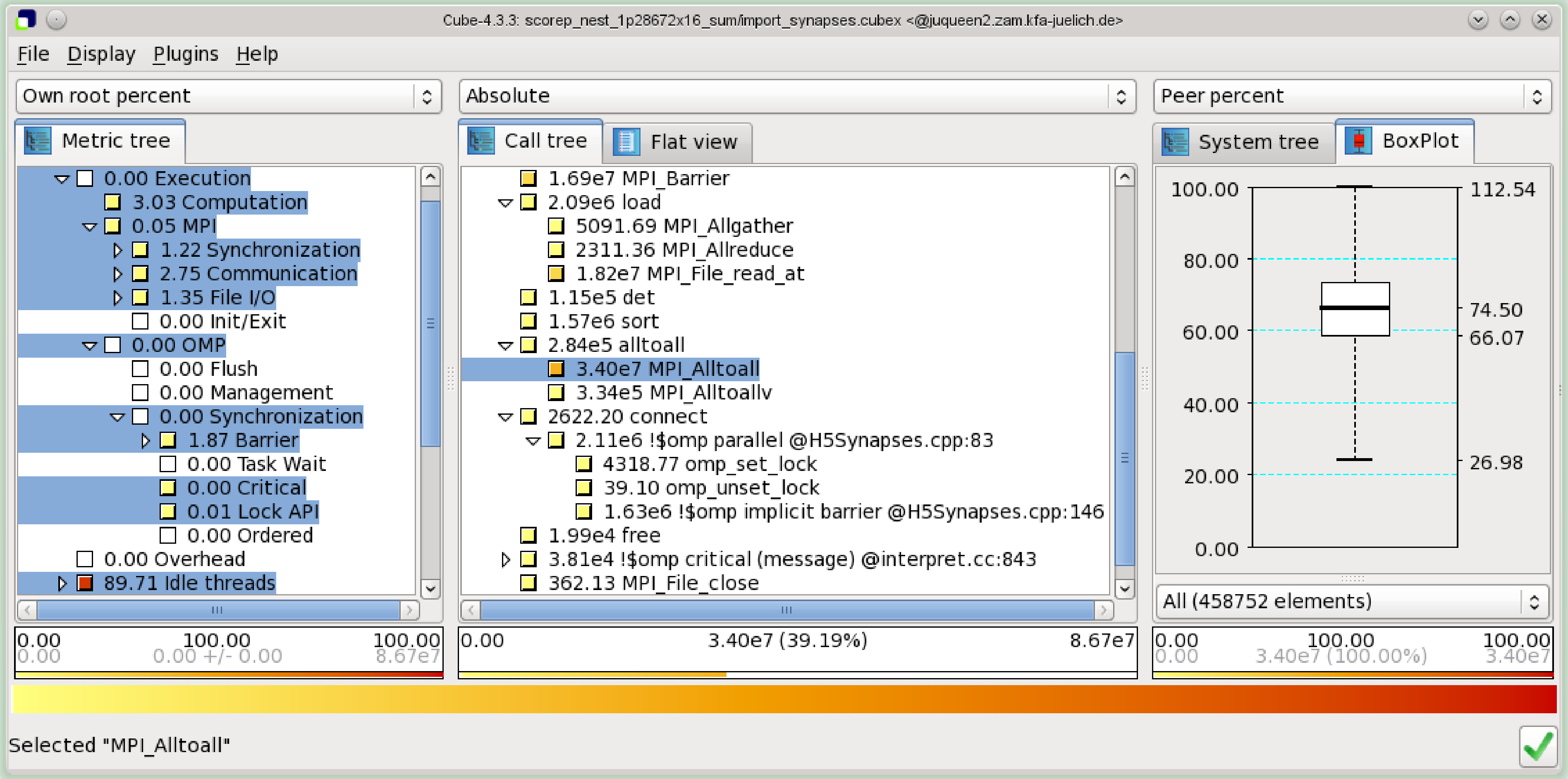
All (458752 elements)

```
0.00          100.00          100.00
0.00     0.00 +/- 0.00        8.67e7

0.00          5.21e7 (60.12%)  8.67e7

0.00          100.00             100.00
0.00     5.21e7 (100.00%)        5.21e7
```

