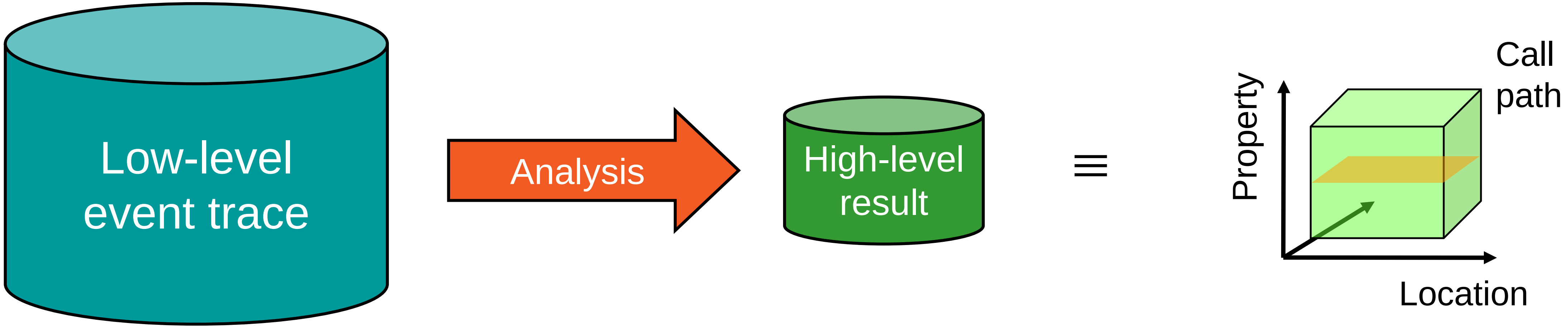
Automatic trace analysis with the Scalasca Trace Tools

Brian Wylie
Jülich Supercomputing Centre

Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & quantification of significance
  - Identification of delays as root causes of inefficiencies



- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

Scalasca Trace Tools: Objective

- Development of a **scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and POSIX threads

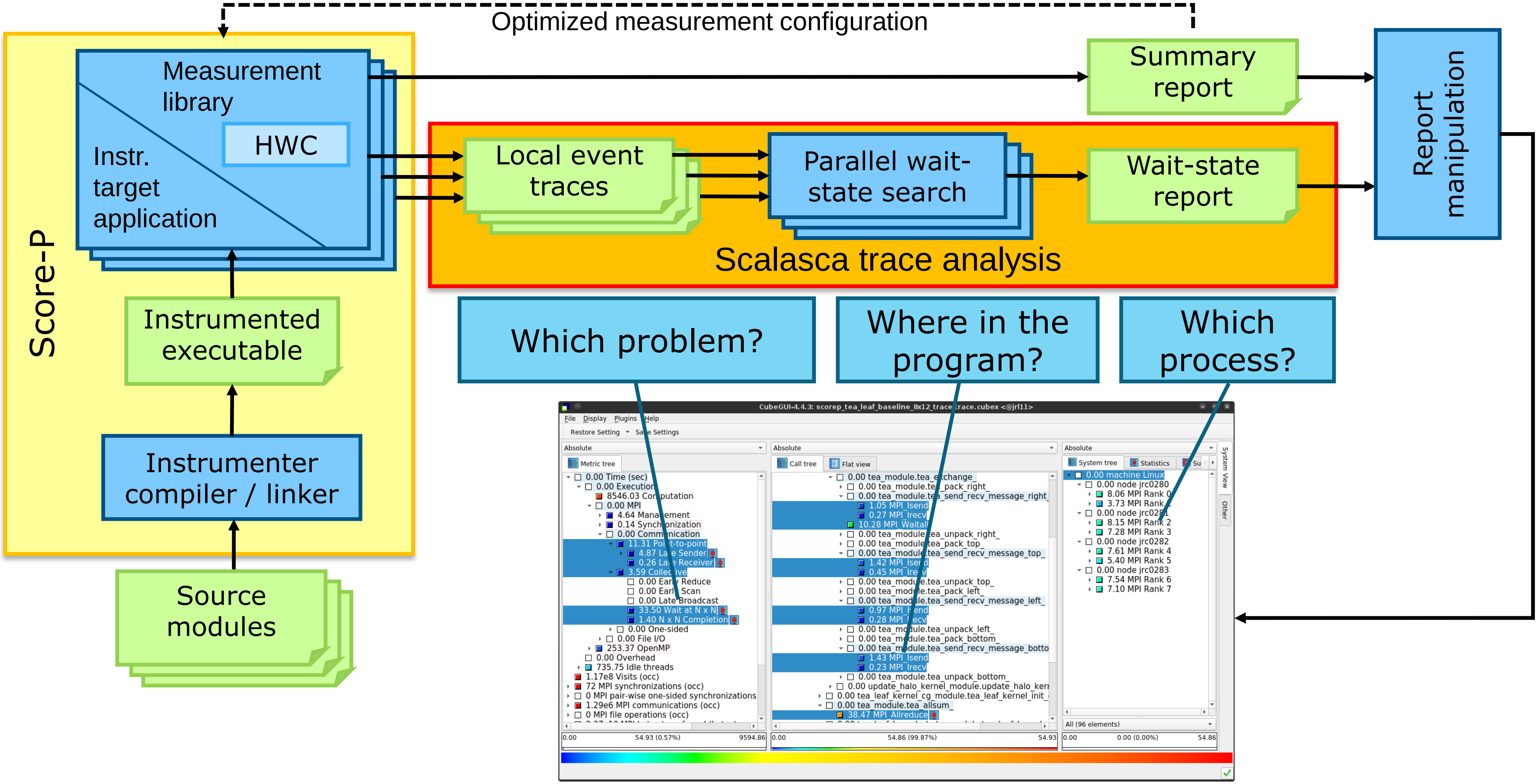
- Specifically targeting large-scale parallel applications
  - Such as those running on IBM Blue Gene or Cray systems with one million or more processes/threads

- Latest release:
  - Scalasca v2.5 coordinated with Score-P v5.0 (March 2019)

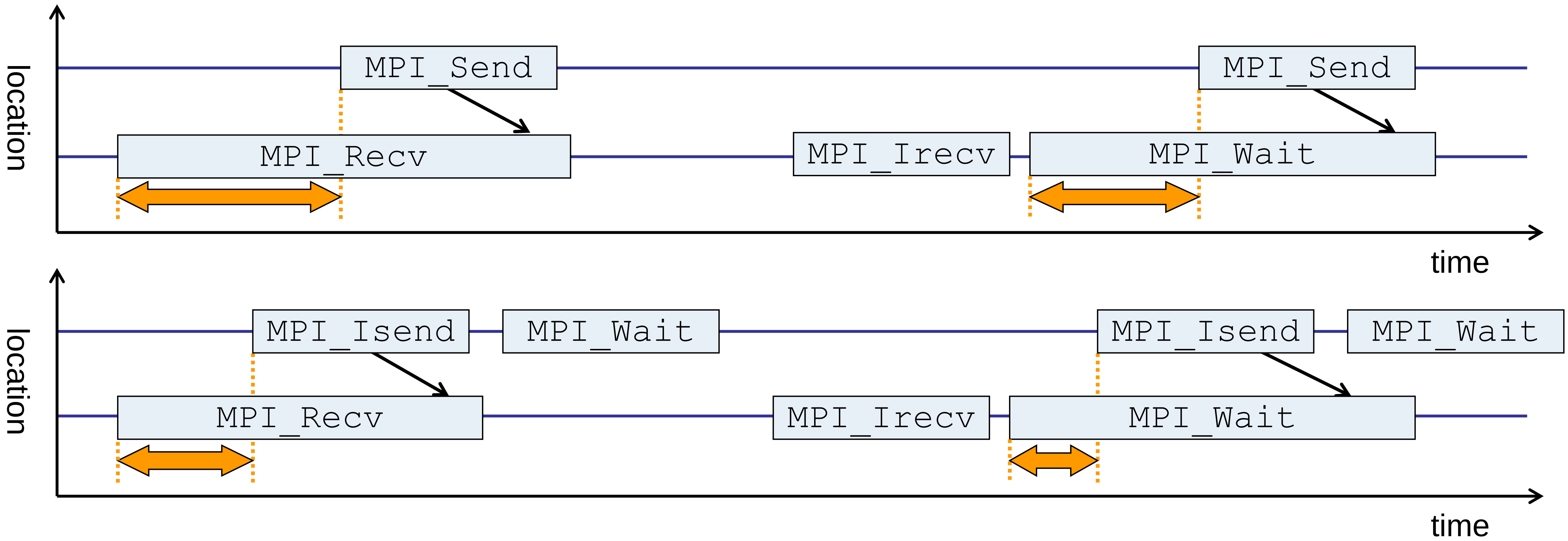
Scalasca Trace Tools features

- Open source, 3-clause BSD license
- Fairly portable
  - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX10/100 & K computer, Linux clusters (x86, Power, ARM), Intel Xeon Phi, ...
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format
- Current limitations:
  - Unable to handle traces
    - With MPI thread level exceeding MPI_THREAD_FUNNELED
    - Containing CUDA or SHMEM events, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored

Scalasca workflow

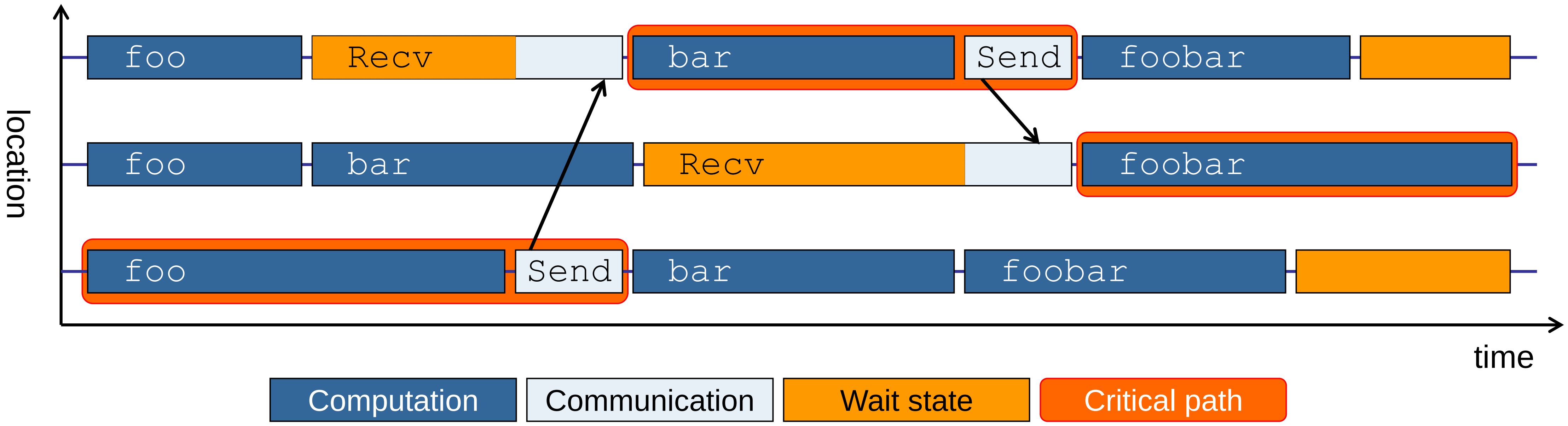


Example: "*Late Sender*" wait state



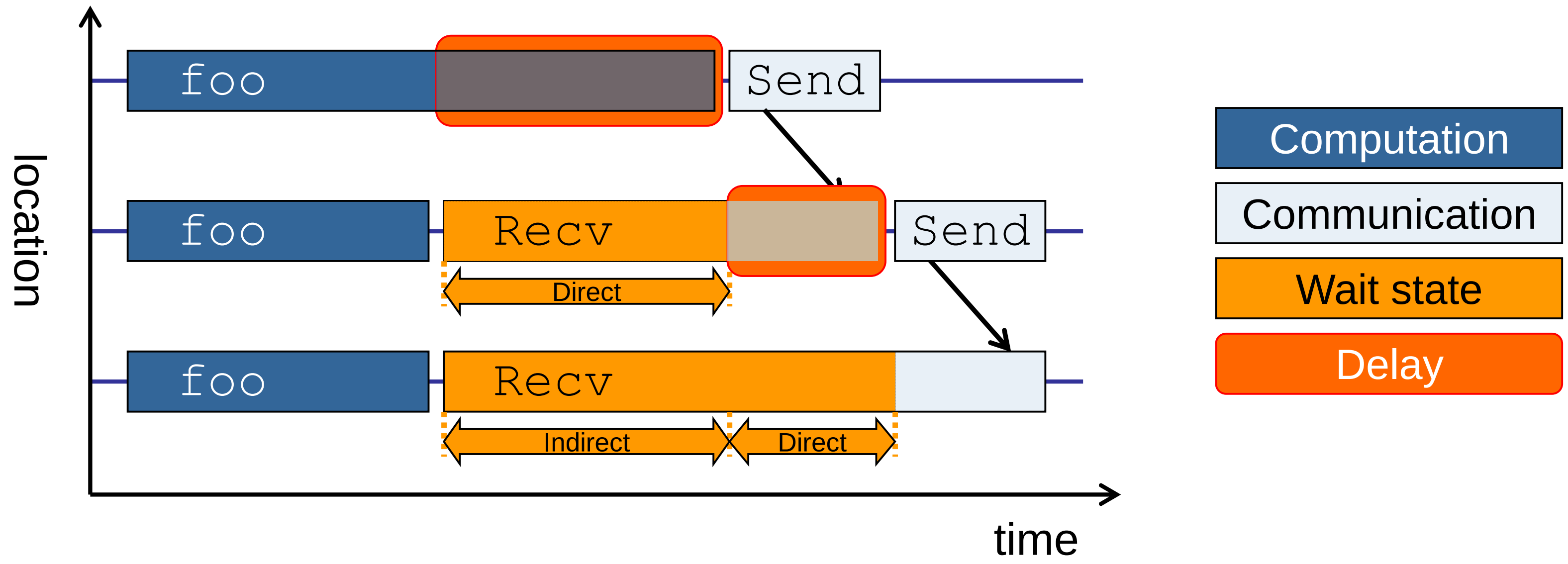
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

