

# Automatic trace analysis with the Scalasca Trace Tools

---

Marc Schlüter  
Jülich Supercomputing Centre



# Scalasca Trace Tools

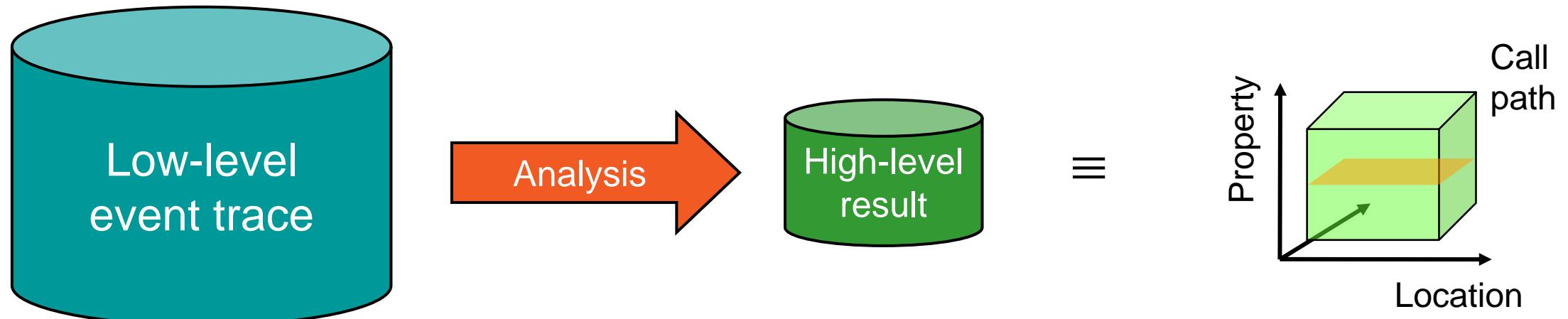
DOI [10.5281/zenodo.1512589](https://doi.org/10.5281/zenodo.1512589)

- **Scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
  - Analysis of traces including *only host-side events* from applications using CUDA, OpenCL, or OpenACC (also in combination with MPI and/or OpenMP) is possible, but results need to be interpreted with some care
- Specifically targeting large-scale parallel applications
  - Demonstrated scalability up to 1.8 million parallel threads
  - Of course also works at small/medium scale
- Latest release:
  - Scalasca Trace Tools v2.6.2 (April 2025)

# Automatic trace analysis

- Idea

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



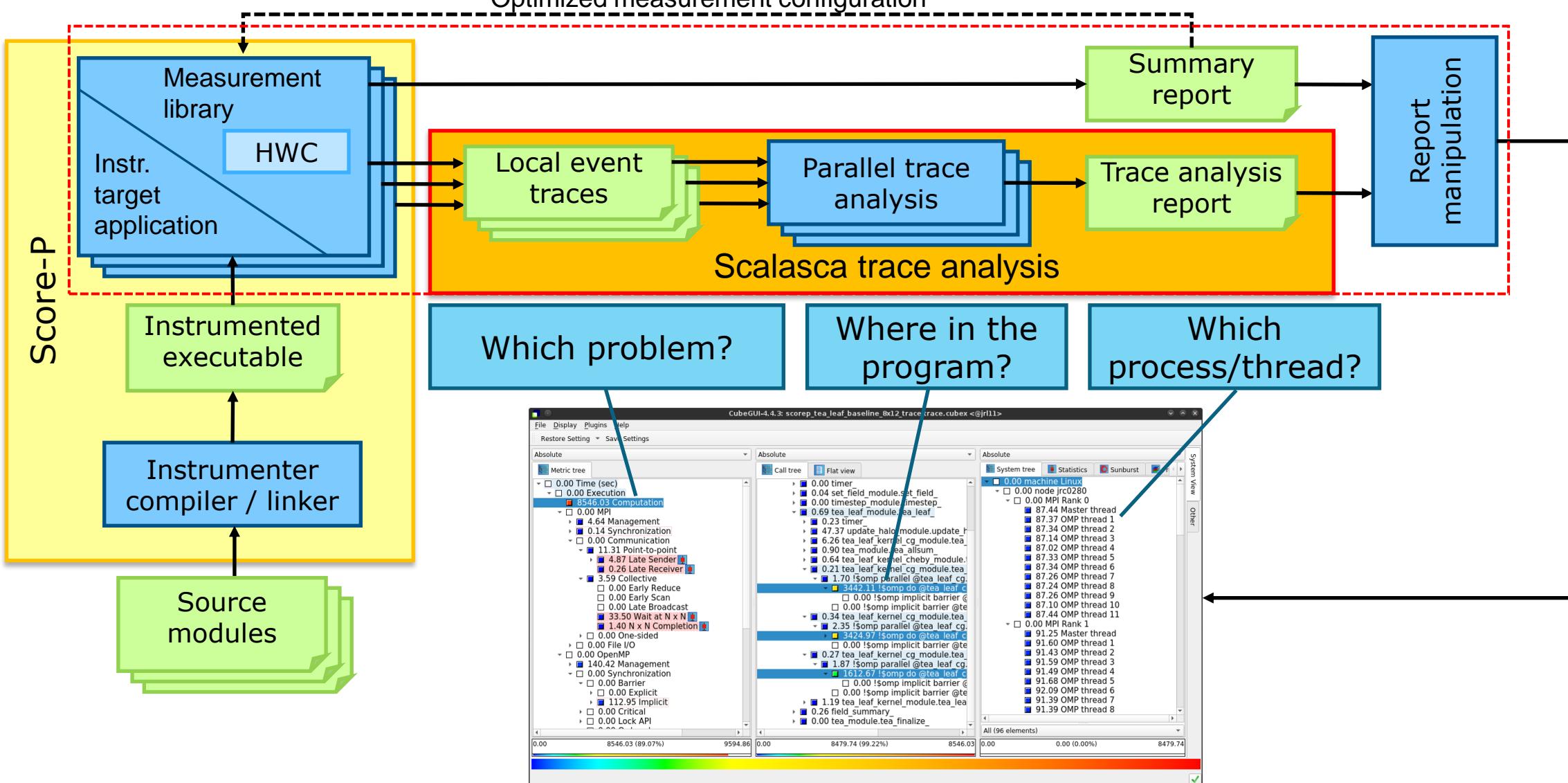
- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