Selected "MPI_File_read_at"

## NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Collective redistribution is expensive, particularly due to preceding computational imbalance



Cube-4.3.3: scorep_nest_1p28672x16_sum/import_synapses.cubex <@juqueen2.zam.kfa-juelich.de>

File Display Plugins Help

Own root percent | Absolute | Peer percent

Metric tree | Call tree | Flat view | System tree | BoxPlot

```
▽ ☐ 0.00 Execution
      ☐ 3.03 Computation
    ▽ ☐ 0.05 MPI
      ▷ ☐ 1.22 Synchronization
      ▷ ☐ 2.75 Communication
      ▷ ☐ 1.35 File I/O
        ☐ 0.00 Init/Exit
    ▽ ☐ 0.00 OMP
        ☐ 0.00 Flush
        ☐ 0.00 Management
      ▽ ☐ 0.00 Synchronization
        ▷ ☐ 1.87 Barrier
          ☐ 0.00 Task Wait
          ☐ 0.00 Critical
          ☐ 0.01 Lock API
          ☐ 0.00 Ordered
  ☐ 0.00 Overhead
▷ ☐ 89.71 Idle threads
```

```
  ☐ 1.69e7 MPI_Barrier
▽ ☐ 2.09e6 load
    ☐ 5091.69 MPI_Allgather
    ☐ 2311.36 MPI_Allreduce
    ☐ 1.82e7 MPI_File_read_at
  ☐ 1.15e5 det
  ☐ 1.57e6 sort
▽ ☐ 2.84e5 alltoall
    ☐ 3.40e7 MPI_Alltoall
    ☐ 3.34e5 MPI_Alltoallv
▽ ☐ 2622.20 connect
  ▽ ☐ 2.11e6 !$omp parallel @H5Synapses.cpp:83
      ☐ 4318.77 omp_set_lock
      ☐ 39.10 omp_unset_lock
      ☐ 1.63e6 !$omp implicit barrier @H5Synapses.cpp:146
  ☐ 1.99e4 free
▷ ☐ 3.81e4 !$omp critical (message) @interpret.cc:843
  ☐ 362.13 MPI_File_close
```