Example: Critical path



- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks

Example: Root-cause analysis



- Classifies wait states into direct and indirect (i.e., caused by other wait states)
- Identifies *delays* (excess computation/communication) as root causes of wait states
- Attributes wait states as *delay costs*

Case study: TeaLeaf

trace tools
scalasca

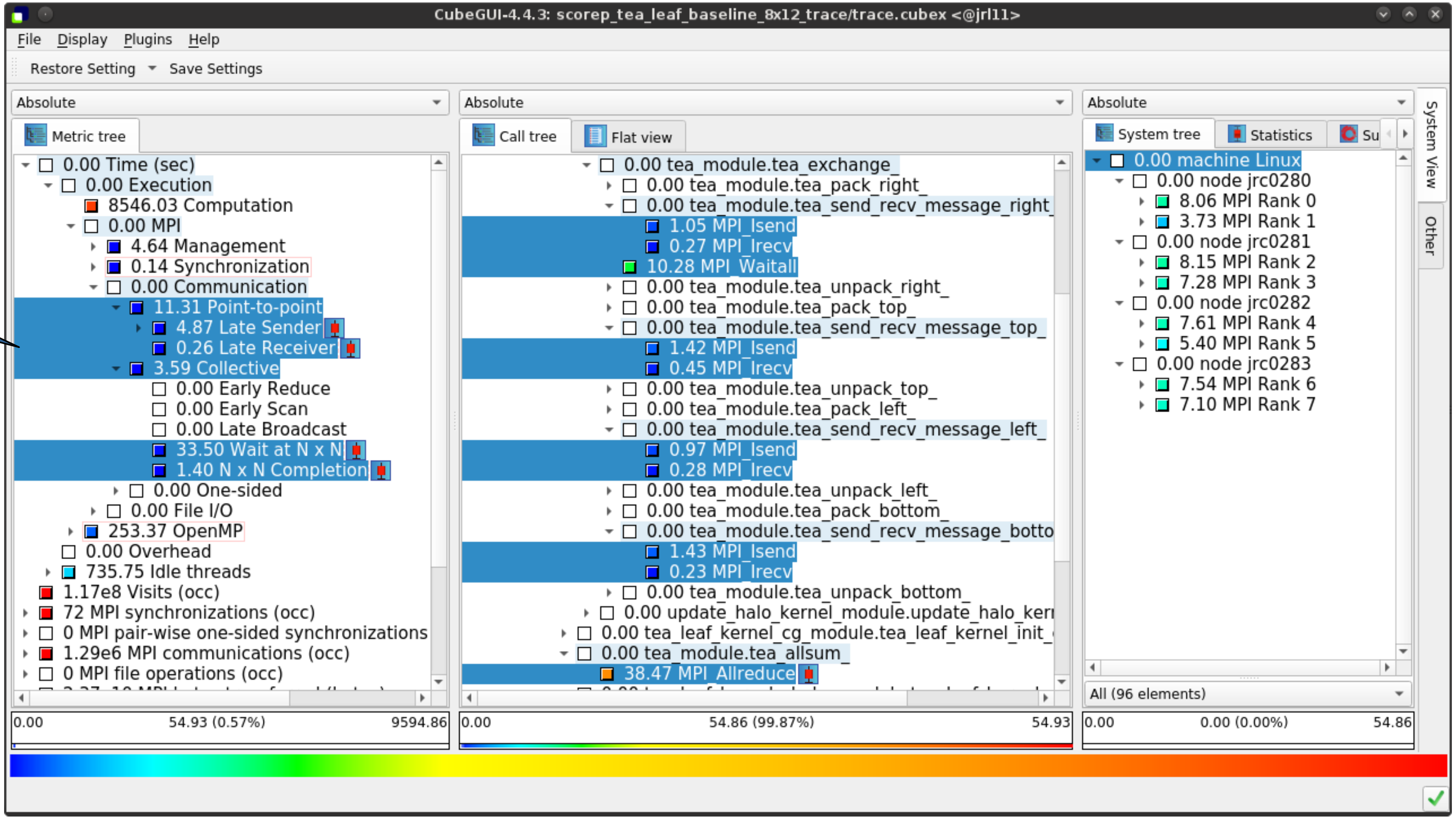
Case study: TeaLeaf

- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: http://uk-mac.github.io/TeaLeaf/
- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, Score-P 5.0, and Scalasca 2.5
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each
    - Distributed across 4 compute nodes (2 ranks per node)
    - Test problem “5”: 4000 × 4000 cells, CG solver

```
% cp -r /p/scratch/share/VI-HPS/examples/TeaLeaf . && cd TeaLeaf
% square scorep_tea_leaf_baseline_8x12_trace
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_tea_leaf_baseline_8x12_trace/trace.cubex...
                         [GUI showing post-processed trace analysis report]
```

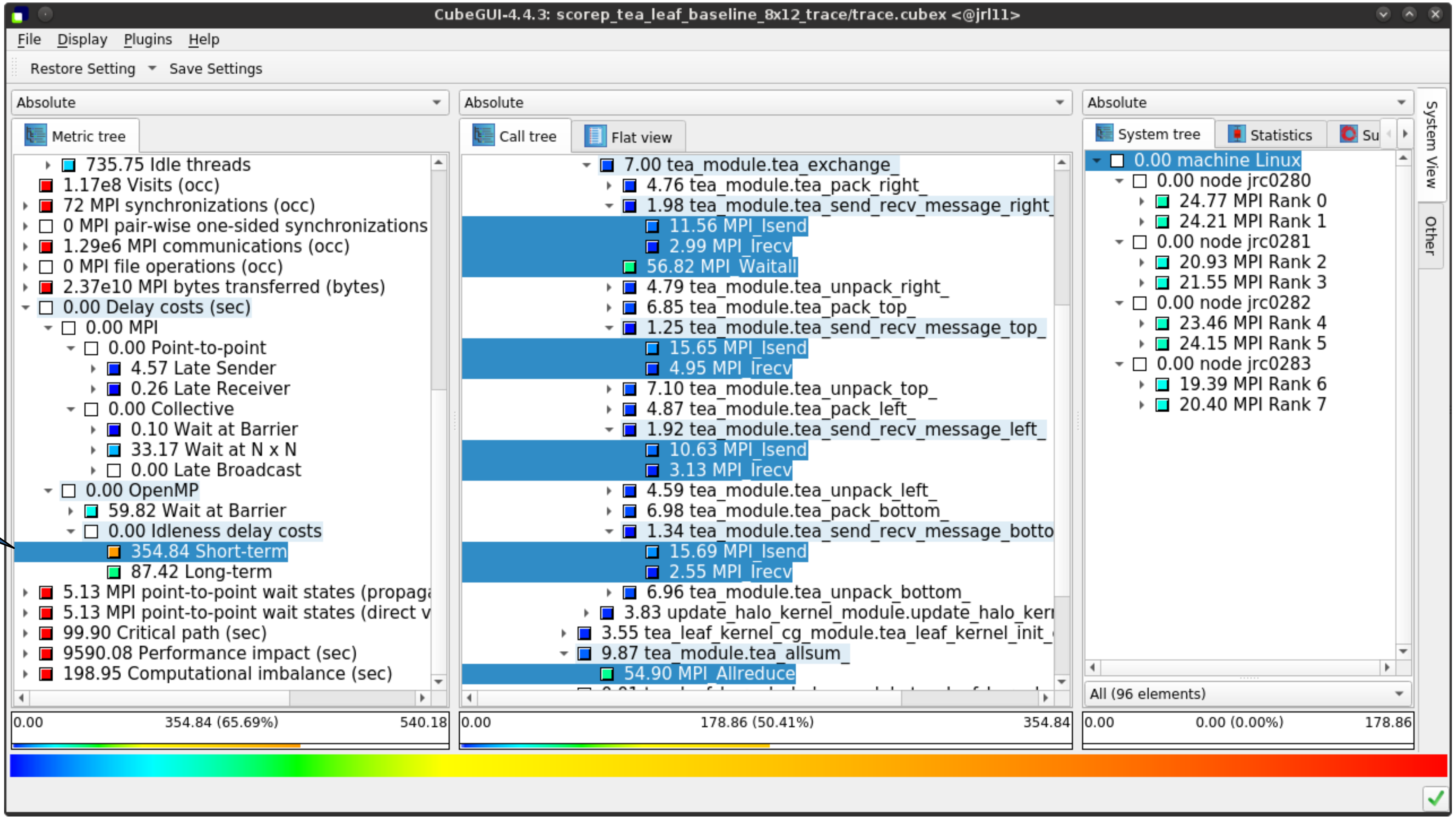
TeaLeaf Scalasca report analysis (I)

While MPI communication time and wait states are small (~0.6% of the total execution time)…



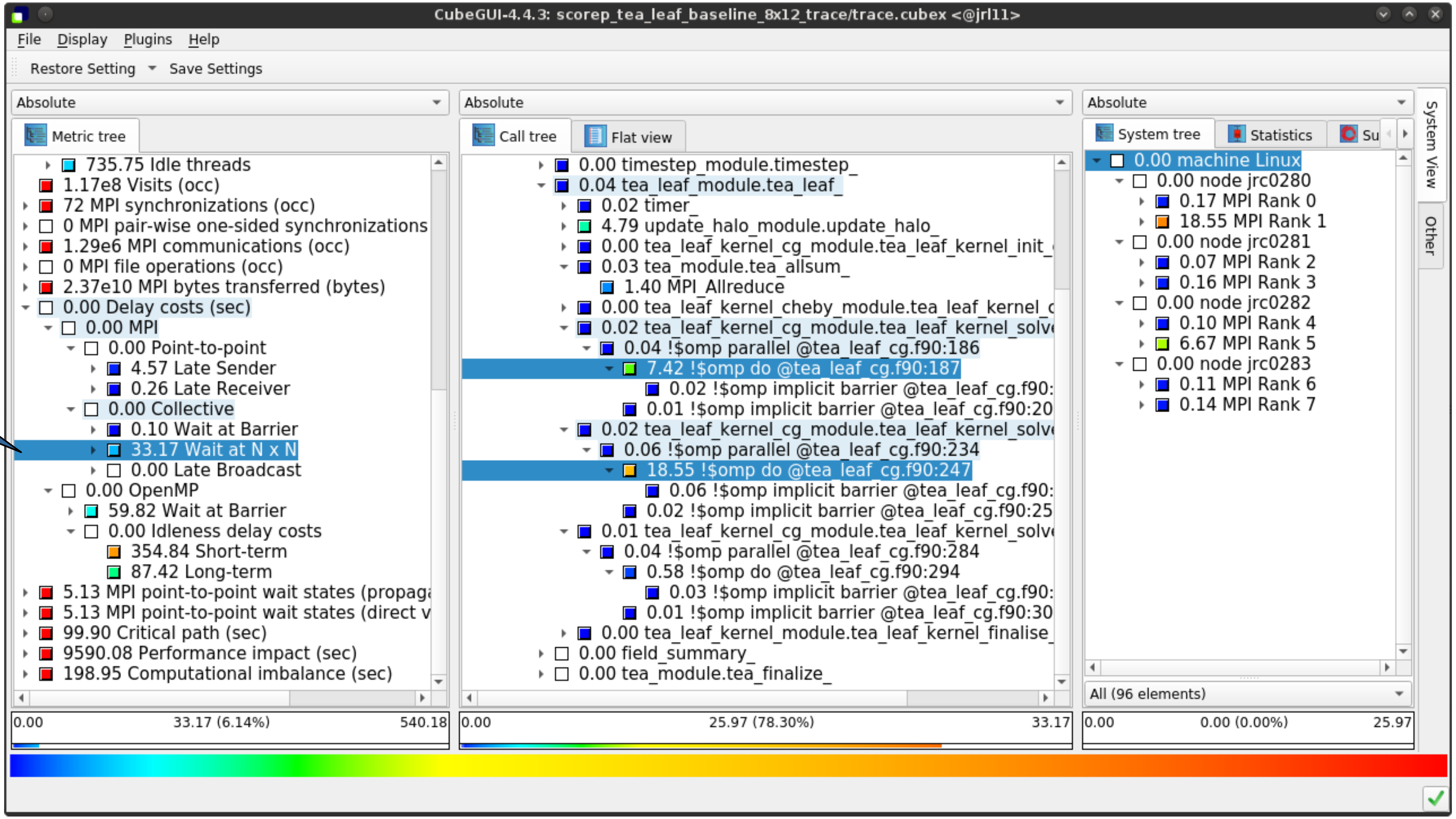
TeaLeaf Scalasca report analysis (II)