# Scalasca Trace Tools: Features

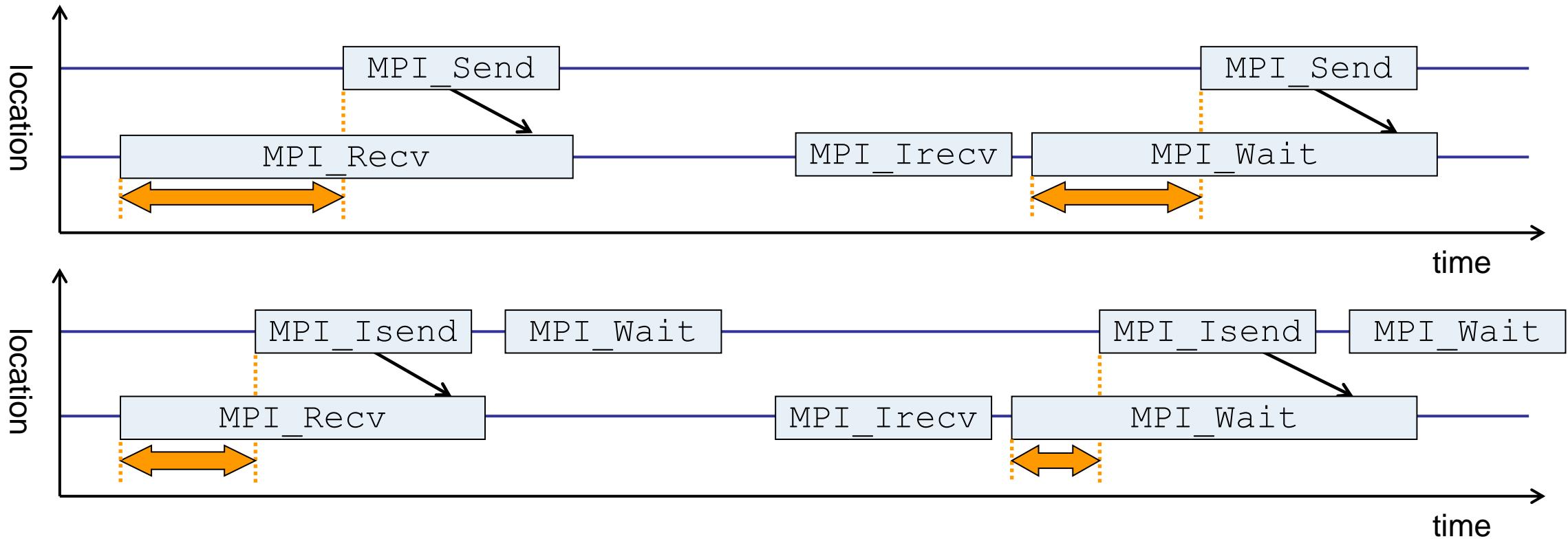
---

- Open source, 3-clause BSD license
- Supports all major HPC platforms
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Provides convenience commands for measurement, analysis, and post-processing
  - 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 memory events, CUDA/OpenCL device events (kernel, memcpy), SHMEM, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored

# Scalasca workflow

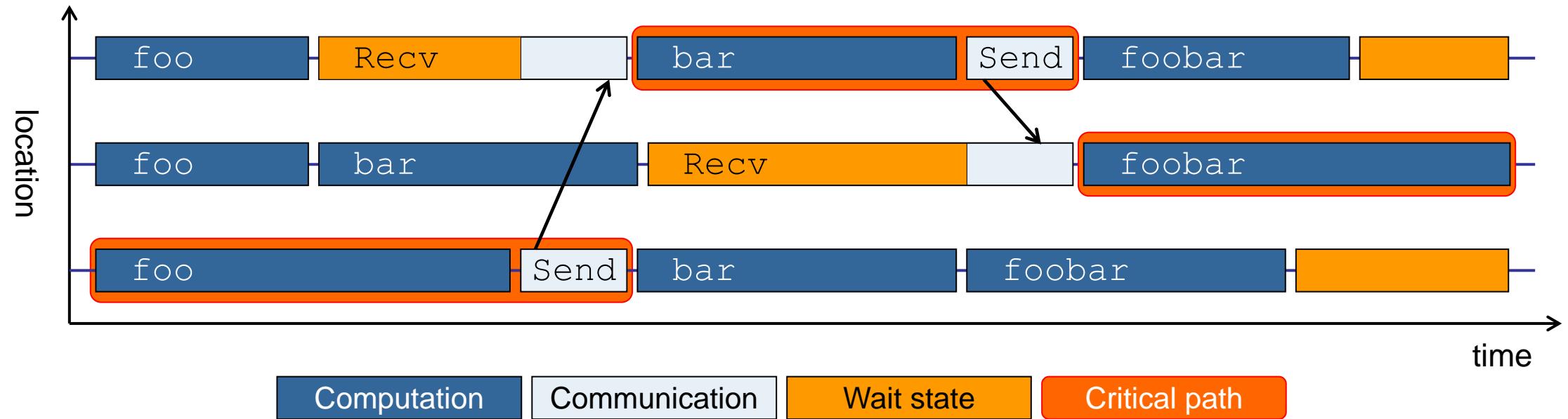


## Example: “*Late Sender*” wait state



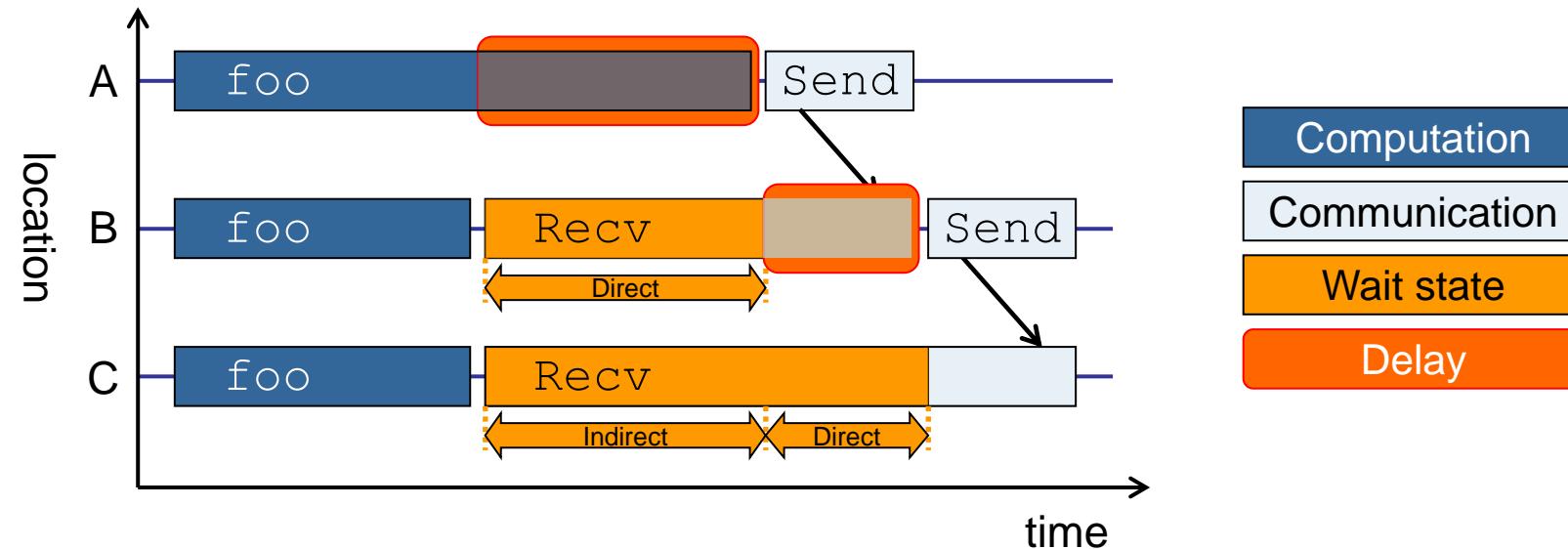
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

## Example: Critical path

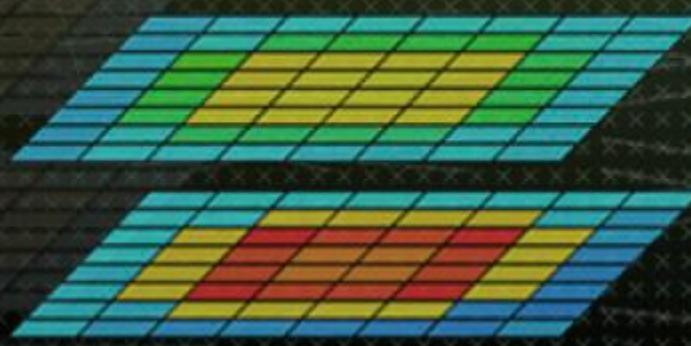


- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks

## 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*



## Hands-on: NPB-MZ-MPI / BT

---

trace tools   
scalasca

## Recap: Setup for exercises

---

- Connect to your account on the local training system
  - MN5 GPP:

```
% ssh -Y <user>@glogin1.bsc.es
```