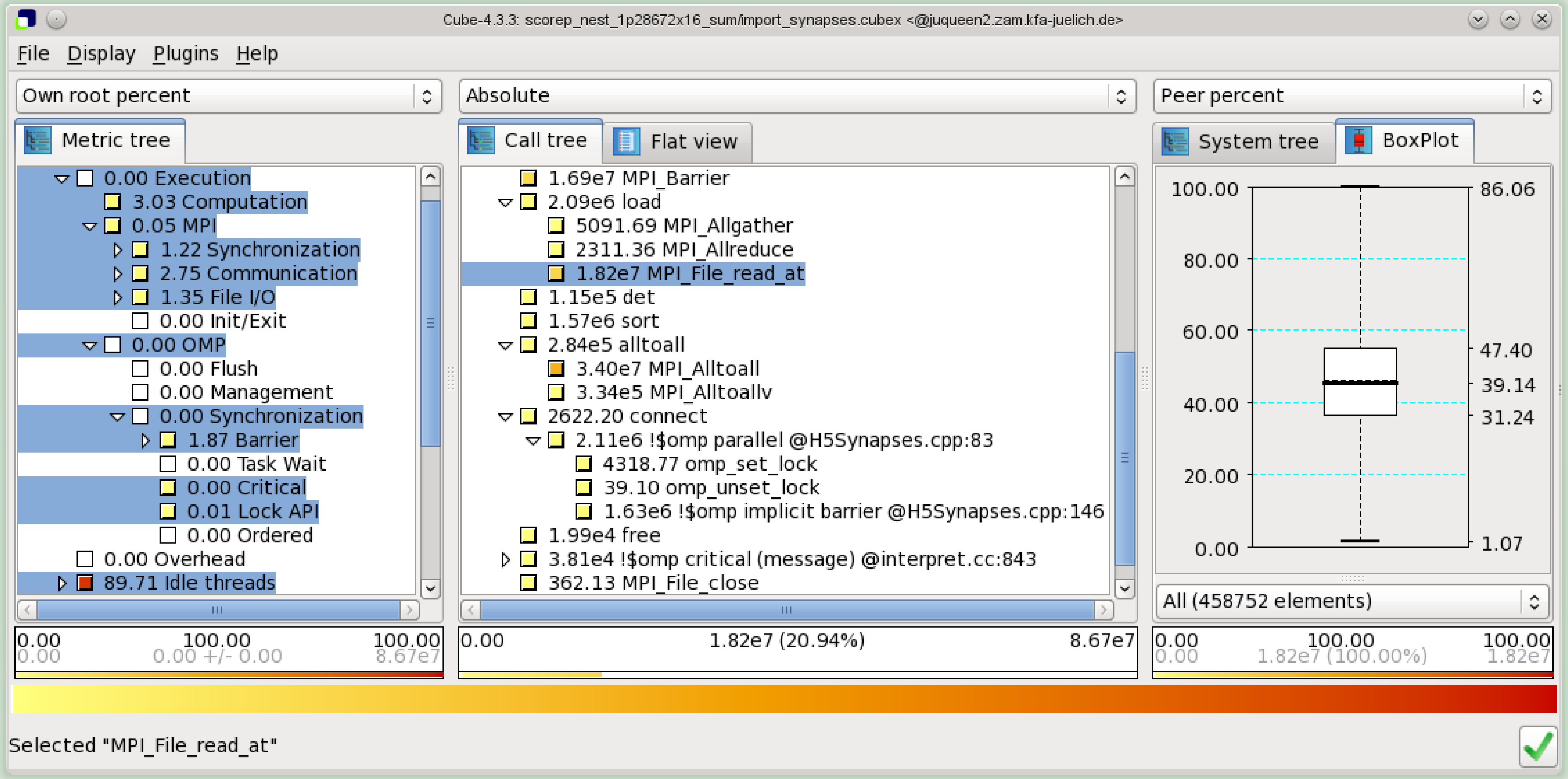
All (458752 elements)

```
0.00            100.00          100.00
0.00       0.00 +/- 0.00        8.67e7

0.00            3.40e7 (39.19%)         8.67e7

0.00            100.00          100.00
0.00       3.40e7 (100.00%)     3.40e7
```

Selected "MPI_Alltoall"

# NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Collective HDF5 read found to be using inefficient individual MPI file reads due to data structure mismatch



Cube-4.3.3: scorep_nest_1p28672x16_sum/import_synapses.cubex <@juqueen2.zam.kfa-juelich.de>

File Display Plugins Help

Own root percent | Absolute | Peer percent

Metric tree

```
▽ ☐ 0.00 Execution
    ☐ 3.03 Computation
  ▽ ☐ 0.05 MPI
    ▷ ☐ 1.22 Synchronization
    ▷ ☐ 2.75 Communication
    ▷ ☐ 1.35 File I/O
      ☐ 0.00 Init/Exit
  ▽ ☐ 0.00 OMP
      ☐ 0.00 Flush
      ☐ 0.00 Management
    ▽ ☐ 0.00 Synchronization
      ▷ ☐ 1.87 Barrier
        ☐ 0.00 Task Wait
        ☐ 0.00 Critical
        ☐ 0.01 Lock API
        ☐ 0.00 Ordered
  ☐ 0.00 Overhead
▷ ☐ 89.71 Idle threads
```