…they directly cause a significant amount of the OpenMP thread idleness



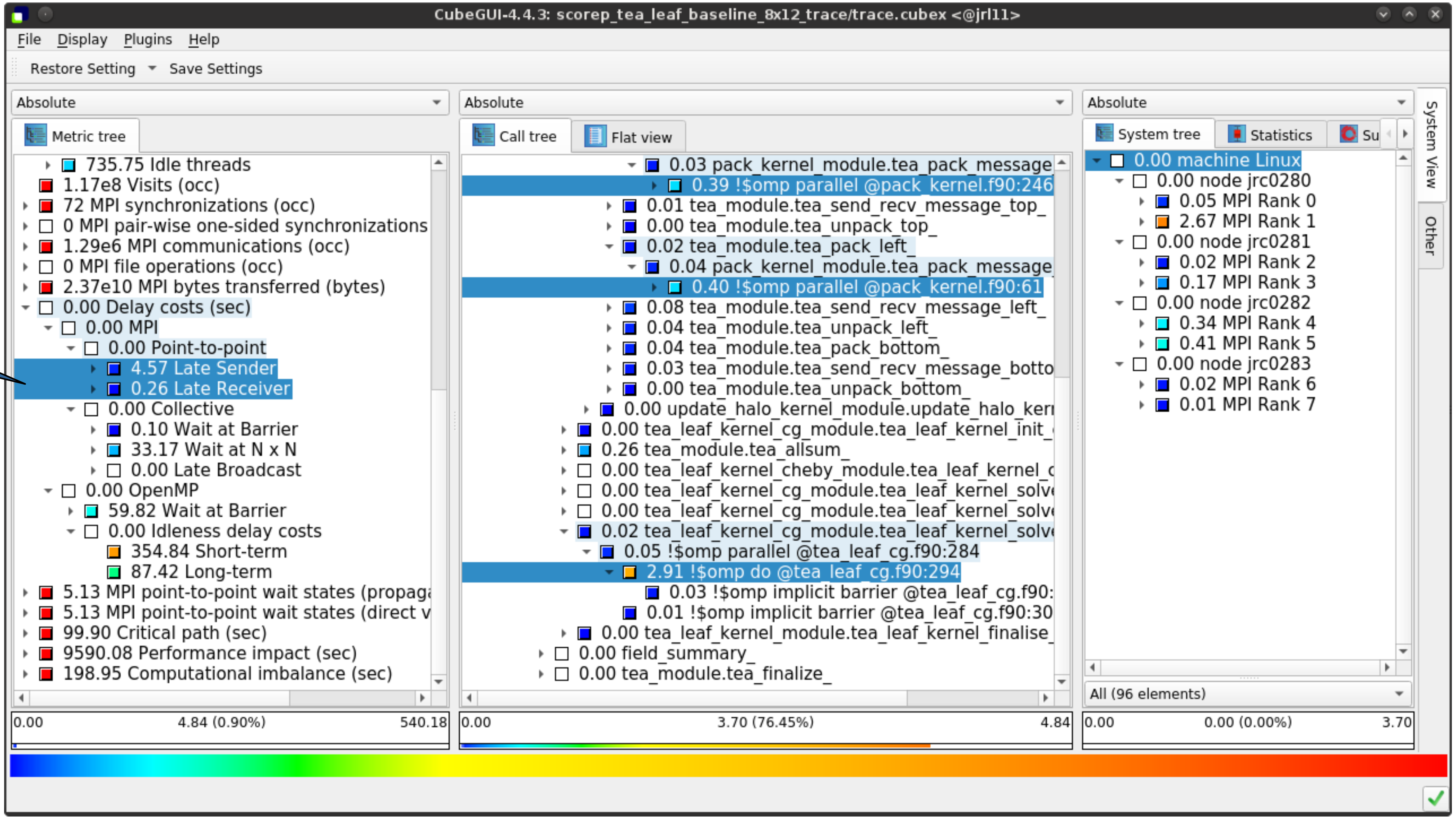
TeaLeaf Scalasca report analysis (III)

The "Wait at NxN" collective wait states are mostly caused by the first 2 OpenMP do loops of the solver (on ranks 5 & 1, resp.)…



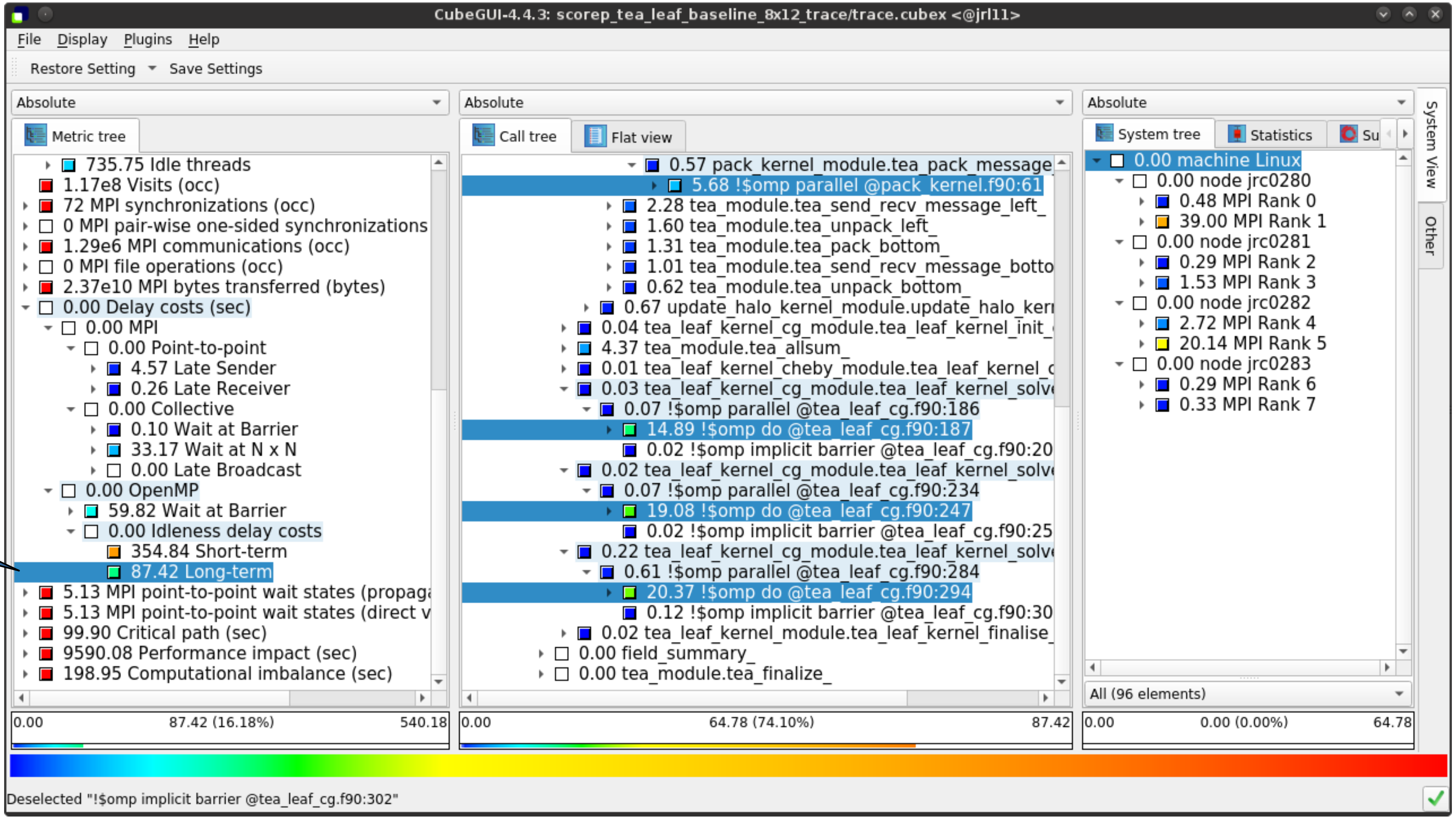
TeaLeaf Scalasca report analysis (IV)

…while the MPI point-to-point wait states are caused by the 3rd solver `do` loop (on rank 1) and two loops in the halo exchange



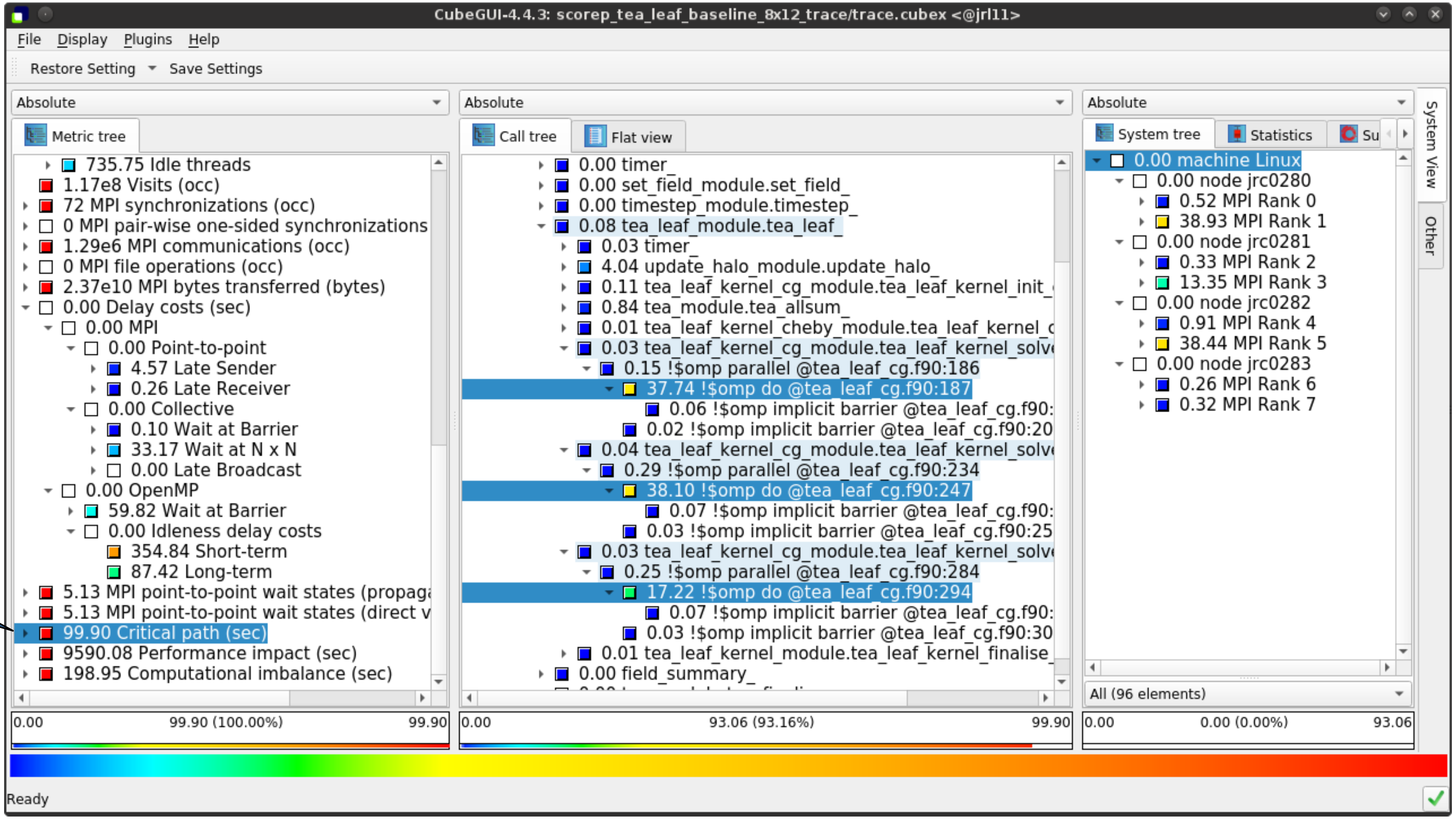
TeaLeaf Scalasca report analysis (V)