- Change to directory containing BT-MZ sources
  - Existing instrumented executable can be reused

```
% module purge
% module load gcc/12.3.0 openmpi/4.1.5-gcc12.3 scorep/9.4-gcc-ompi papi/7.1.0-gcc
% module load cubelib/4.9.1 scalasca/2.6.2-gcc-ompi
% cd /gpfs/scratch/nct_362/users/<yourName>/hands-on/scorep/NPB-BT-MZ/bin.scorep
```

# Demo: BT-MZ summary measurement collection...

```
% vi ../jobscript/marenostrum5/scalasca.slurm

#SBATCH -J filter          <= change this to "scalasca"

# Measurement configuration
export SCOREP_FILTERING_FILE=initial_scorep.filter
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_TOTAL_MEMORY=150M
#export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
scalasca -analyze srun ./bt-mz_${CLASS}.${PROCS}
```

- Change to the *bin.scorep* directory and edit the job script

**Hint:**  
`scan` = scalasca -analyze  
`-s` = profile/summary  
(default)

- Submit the job

```
% sbatch ../jobscript/marenostrum5/scalasca.slurm
```

# scan: 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)

## Demo: BT-MZ summary measurement

```
S=C=A=N: Scalasca 2.6.2 runtime summarization
S=C=A=N: ./scorep_bt-mz_D_32x7_sum experiment archive
S=C=A=N: Thu Feb  5 17:19:17 2026: Collect start
/usr/bin/srun ./bt-mz_D.32

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

Number of zones: 32 x 32
Iterations: 250      dt:  0.000020
Number of active processes: 32

[... More application output ...]

S=C=A=N: Thu Feb  5 17:20:26 2026: Collect done (status=0) 69s
S=C=A=N: ./scorep_bt-mz_D_32x7_sum complete.
```

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command
- Creates experiment directory:  
`scorep_bt-mz_D_32x7_sum`

# Demo: BT-MZ summary analysis report examination

- Score summary analysis report

```
% square -s scorep_bt-mz_D_32x7_sum
INFO: Post-processing runtime summarization report (profile.cubex) ...
INFO: Score report written to scorep_bt-mz_D_32x7_sum/scorep.score
```

- Post-processing and interactive exploration with Cube

```
% square scorep_bt-mz_D_32x7_sum
INFO: Displaying scorep_bt-mz_D_32x7_sum/summary.cubex
```

[GUI showing summary analysis report]

**Hint:**  
Copy 'summary.cubex' to local system (laptop) using 'scp' to improve responsiveness of GUI

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

# BT-MZ trace measurement collection...

```
% vi ../jobscript/marenostrum5/scalasca.slurm

#SBATCH -J scalasca          <= change this to "trace"

# Measurement configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
export SCOREP_TOTAL_MEMORY=250M
#export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
scalasca -analyze -t  srun ./bt-mz_${CLASS}.${PROCS}

% sbatch ../jobscript/marenostrum5/scalasca.slurm
```

- Change to the top-level directory and edit the job script
- Add “-t” to the scan command
- Submit the job

## BT-MZ trace measurement ... collection

---

```
S=C=A=N: Scalasca 2.6.2 trace collection and analysis
S=C=A=N: ./scorep_bt-mz D 32x7 trace experiment archive
S=C=A=N: Thu Feb 5 17:31:40 2026: Collect start
/usr/bin/srun ./bt-mz_D.32

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

Number of zones: 32 x 32
Iterations: 250      dt: 0.000020
Number of active processes: 32

[... More application output ...]

S=C=A=N: Thu Feb 5 17:32:51 2026: Collect done (status=0) 71s
```

- Starts measurement with collection of trace files ...

# BT-MZ trace measurement ... analysis

```
...  
S=C=A=N: Thu Feb  5 17:32:51 2026: Analyze start  
/usr/bin/srun /apps/GPP/.../bin/scout.hyb \  
./scorep_bt-mz_D_32x7_trace/traces.otf2  
  
SCOUT      (Scalasca 2.6.2)  
  
Analyzing experiment archive ./scorep_bt-mz_D_32x7_trace/traces.otf2  
  
Opening experiment archive ... done (0.003s).  
Reading definition data    ... done (0.005s).  
Reading event trace data  ... done (0.273s).  
Preprocessing                ... done (0.466s).  
Analyzing trace data       ... done (16.404s).  
Writing analysis report     ... done (0.160s).  
  
Max. memory usage          : 1276.949MB  
  
Total processing time       : 17.468s  
S=C=A=N: Thu Feb  5 17:33:15 2026: Analyze done (status=0) 24s
```

- Continues with automatic (parallel) analysis of trace files

## BT-MZ trace analysis report exploration

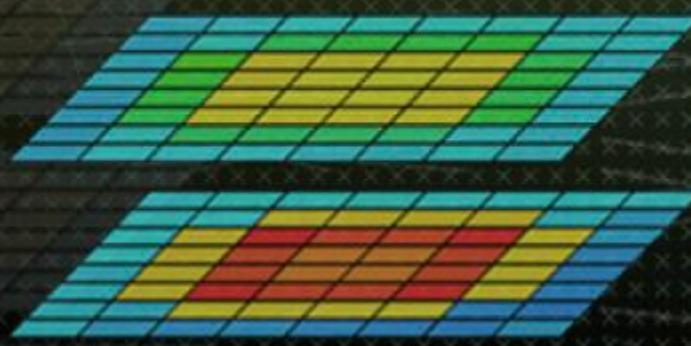
- Produces trace analysis report in the experiment directory containing trace-based wait-state metrics

```
% square scorep_bt-mz_D_32x7_trace  
INFO: Post-processing runtime summarization report (profile.cubex) ...  
INFO: Post-processing trace analysis report (scout.cubex) ...  
INFO: Displaying scorep_bt-mz_D_32x7_trace /trace.cubex...
```

[GUI showing trace analysis report]

### Hint:

Run ‘square -s’ first and then copy ‘trace.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI



## Case study: TeaLeaf MPI+OpenMP

---

trace tools   
scalasca

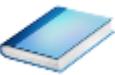
# Case study: TeaLeaf MPI+OpenMP

- 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: <https://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”:  $4000 \times 4000$  cells, CG solver