Call tree | Flat view

```
☐ 1.69e7 MPI_Barrier
▽ ☐ 2.09e6 load
    ☐ 5091.69 MPI_Allgather
    ☐ 2311.36 MPI_Allreduce
    ☐ 1.82e7 MPI_File_read_at
☐ 1.15e5 det
☐ 1.57e6 sort
▽ ☐ 2.84e5 alltoall
    ☐ 3.40e7 MPI_Alltoall
    ☐ 3.34e5 MPI_Alltoallv
▽ ☐ 2622.20 connect
  ▽ ☐ 2.11e6 !$omp parallel @H5Synapses.cpp:83
      ☐ 4318.77 omp_set_lock
      ☐ 39.10 omp_unset_lock
      ☐ 1.63e6 !$omp implicit barrier @H5Synapses.cpp:146
☐ 1.99e4 free
▷ ☐ 3.81e4 !$omp critical (message) @interpret.cc:843
☐ 362.13 MPI_File_close
```

System tree | BoxPlot

All (458752 elements)

```
0.00        100.00          100.00 | 0.00    1.82e7 (20.94%)    8.67e7 | 0.00       100.00            100.00
0.00    0.00 +/- 0.00       8.67e7 |                                   | 0.00   1.82e7 (100.00%)     1.82e7
```

Selected "MPI_File_read_at"

## Extreme scaling

7th Jülich Blue Gene Extreme Scaling Workshop (1-3 Feb 2016)

- latest of series starting with JUBL (BG/L), then JUGENE (BG/P), now JUQUEEN (BG/Q)
- full 28 racks dedicated over 50 hours for 8 international code teams

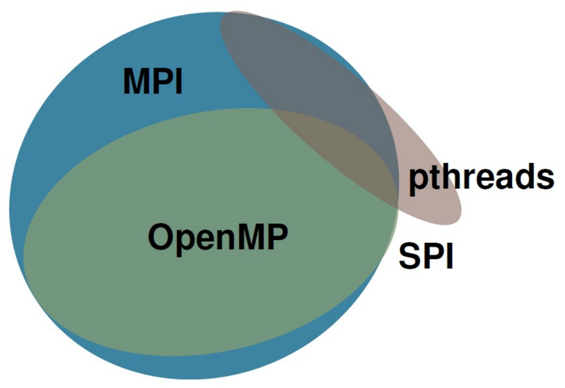
High Q Club [www.fz-juelich.de/ias/jsc/high-q-club]

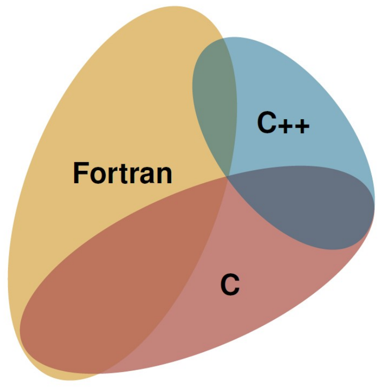
- application codes with demonstrated scalability to use entire JUQUEEN resource
  - *up to 1.75M MPI processes or OpenMP threads running on 458,752 cores*
- currently 25 members, several more applications pending