Various OpenMP do loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation



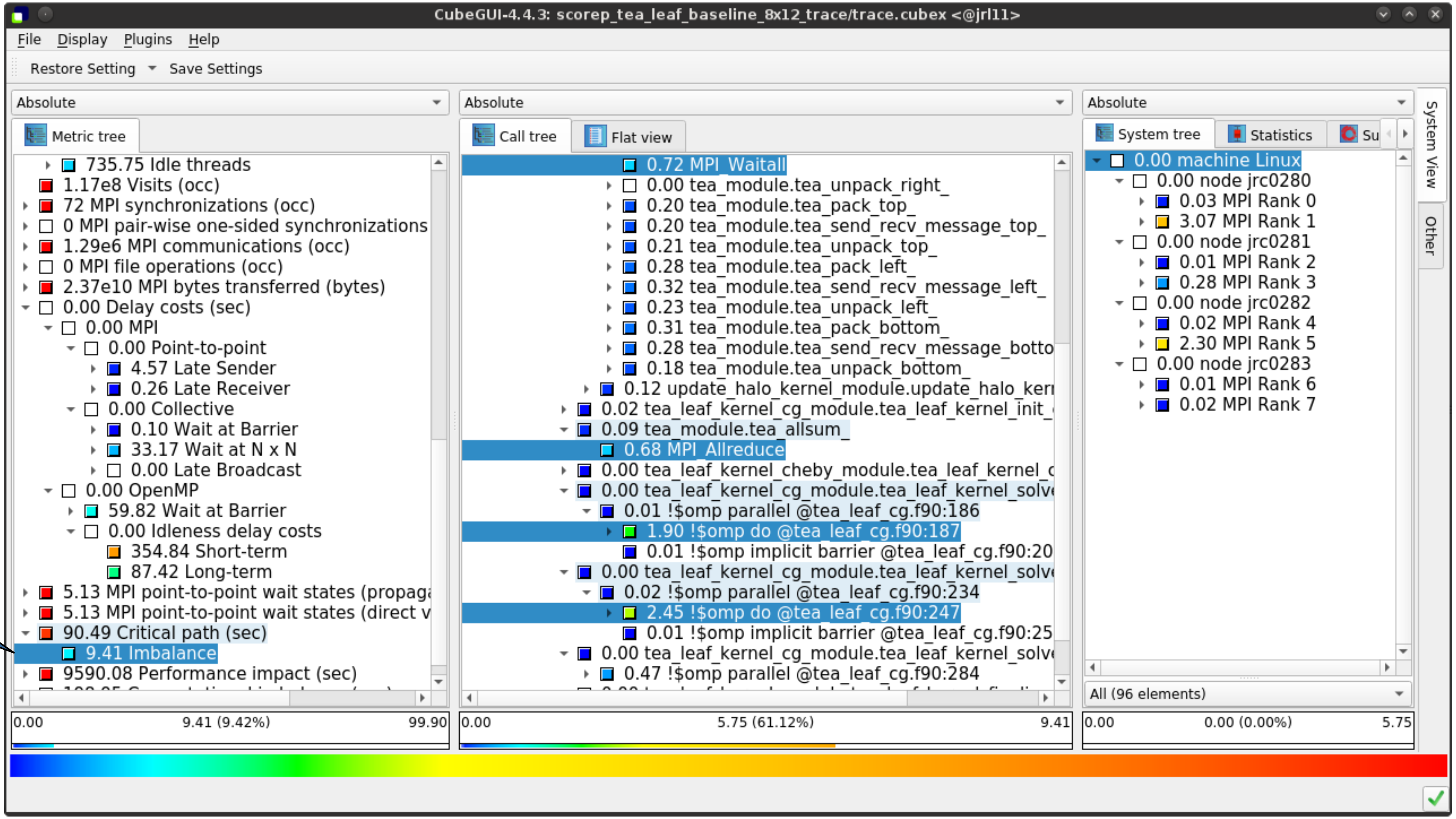
TeaLeaf Scalasca report analysis (VI)

The Critical Path also highlights the three solver loops…



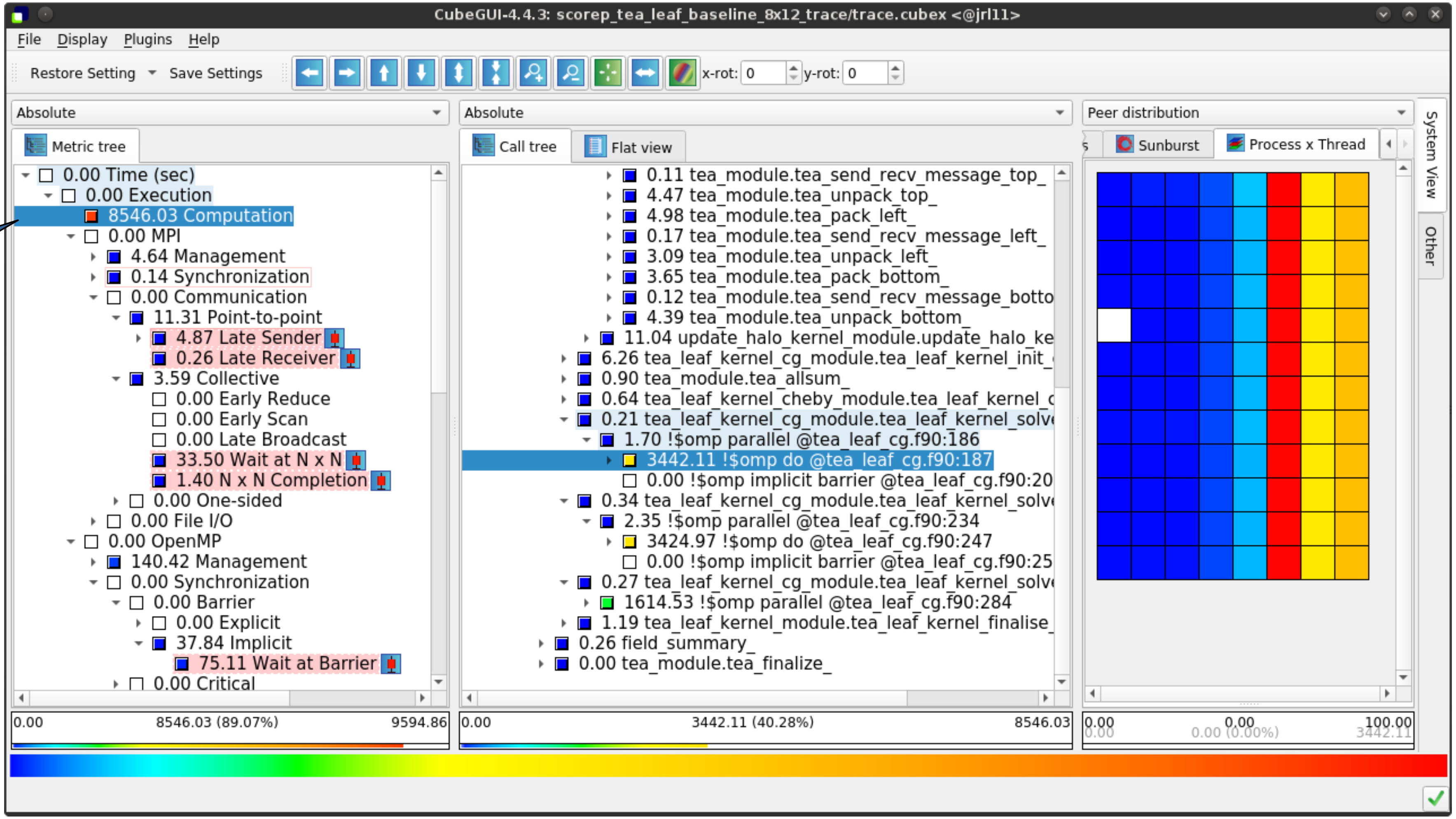
TeaLeaf Scalasca report analysis (VII)

…with imbalance (time on critical path above average) mostly in the first two loops and MPI communication



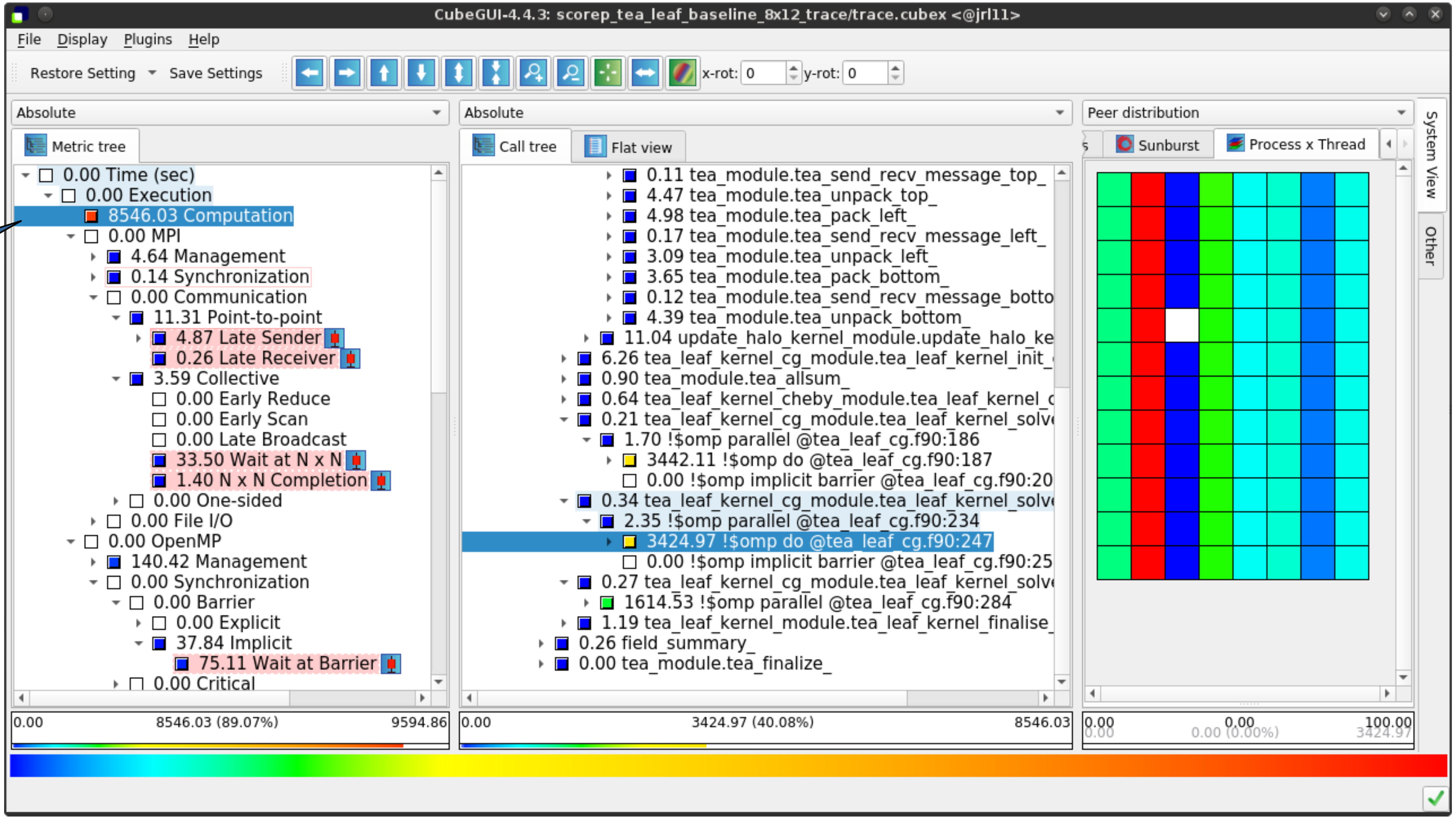
TeaLeaf Scalasca report analysis (VIII)