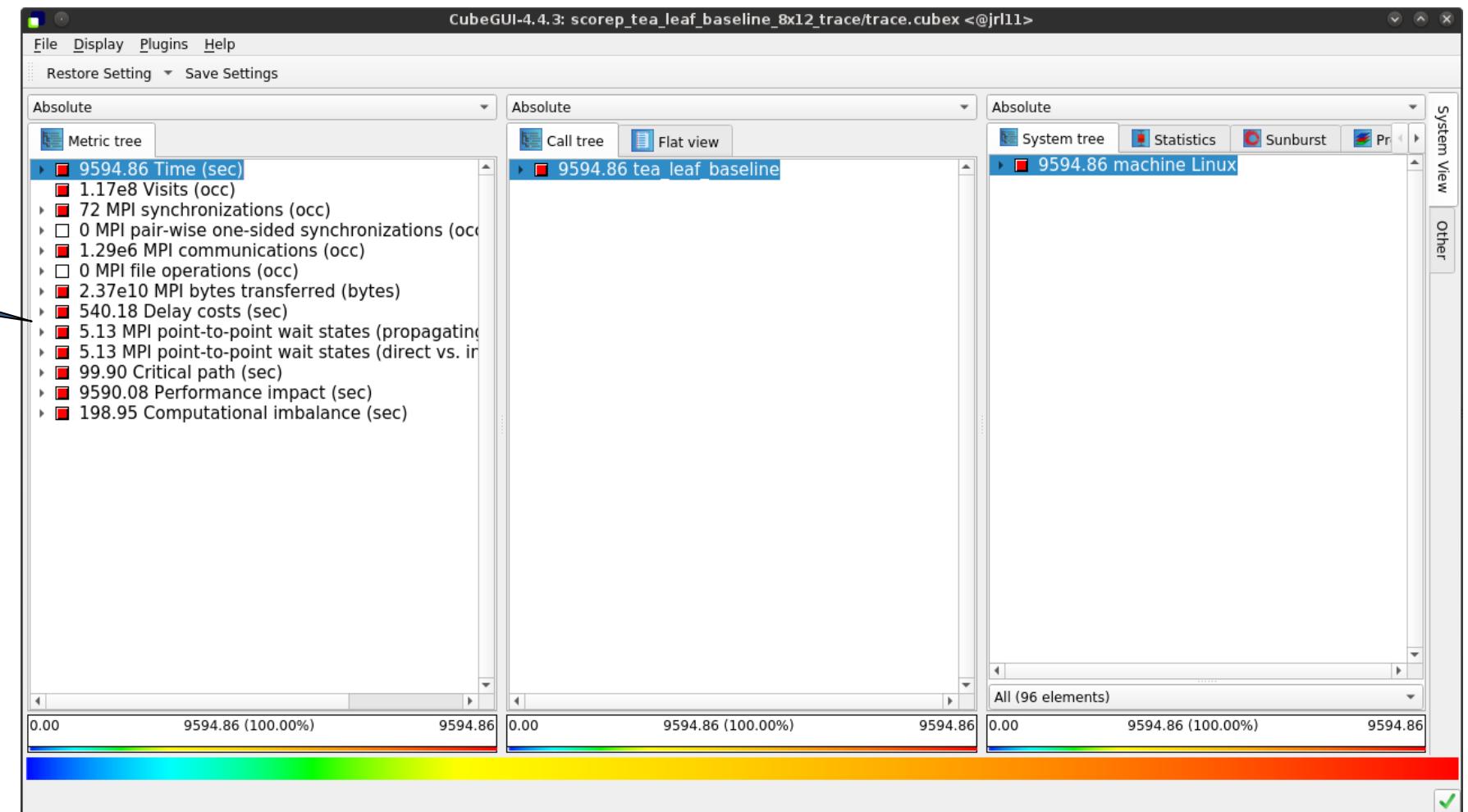


```
% cube scorep_tea_leaf_baseline_8x12_trace/trace.cubex
[GUI showing post-processed trace analysis report]
```

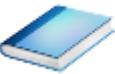
# Scalasca analysis report exploration (opening view)



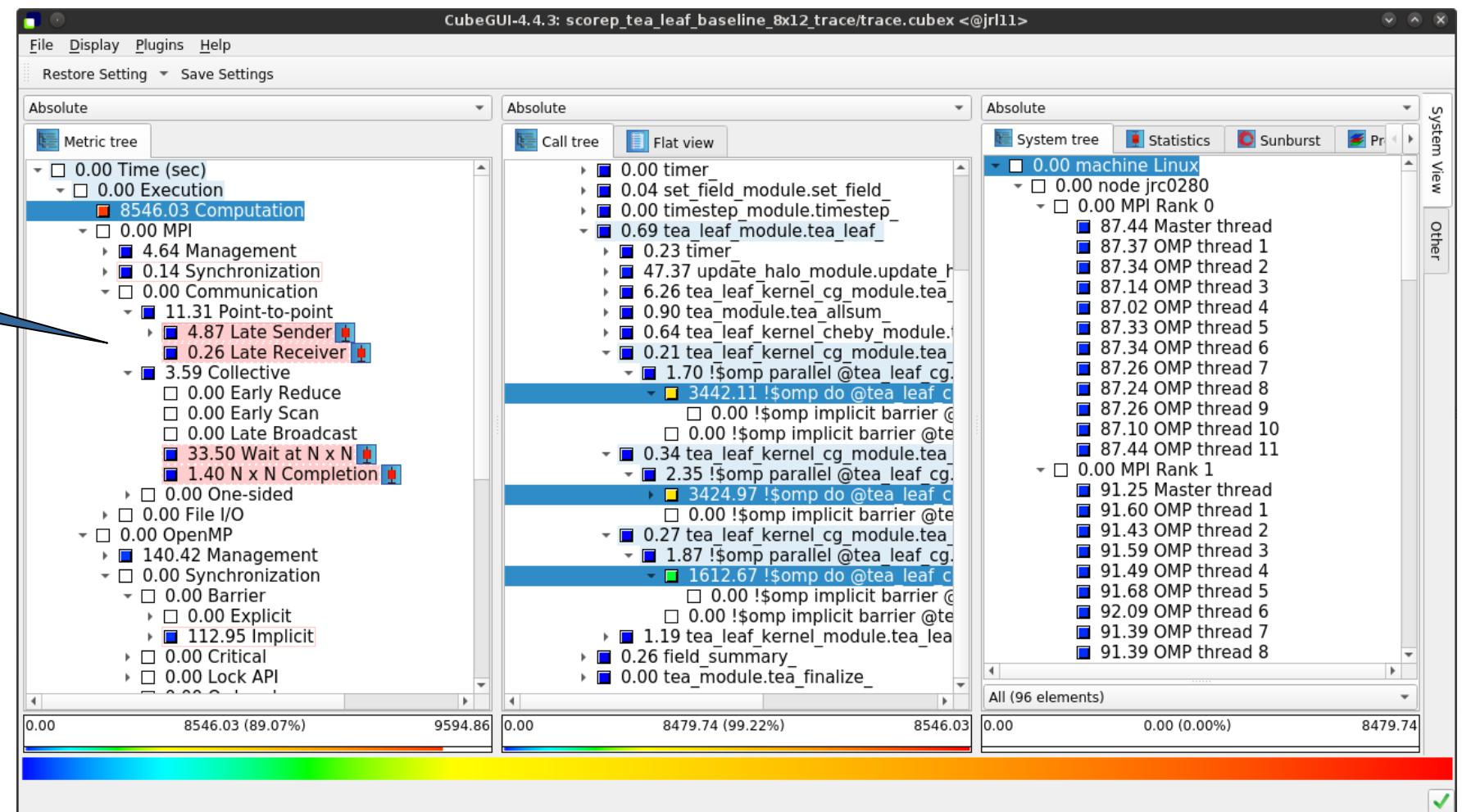
Additional top-level metrics produced by the trace analysis...