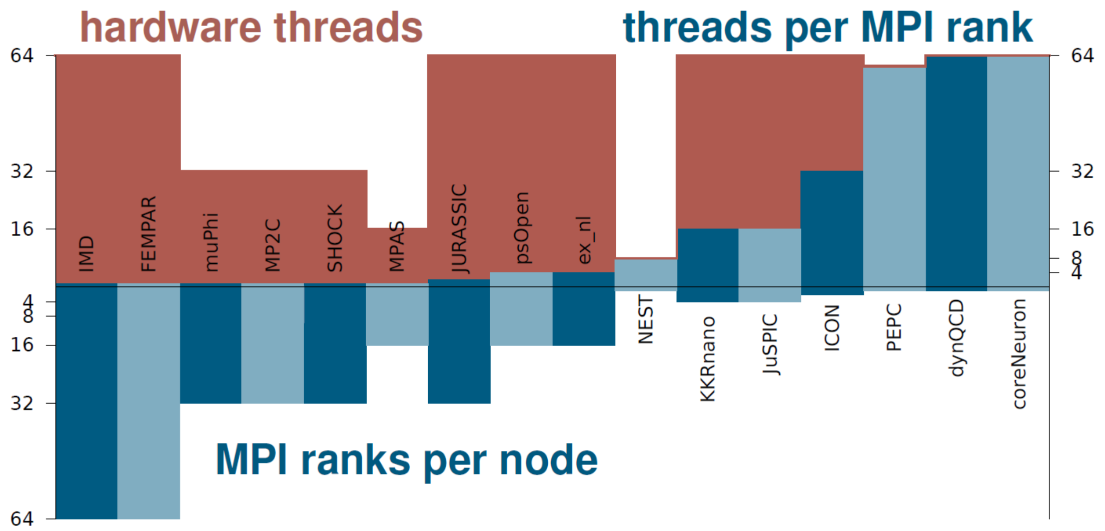
Discussion and information exchange

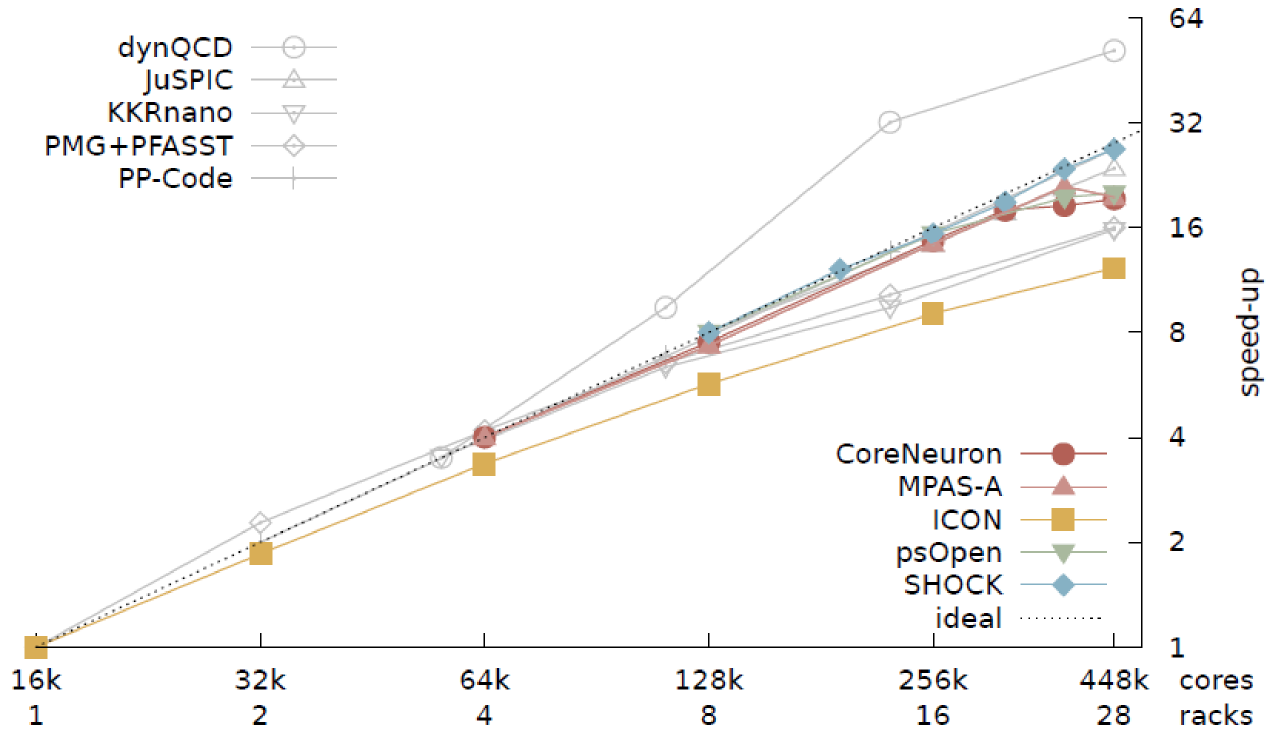
- aXXLs@ISC-HPC15: Application Extreme-scaling Experience of Leading Supercomputing Centres
- MAXI@ParCo15: Multi-system Application Extreme-scaling Imperative