Computation time of 1st…



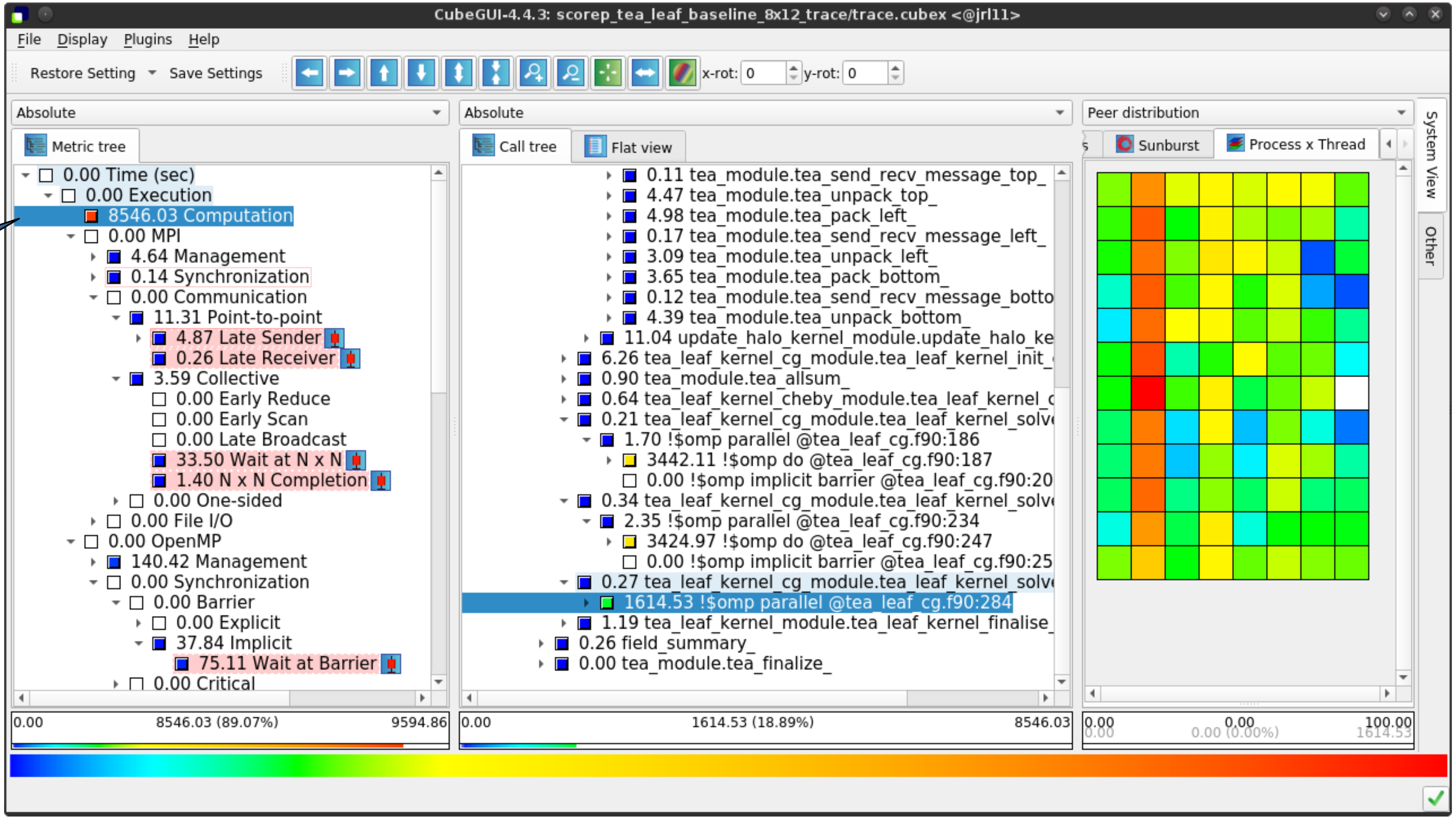
TeaLeaf Scalasca report analysis (IX)

…and 2nd `do` loop mostly balanced within each rank, but vary considerably across ranks…



TeaLeaf Scalasca report analysis (X)

…while the 3rd `do` loop also shows imbalance within each rank



TeaLeaf analysis summary

- The first two OpenMP do loops of the solver are well balanced within a rank, but are imbalanced across ranks
  ➔ Requires a global load balancing strategy
- The third OpenMP do loop, however, is imbalanced within ranks,
  - causing direct "Wait at OpenMP Barrier" wait states,
  - which cause indirect MPI point-to-point wait states,
  - which in turn cause OpenMP thread idleness
  ➔ Low-hanging fruit

- Adding a `SCHEDULE(guided)` clause reduced
  - the MPI point-to-point wait states by ~66%
  - the MPI collective wait states by ~50%
  - the OpenMP "Wait at Barrier" wait states by ~55%
  - the OpenMP thread idleness by ~11%
  ➔ **Overall runtime (wall-clock) reduction by ~5%**

Hands-on:
NPB-MZ-MPI / BT

trace tools
scalasca

Performance analysis steps

- 0.0 Reference preparation for validation

- 1.0 Program instrumentation
- 1.1 Summary measurement collection
- 1.2 Summary analysis report examination

- 2.0 Summary experiment scoring
- 2.1 Summary measurement collection with filtering
- 2.2 Filtered summary analysis report examination

- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

Scalasca command – One command for (almost) everything

```
% scalasca
Scalasca 2.5
Toolset for scalable performance analysis of large-scale parallel applications
usage: scalasca [OPTION]... ACTION <argument>...
    1. prepare application objects and executable for measurement:
       scalasca -instrument <compile-or-link-command> # skin (using scorep)
    2. run application under control of measurement system:
       scalasca -analyze <application-launch-command> # scan
    3. interactively explore measurement analysis report:
       scalasca -examine <experiment-archive|report>  # square

Options:
   -c, --show-config     show configuration summary and exit
   -h, --help            show this help and exit
   -n, --dry-run         show actions without taking them
       --quickref        show quick reference guide and exit
       --remap-specfile  show path to remapper specification file and exit
   -v, --verbose         enable verbose commentary
   -V, --version         show version information and exit
```

- The '`scalasca -instrument`' command is deprecated and only provided for backwards compatibility with Scalasca 1.x., recommended: use Score-P instrumenter directly

Scalasca compatibility command: skin / scalasca -instrument

```
% skin
Scalasca 2.5: application instrumenter (using Score-P instrumenter)
usage: skin [-v] [-comp] [-pdt] [-pomp] [-user] [--*] <compile-or-link-command>
  -comp={all|none|...}: routines to be instrumented by compiler [default: all]
                 (... custom instrumentation specification depends on compiler)
  -pdt:  process source files with PDT/TAU instrumenter
  -pomp: process source files for POMP directives
  -user: enable EPIK user instrumentation API macros in source code
  -v:    enable verbose commentary when instrumenting

  --*:   options to pass to Score-P instrumenter
```

- Scalasca application instrumenter
  - Provides compatibility with Scalasca 1.x
  - **Deprecated! Use Score-P instrumenter directly.**

Scalasca convenience command: scan / scalasca -analyze

```
% scan
Scalasca 2.5: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
      where {options} may include:
  -h    Help      : show this brief usage message and exit.
  -v    Verbose   : increase verbosity.
  -n    Preview   : show command(s) to be launched but don't execute.
  -q    Quiescent : execution with neither summarization nor tracing.
  -s    Summary   : enable runtime summarization. [Default]
  -t    Tracing   : enable trace collection and analysis.
  -a    Analyze   : skip measurement to (re-)analyze an existing trace.
  -e exptdir      : Experiment archive to generate and/or analyze.
                    (overrides default experiment archive title)
  -f filtfile     : File specifying measurement filter.
  -l lockfile     : File that blocks start of measurement.
  -R #runs        : Specify the number of measurement runs per config.
  -M cfgfile      : Specify a config file for a multi-run measurement.
```

- Scalasca measurement collection & analysis nexus

Scalasca advanced command: scout - Scalasca automatic trace analyzer

```
% scout.hyb --help
SCOUT   (Scalasca 2.5)
Copyright (c) 1998-2019 Forschungszentrum Juelich GmbH
Copyright (c) 2009-2014 German Research School for Simulation Sciences GmbH

Usage: <launchcmd> scout.hyb [OPTION]... <ANCHORFILE | EPIK DIRECTORY>
Options:
  --statistics       Enables instance tracking and statistics [default]
  --no-statistics    Disables instance tracking and statistics
  --critical-path    Enables critical-path analysis [default]
  --no-critical-path Disables critical-path analysis
  --rootcause        Enables root-cause analysis [default]
  --no-rootcause     Disables root-cause analysis
  --single-pass      Single-pass forward analysis only
  --time-correct     Enables enhanced timestamp correction
  --no-time-correct  Disables enhanced timestamp correction [default]
  --verbose, -v      Increase verbosity
  --help             Display this information and exit
```

- Provided in serial (.ser), OpenMP (.omp), MPI (.mpi) and MPI+OpenMP (.hyb) variants

Scalasca advanced command: clc_synchronize

- Scalasca trace event timestamp consistency correction

```
Usage: <launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>
```

  - Provided in MPI (.mpi) and MPI+OpenMP (.hyb) variants
  - Takes as input a trace experiment archive where the events may have timestamp inconsistencies
    - E.g., multi-node measurements on systems without adequately synchronized clocks on each compute node
  - Generates a new experiment archive (always called ./clc_sync) containing a trace with event timestamp inconsistencies resolved
    - E.g., suitable for detailed examination with a time-line visualizer

Scalasca convenience command: square / scalasca -examine

```
% square
Scalasca 2.5: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
   -c <none | quick | full> : Level of sanity checks for newly created reports
   -F                       : Force remapping of already existing reports
   -f filtfile              : Use specified filter file when doing scoring (-s)
   -s                       : Skip display and output textual score report
   -v                       : Enable verbose mode
   -n                       : Do not include idle thread metric
   -S <mean | merge>        : Aggregation method for summarization results of
                              each configuration (default: merge)
   -T <mean | merge>        : Aggregation method for trace analysis results of
                              each configuration (default: merge)
   -A                       : Post-process every step of a multi-run experiment
```

- Scalasca analysis report explorer (Cube)

Automatic measurement configuration

- scan configures Score-P measurement by automatically setting some environment variables and exporting them
  - E.g., experiment title, profiling/tracing mode, filter file, …
  - Precedence order:
    - Command-line arguments
    - Environment variables already set
    - Automatically determined values
- Also, scan includes consistency checks and prevents corrupting existing experiment directories
- For tracing experiments, after trace collection completes then automatic parallel trace analysis is initiated
  - Uses identical launch configuration to that used for measurement (i.e., the same allocated compute resources)

Recap: Compiler and MPI modules, local installation

- Select appropriate compiler / MPI combination

```
% module load Architecture/KNL   (JURECA booster only!)
% module load Intel IntelMPI
```

- Copy tutorial sources to your scratch directory

```
% jutil env activate -p cjzam11 -A jzam11  (any of your projects)
% mkdir $SCRATCH/$USER
% cd $SCRATCH/$USER
% tar zxvf /p/scratch/share/VI-HPS/examples/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

- VI-HPS tools
  - CUBE release preview installed locally
  - Load environment modules, then load tool modules

```
% module use /p/scratch/share/VI-HPS/JURECA/mf
% module load Score-P Scalasca CubeGUI
```

BT-MZ summary measurement collection...

```
% cd bin.scorep
% cp ../jobscript/jureca/scalasca2.sbatch .
% vim scalasca2.sbatch

# Score-P measurement configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_TOTAL_MEMORY=250M

# Run the application using Scalasca nexus
#export SCAN_ANALYZE_OPTS="--time-correct"
NEXUS="scalasca -analyze -s"
$NEXUS srun $EXE
```

- Change to directory with the executable and edit the job script

```
% sbatch scalasca2.sbatch
```

- Submit the job

BT-MZ summary measurement

```
S=C=A=N: Scalasca 2.5 runtime summarization
S=C=A=N: ./scorep_bt-mz_C_8x6_sum experiment archive
S=C=A=N: Mon Jun 24 11:03:45 2019: Collect start
/usr/bin/srun ./bt-mz_C.8

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) -
    BT-MZ MPI+OpenMP Benchmark

 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8

 [... More application output ...]