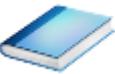


# Scalasca wait-state metrics



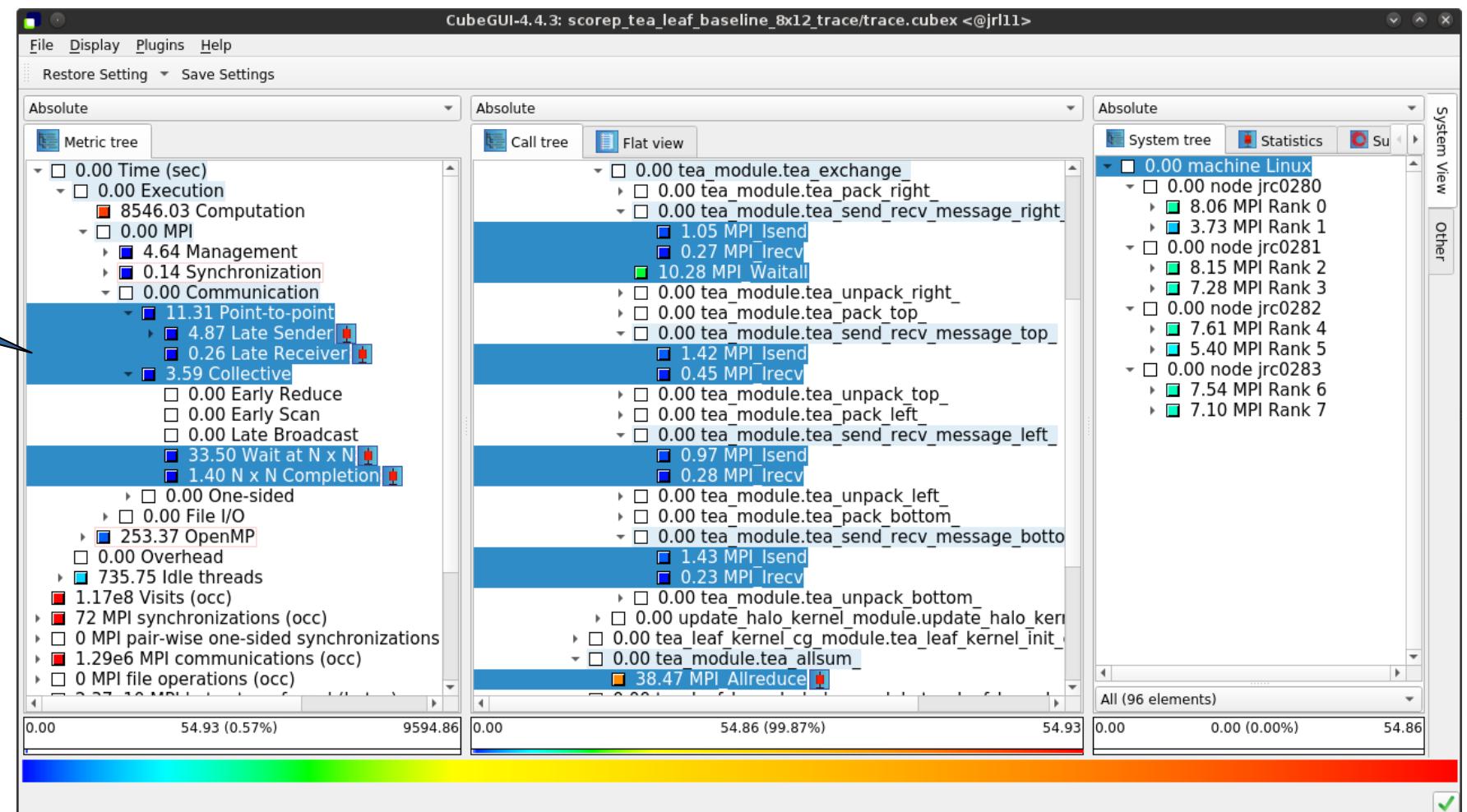
...plus additional wait-state metrics as part of the “Time” hierarchy



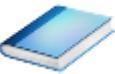


# TeaLeaf Scalasca report analysis (I)

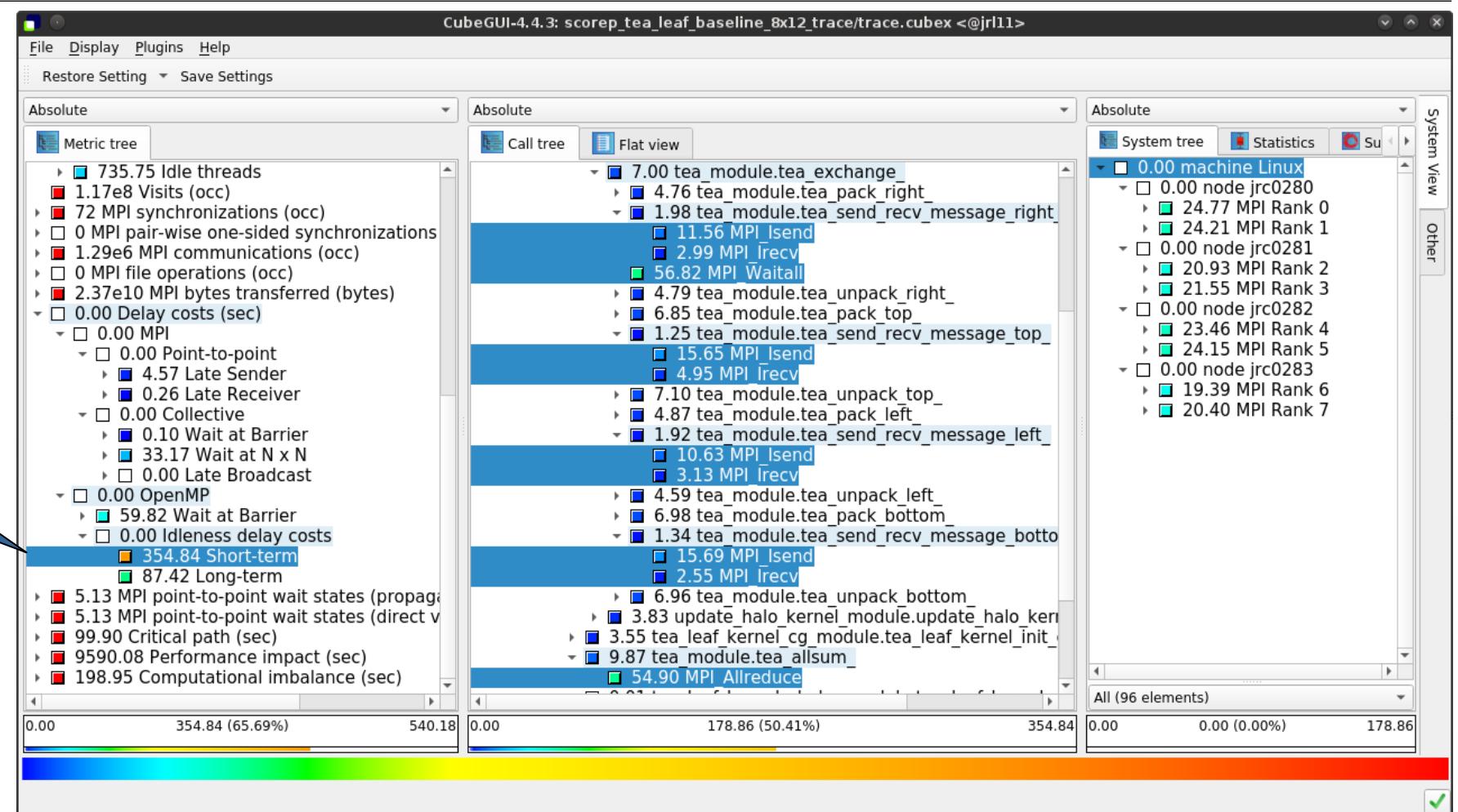
While MPI communication time and wait states are small (~0.6% of the total execution time)...