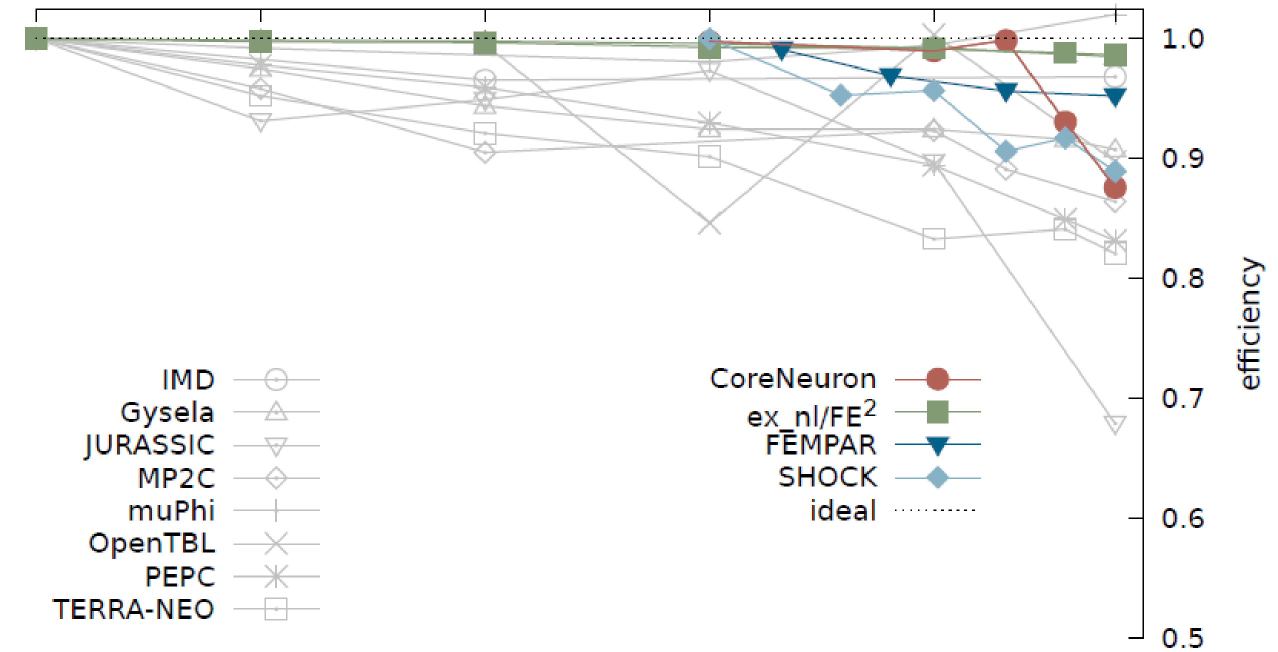
# High-Q Club member characteristics









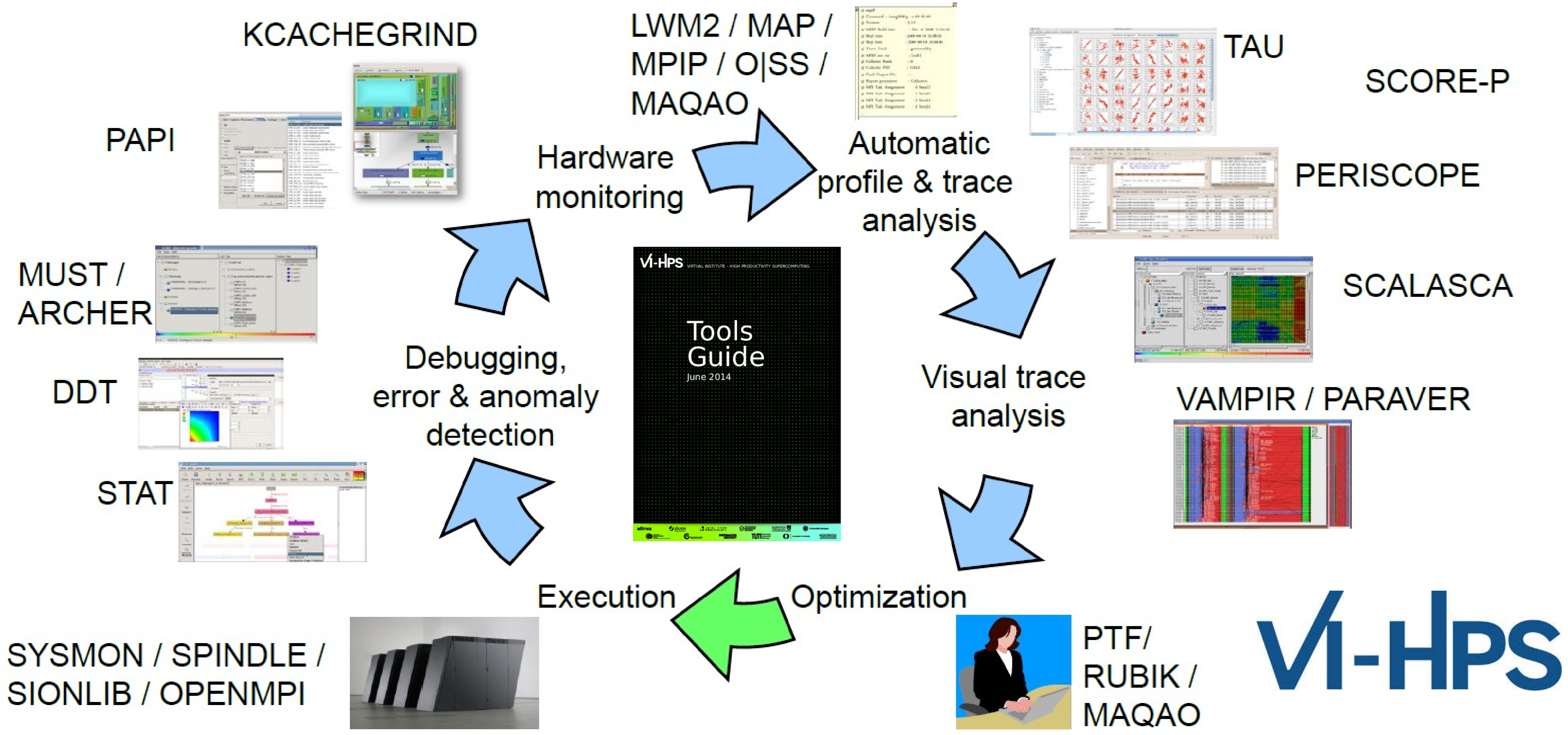


## Scalasca training offered through VI-HPS

Virtual Institute – High Productivity Supercomputing [www.vi-hps.org]

- focus on parallel performance, correctness & debugging tools
- several VI-HPS Tuning Workshops each year
  - *3-5 days for application developers to get introduced to tools and receive guidance and assistance applying tools to their own codes*
  - *RIKEN AICS hosting VI-HPS-TW20 (24-26 Feb 2016) for users of K computer and related Fujitsu FX10/100 systems*
    - – Score-P, Scalasca, TAU & BSC tools
  - *additional workshops at LRZ in Garching, Germany & CINES in Montpellier, France*

## VI-HPS tools and their integration


KCACHEGRIND
LWM2 / MAP / MPIP / O|SS / MAQAO
TAU
SCORE-P
PAPI
Hardware monitoring
Automatic profile & trace analysis
PERISCOPE
MUST / ARCHER
SCALASCA
DDT
Debugging, error & anomaly detection
Tools Guide
June 2014
Visual trace analysis
VAMPIR / PARAVER
STAT
Execution
Optimization
SYSMON / SPINDLE / SIONLIB / OPENMPI
PTF/ RUBIK / MAQAO
VI-HPS