S=C=A=N: Mon Jun 24 11:04:07 2019: Collect done (status=0) 22s
S=C=A=N: ./scorep_bt-mz_C_8x6_sum complete.
```

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command
- Creates experiment directory: scorep_bt-mz_C_8x6_sum

BT-MZ summary analysis report examination

- Score summary analysis report

```
% square -s  scorep_bt-mz_C_8x6_sum
INFO: Post-processing runtime summarization result...
INFO: Score report written to ./scorep_bt-mz_C_8x6_sum/scorep.score
```

- Post-processing and interactive exploration with Cube

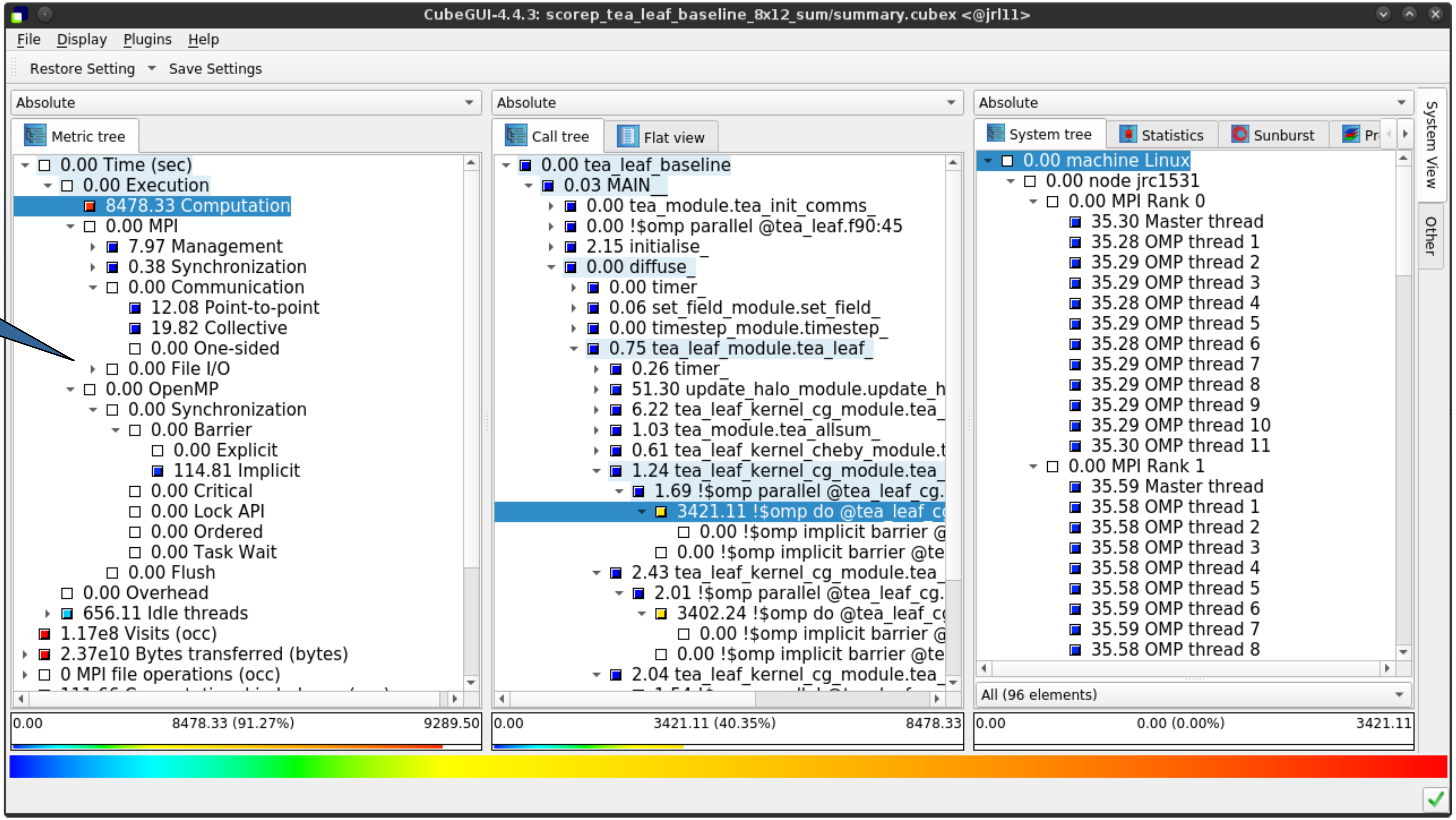
```
% square  scorep_bt-mz_C_8x6_sum
INFO: Displaying ./scorep_bt-mz_C_8x6_sum/summary.cubex...

                    [GUI showing summary analysis report]
```

- The post-processing derives additional metrics and generates a structured metric hierarchy

Post-processed summary analysis report

Split base metrics into more specific metrics



Performance analysis steps

- 0.0 Reference preparation for validation

- 1.0 Program instrumentation
- 1.1 Summary measurement collection
- 1.2 Summary analysis report examination

- 2.0 Summary experiment scoring
- 2.1 Summary measurement collection with filtering
- 2.2 Filtered summary analysis report examination

- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

BT-MZ trace measurement collection...

```
% cd bin.scorep
% cp ../jobscript/jureca/scalasca2.sbatch .
% vim scalasca2.sbatch

# Score-P measurement configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
export SCOREP_TOTAL_MEMORY=250M

# Run the application using Scalasca nexus
export SCAN_ANALYZE_OPTS="--time-correct"
NEXUS="scalasca -analyze -t"
$NEXUS srun $EXE
```

- Change to directory with the executable and edit the job script
- Add "-t" to the scalasca -analyze command

```
% sbatch scalasca2.sbatch
```

- Submit the job

BT-MZ trace measurement ... collection

```
S=C=A=N: Scalasca 2.5 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_C_8x6_trace experiment archive
S=C=A=N: Mon Jun 24 11:22:41 2019: Collect start
srun ./bt-mz_C.8

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP \
>Benchmark

 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8

 [... More application output ...]

S=C=A=N: Mon Jun 24 11:23:04 2019: Collect done (status=0) 23s
```

- Starts measurement with collection of trace files ...

BT-MZ trace measurement ... analysis

```
S=C=A=N: Mon Jun 24 11:23:04 2019: Analyze start
srun scout.hyb --time-correct ./scorep_bt-mz_C_8x6_trace/traces.otf2
SCOUT   (Scalasca 2.5)

Analyzing experiment archive ./scorep_bt-mz_C_8x6_trace/traces.otf2

Opening experiment archive ... done (0.006s).
Reading definition data    ... done (0.008s).
Reading event trace data   ... done (0.419s).
Preprocessing              ... done (0.341s).
Timestamp correction       ... done (0.753s).
Analyzing trace data       ... done (9.230s).
Writing analysis report    ... done (0.255s).

Max. memory usage          : 844.727MB

        # passes         : 1
        # violated       : 0

Total processing time      : 11.112s
S=C=A=N: Mon Jun 24 11:23:18 2019: Analyze done (status=0) 14s
```

- Continues with automatic (parallel) analysis of trace files

BT-MZ trace analysis report exploration

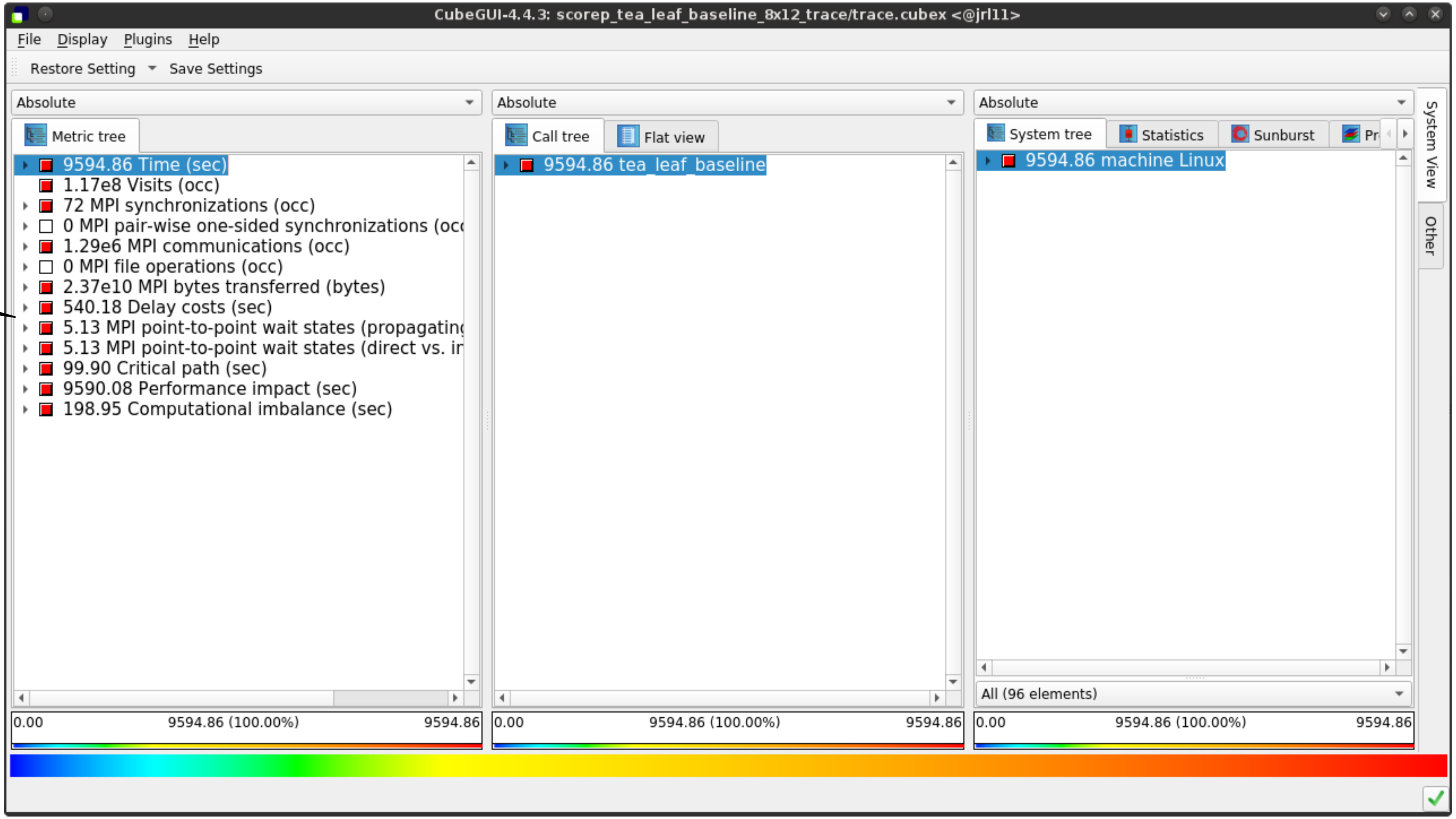
- Produces trace analysis report in the experiment directory containing trace-based wait-state metrics

```
% square  scorep_bt-mz_C_8x6_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report...
INFO: Displaying ./scorep_bt-mz_C_8x6_trace/trace.cubex...