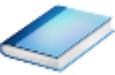


# TeaLeaf Scalasca report analysis (II)



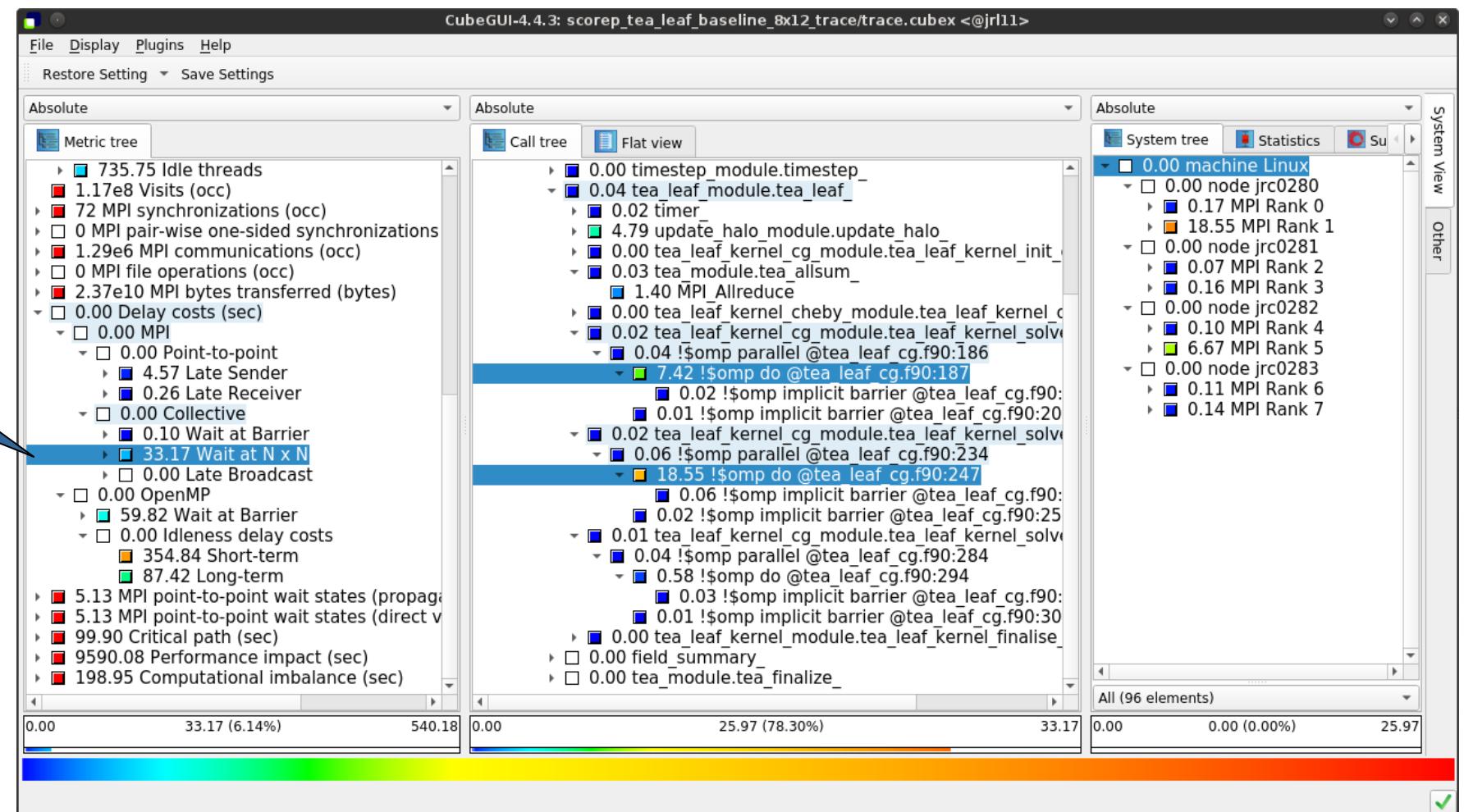
...they directly cause a significant amount of the OpenMP thread idleness

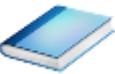




# TeaLeaf Scalasca report analysis (III)

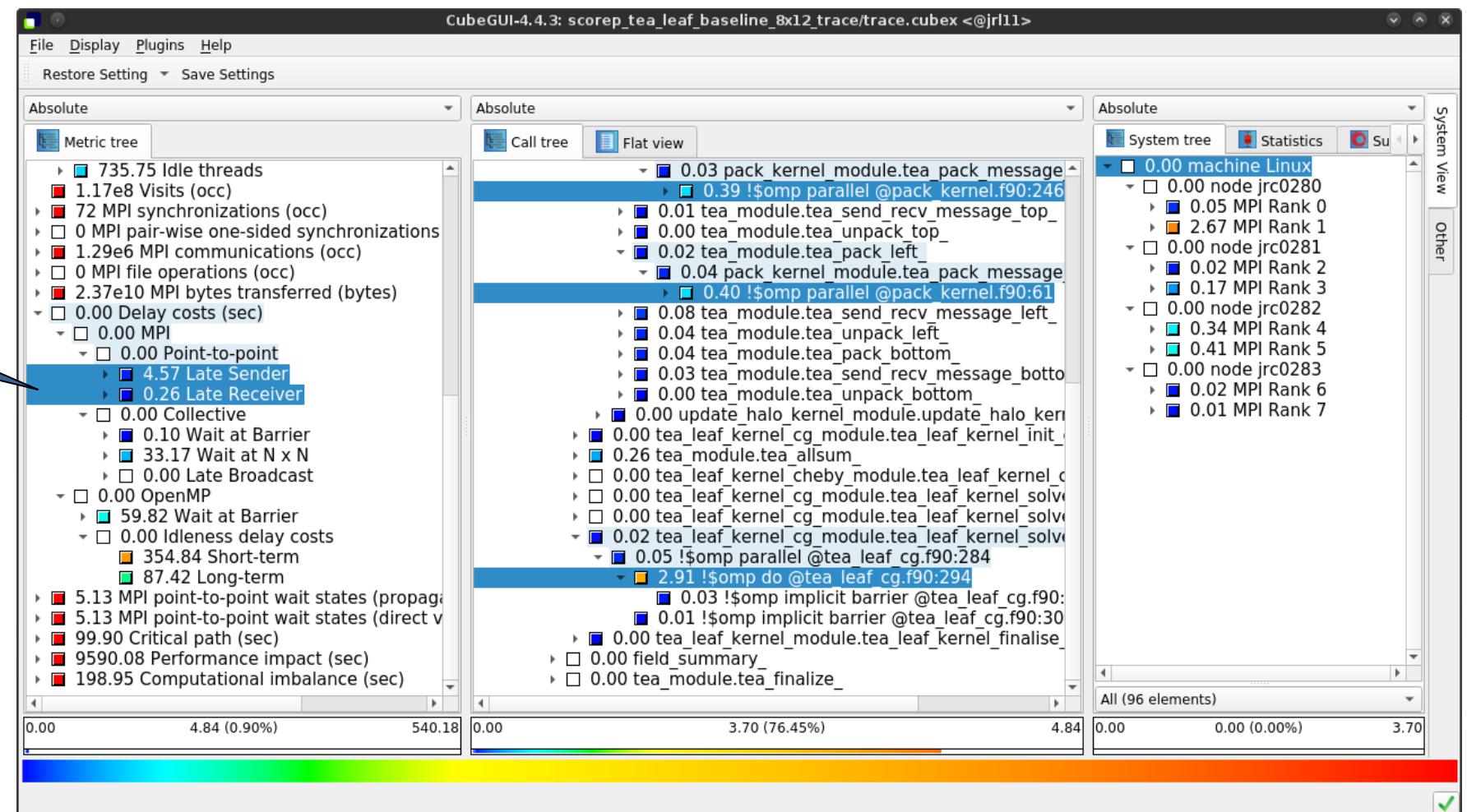
The “Wait at NxN” collective wait states are mostly caused by the first 2 OpenMP do loops of the solver (on ranks 5 & 1, resp.)...



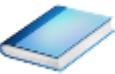


# TeaLeaf Scalasca report analysis (IV)

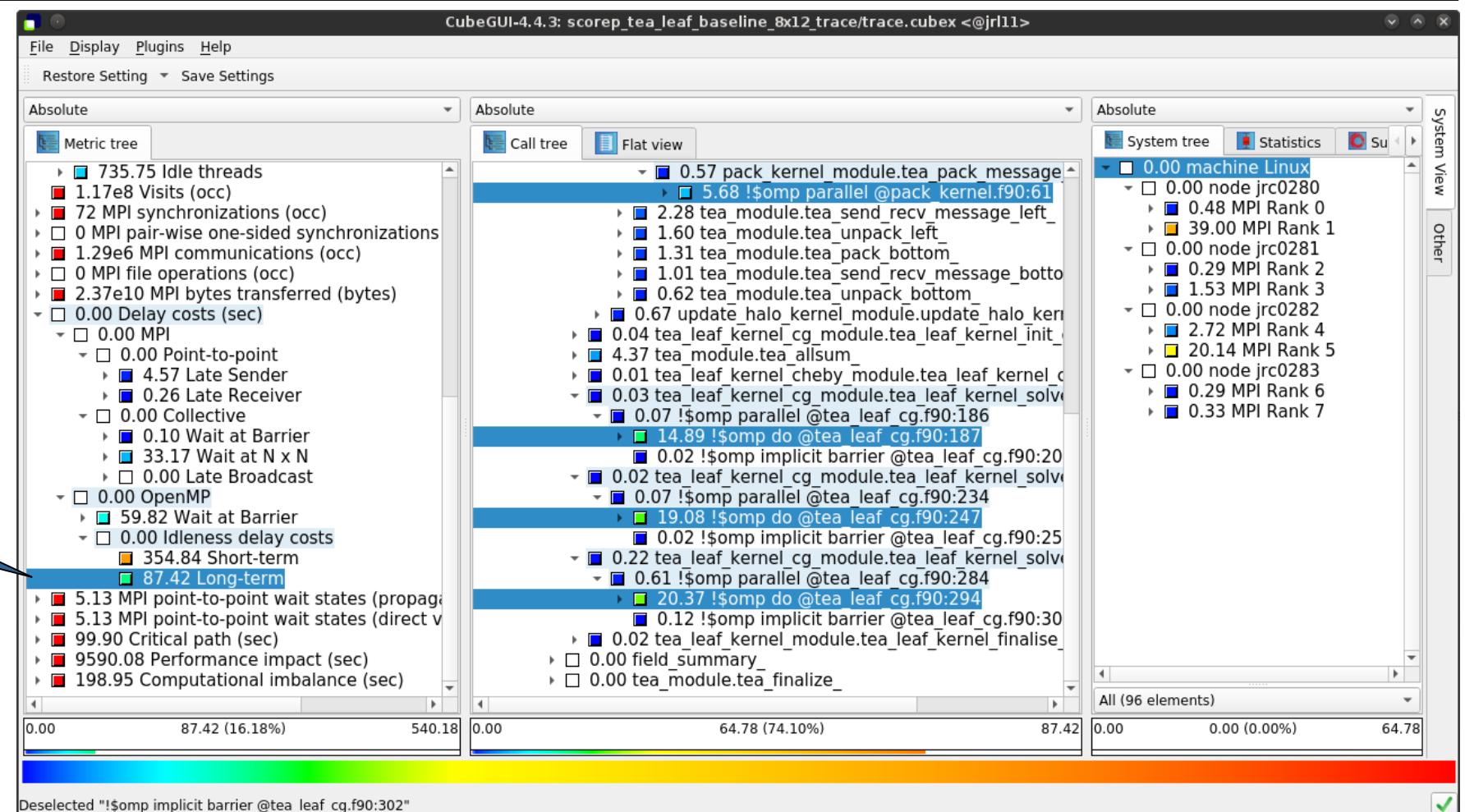
...while the MPI point-to-point wait states are caused by the 3<sup>rd</sup> solver do loop (on rank 1) and two loops in the halo exchange



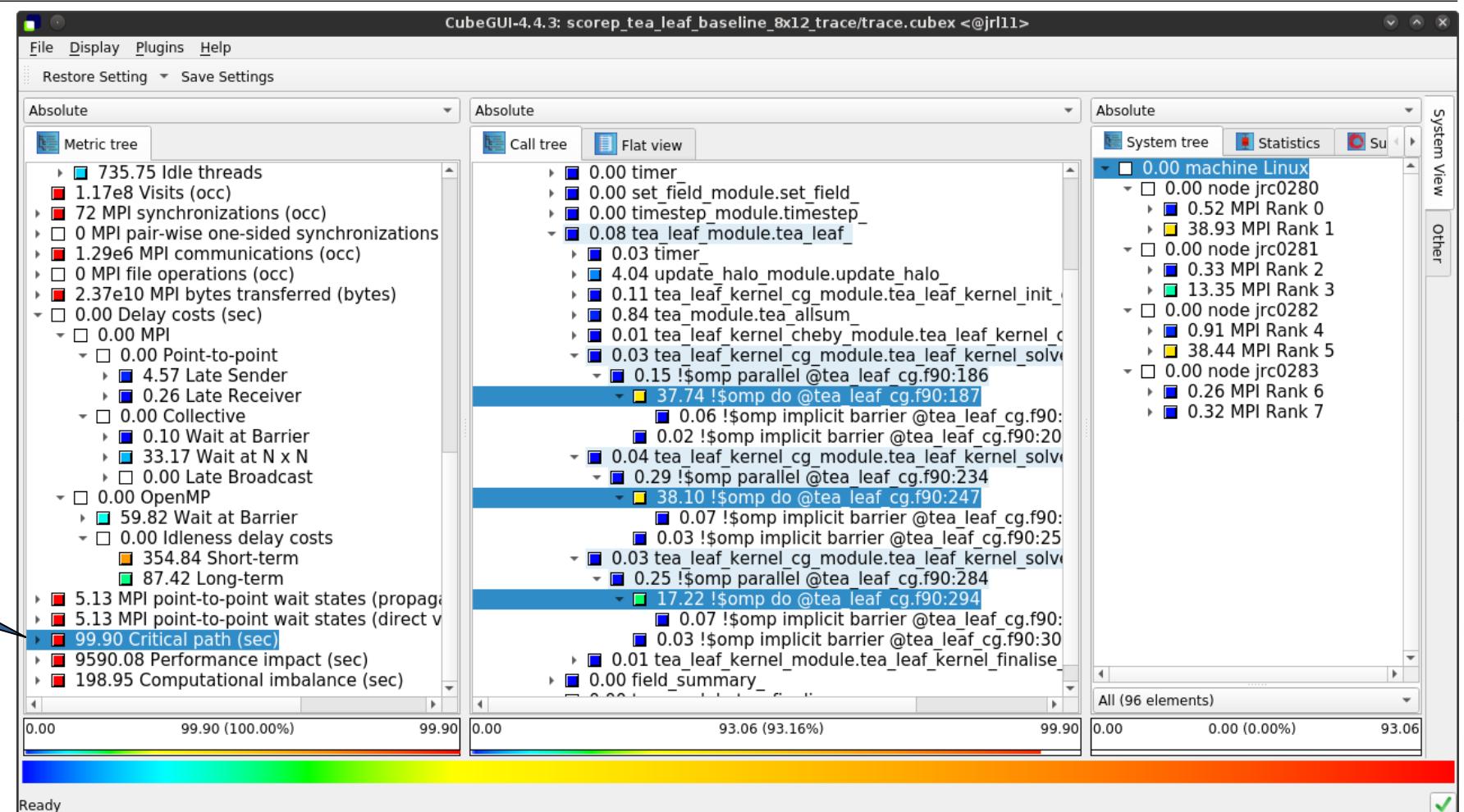
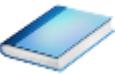
# TeaLeaf Scalasca report analysis (V)