              [GUI showing trace analysis report]
```

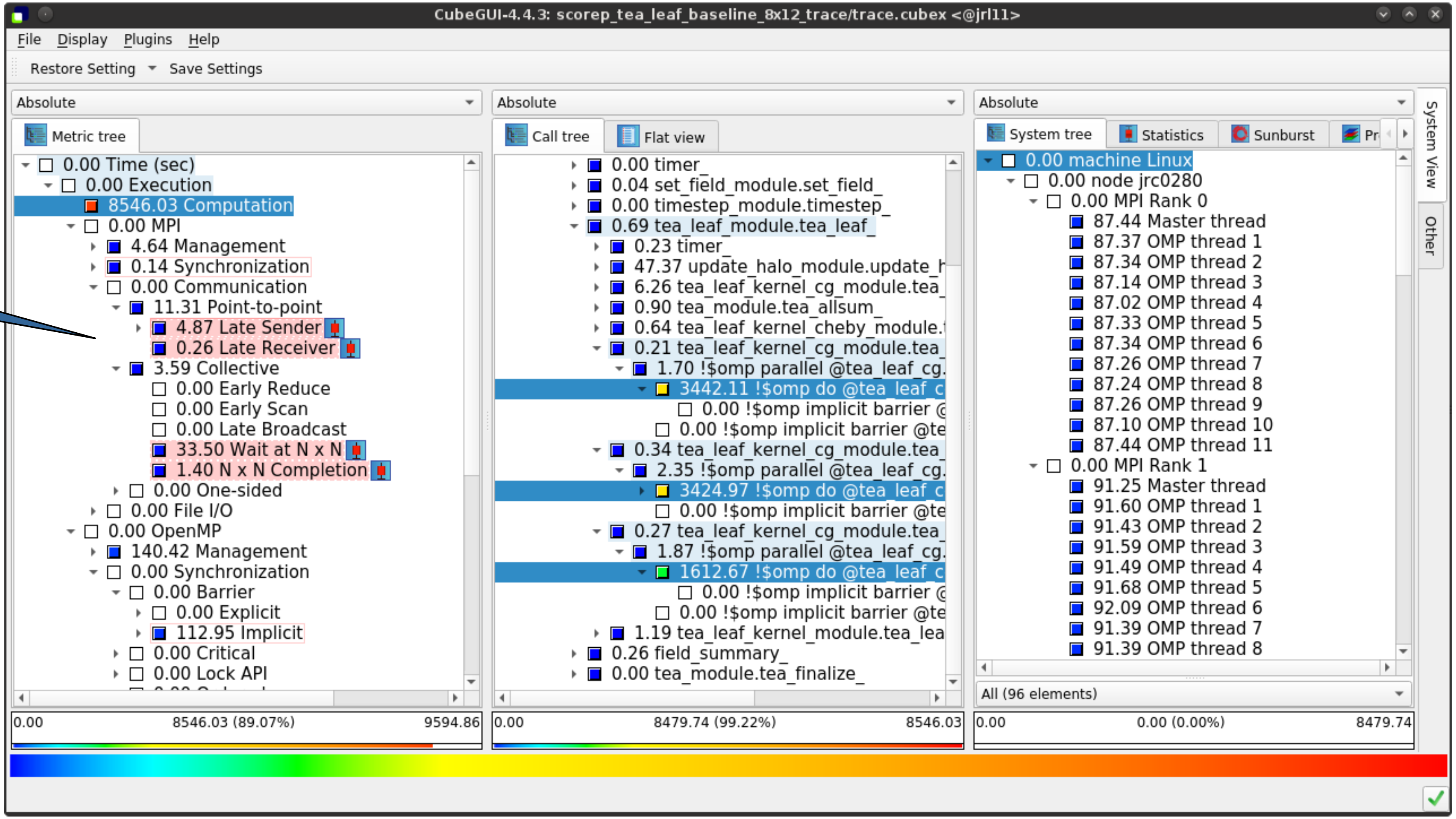
Scalasca analysis report exploration (opening view)

Additional top-level metrics produced by the trace analysis…



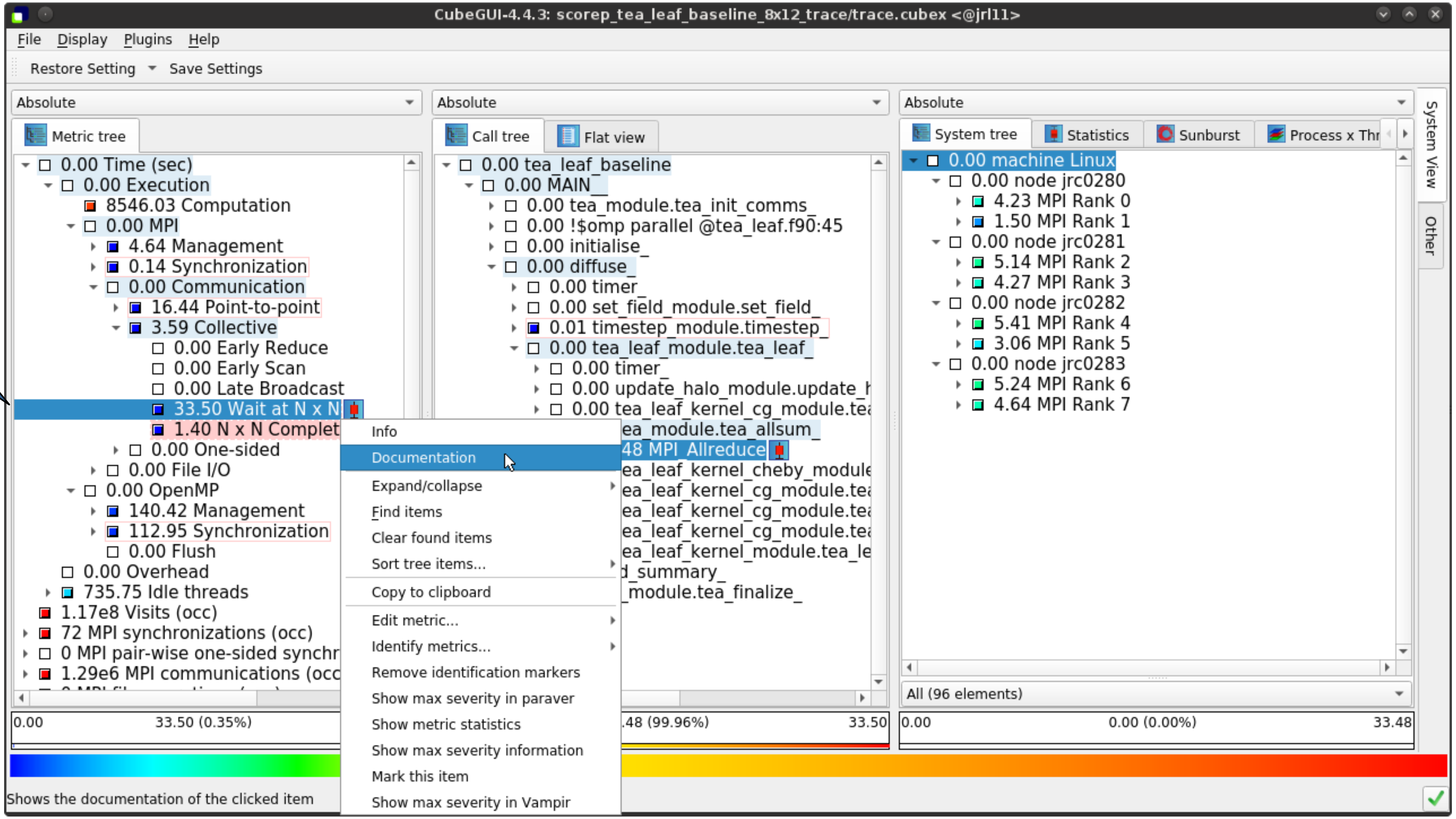
Scalasca wait-state metrics

…plus additional wait-state metrics as part of the “Time” hierarchy



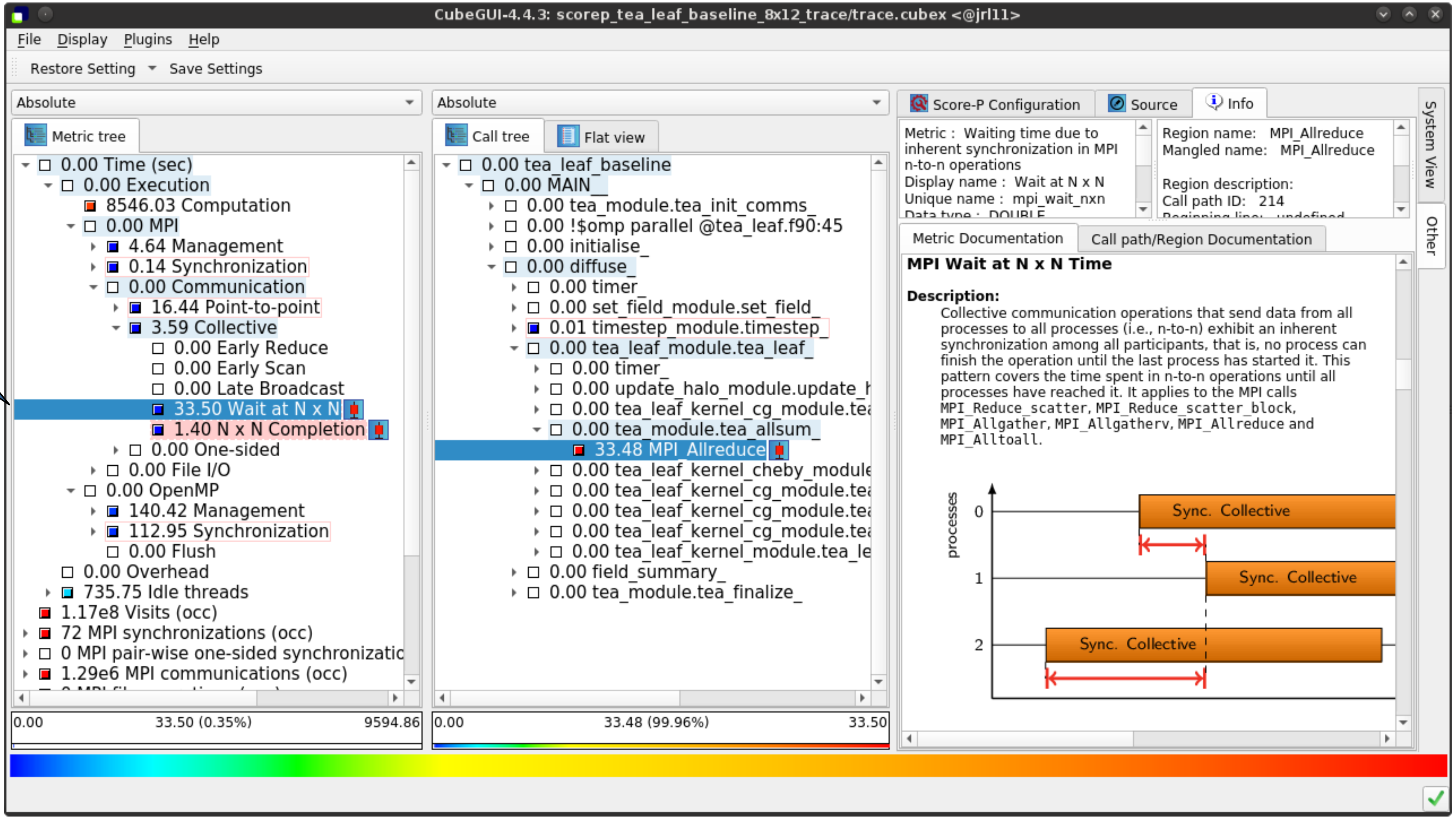
Online metric description

Access online metric description via context menu (right-click)



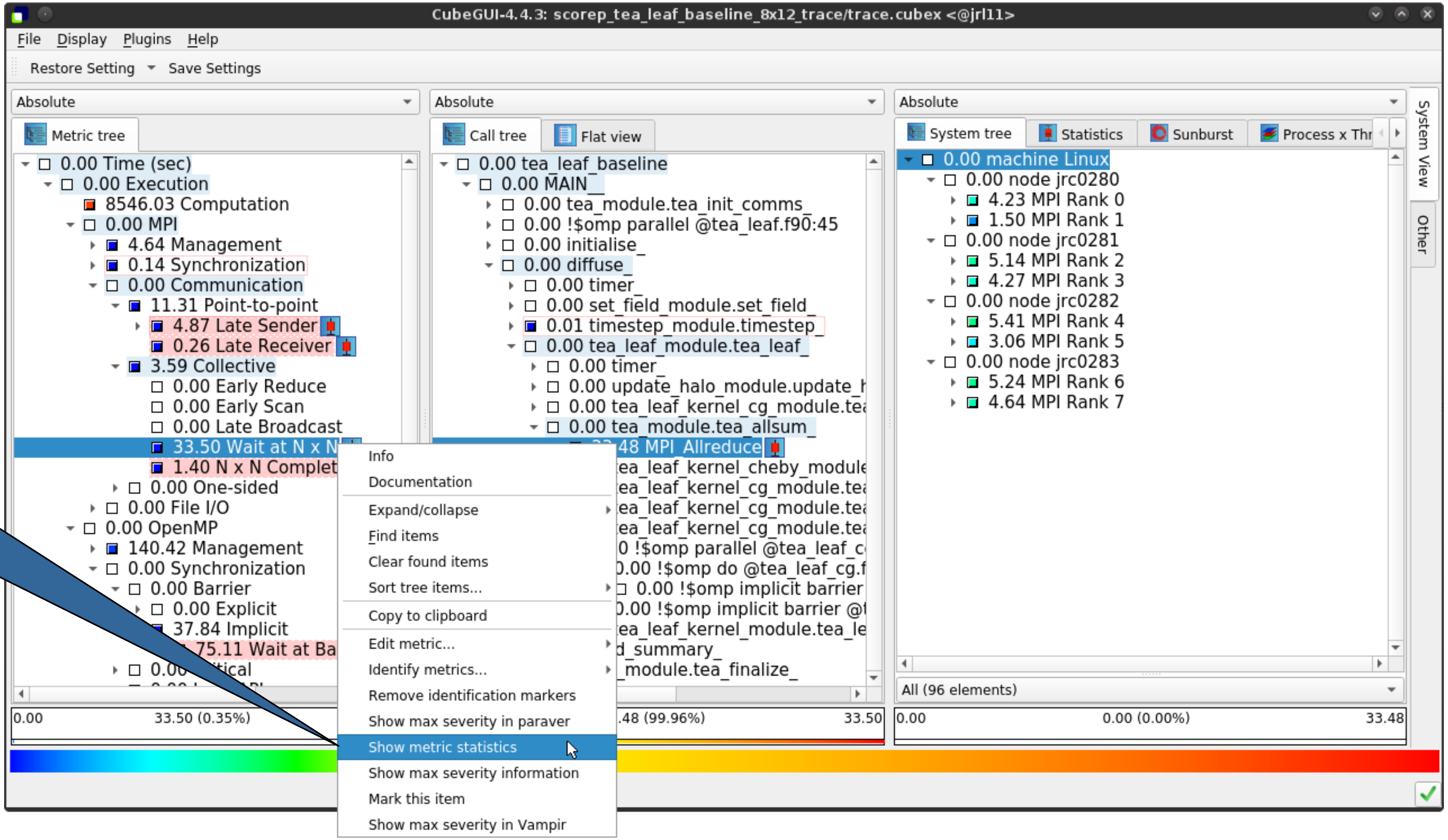
Online metric description (cont.)

Selection of different metric automatically updates description



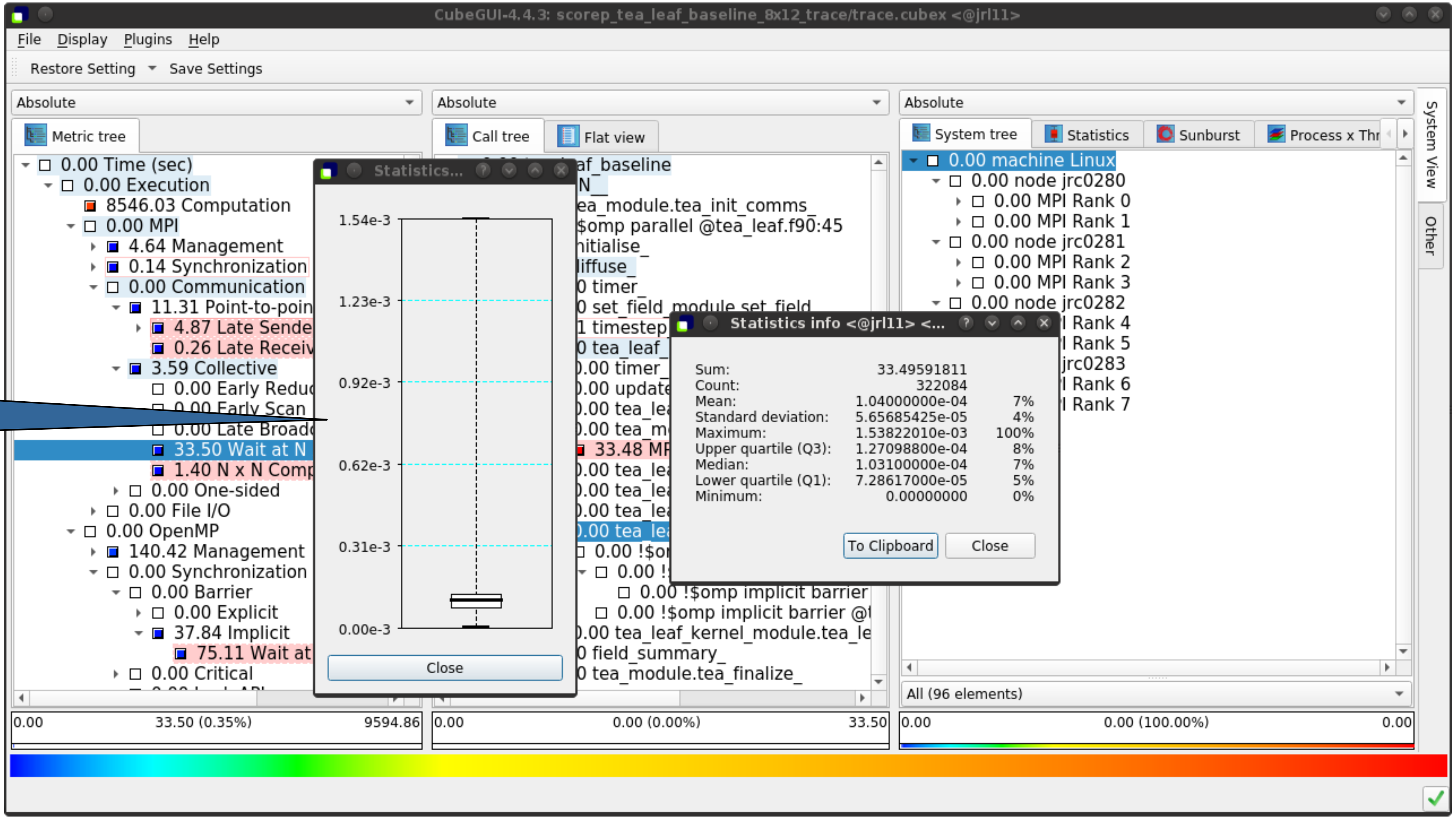
Metric statistics

Access metric statistics for metrics marked with box plot icon from context menu



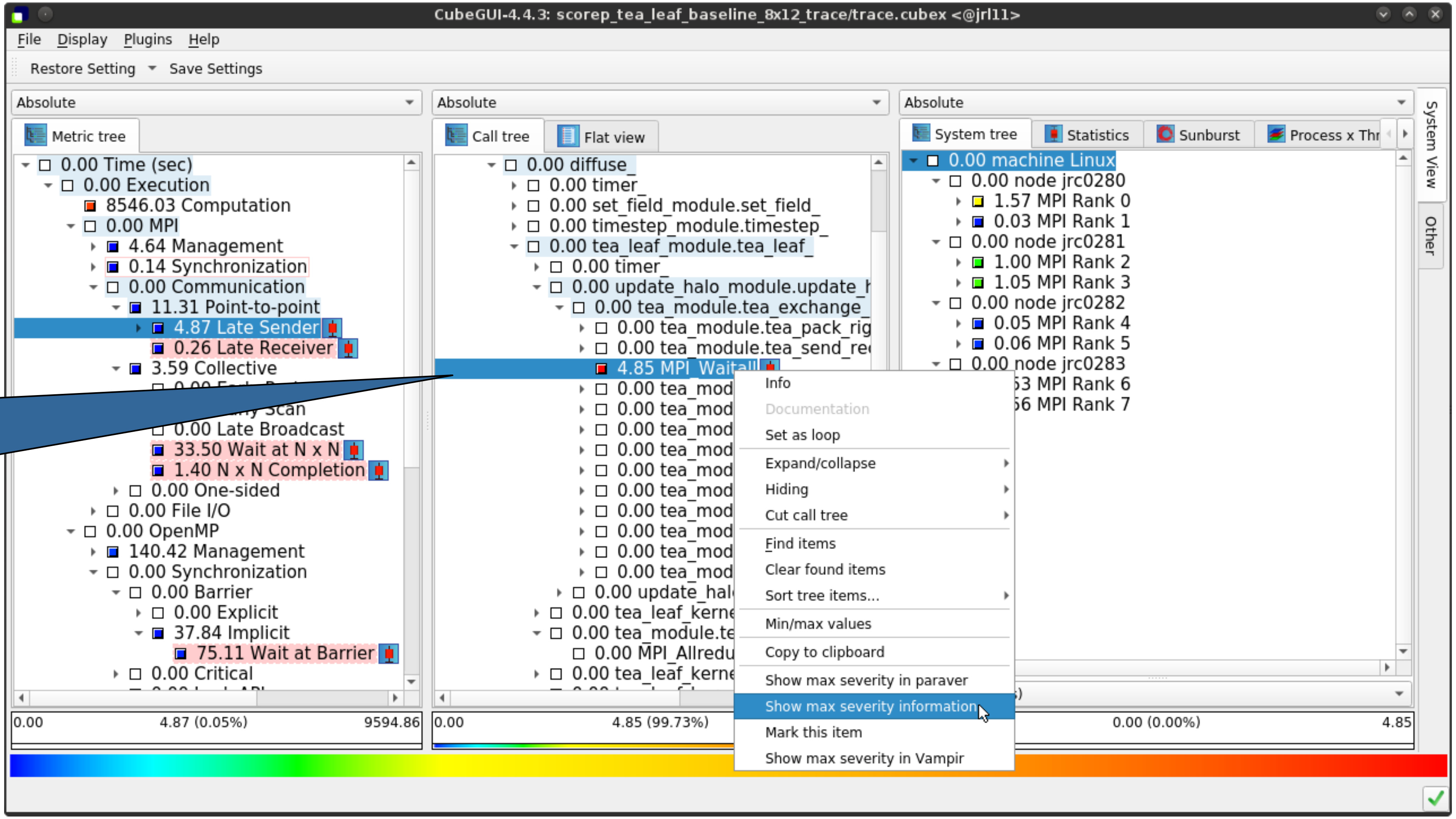
Metric statistics (cont.)

Shows instance statistics box plot, click to get details



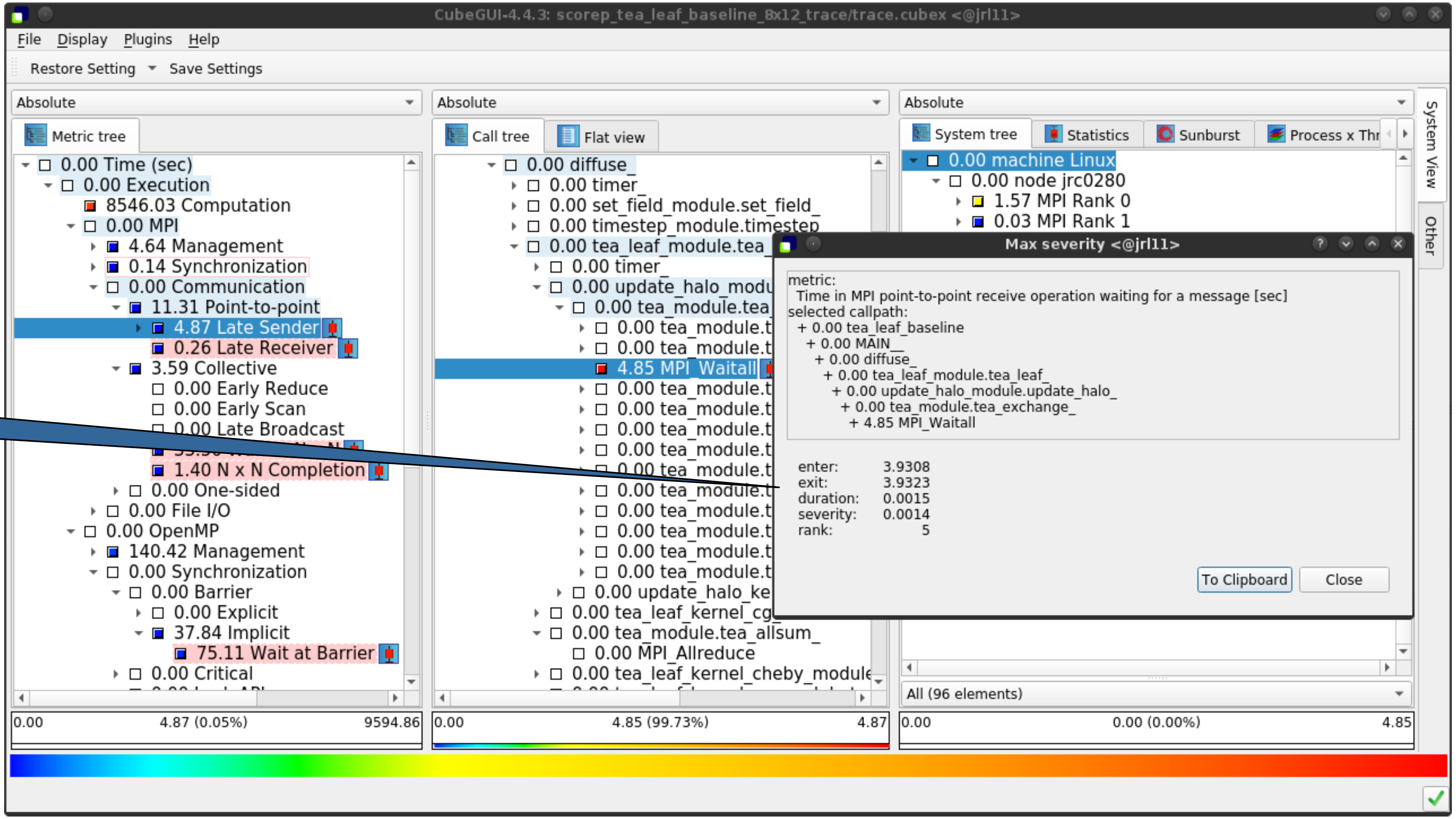
Metric instance statistics

Access most-severe instance information for call paths marked with box plot icon via context menu



Metric instance statistics (cont.)

Shows instance details



Scalasca Trace Tools: Further information

- Collection of trace-based performance tools
  - Specifically designed for large-scale systems
  - Features an automatic trace analyzer providing wait-state, critical-path, and delay analysis
  - Supports MPI, OpenMP, POSIX threads, and hybrid MPI+OpenMP/Pthreads
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - http://www.scalasca.org
- Contact:
  - mailto: scalasca@fz-juelich.de