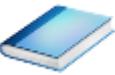


Various OpenMP do loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation

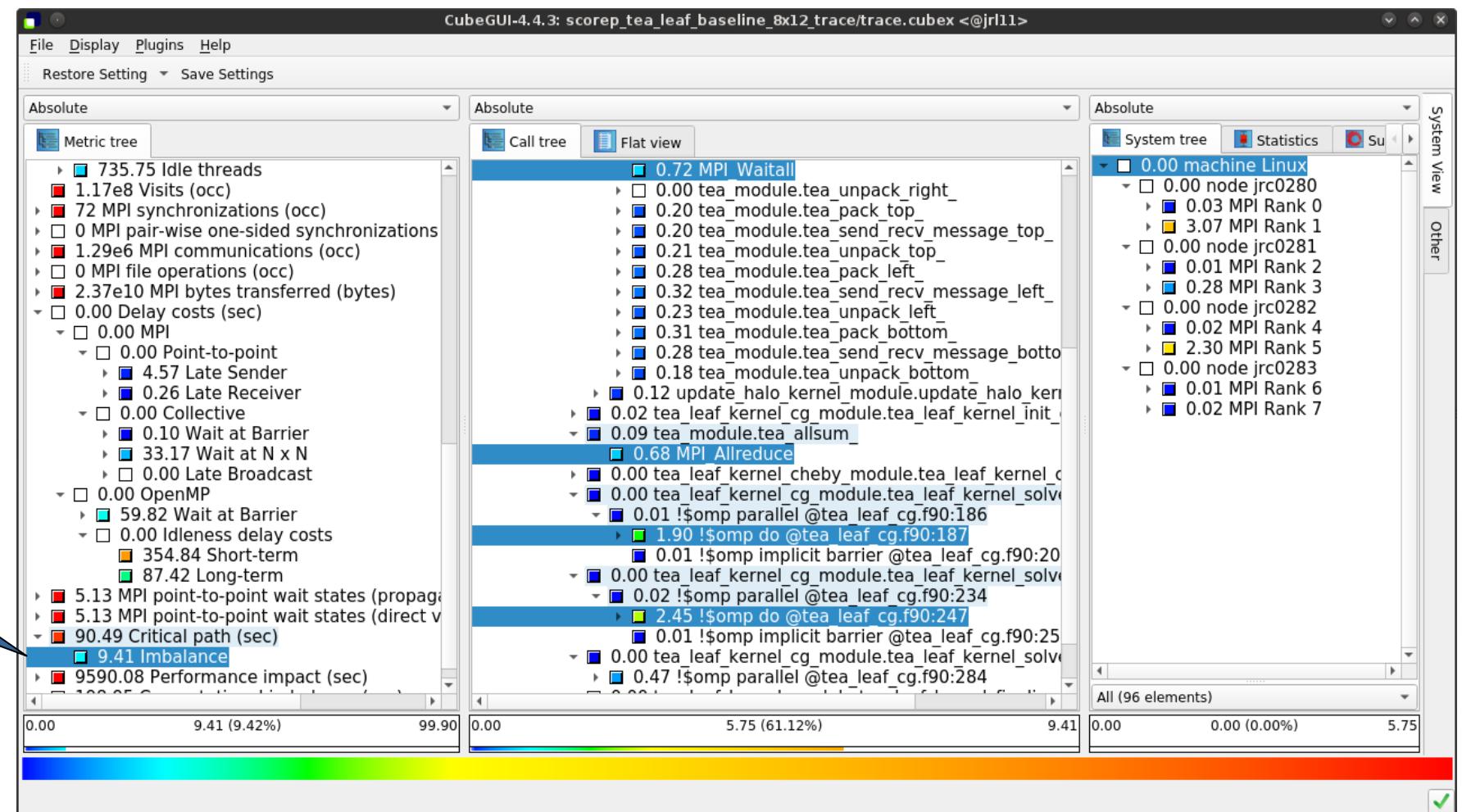


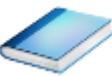
# TeaLeaf Scalasca report analysis (VI)





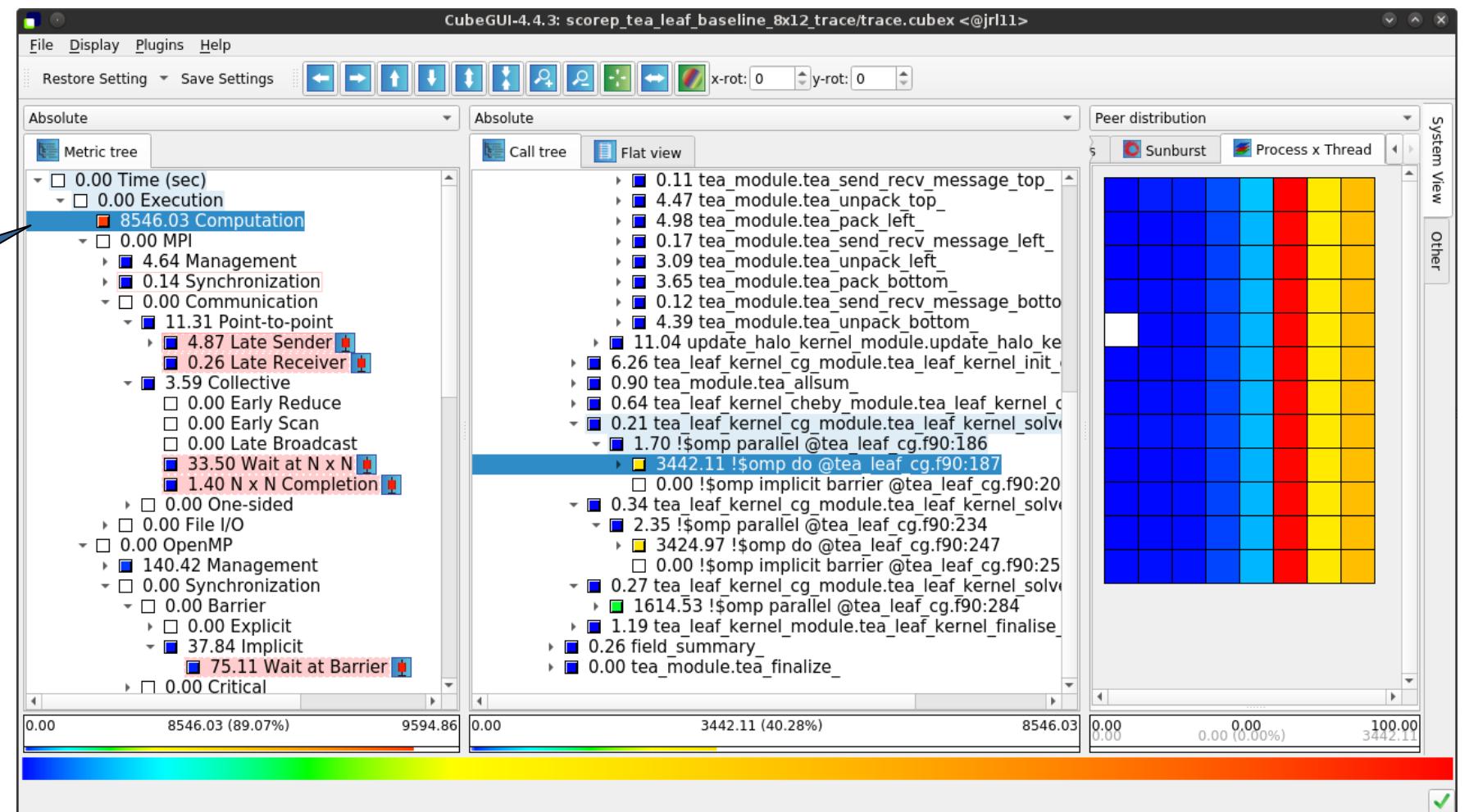
# TeaLeaf Scalasca report analysis (VII)



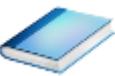


# TeaLeaf Scalasca report analysis (VIII)

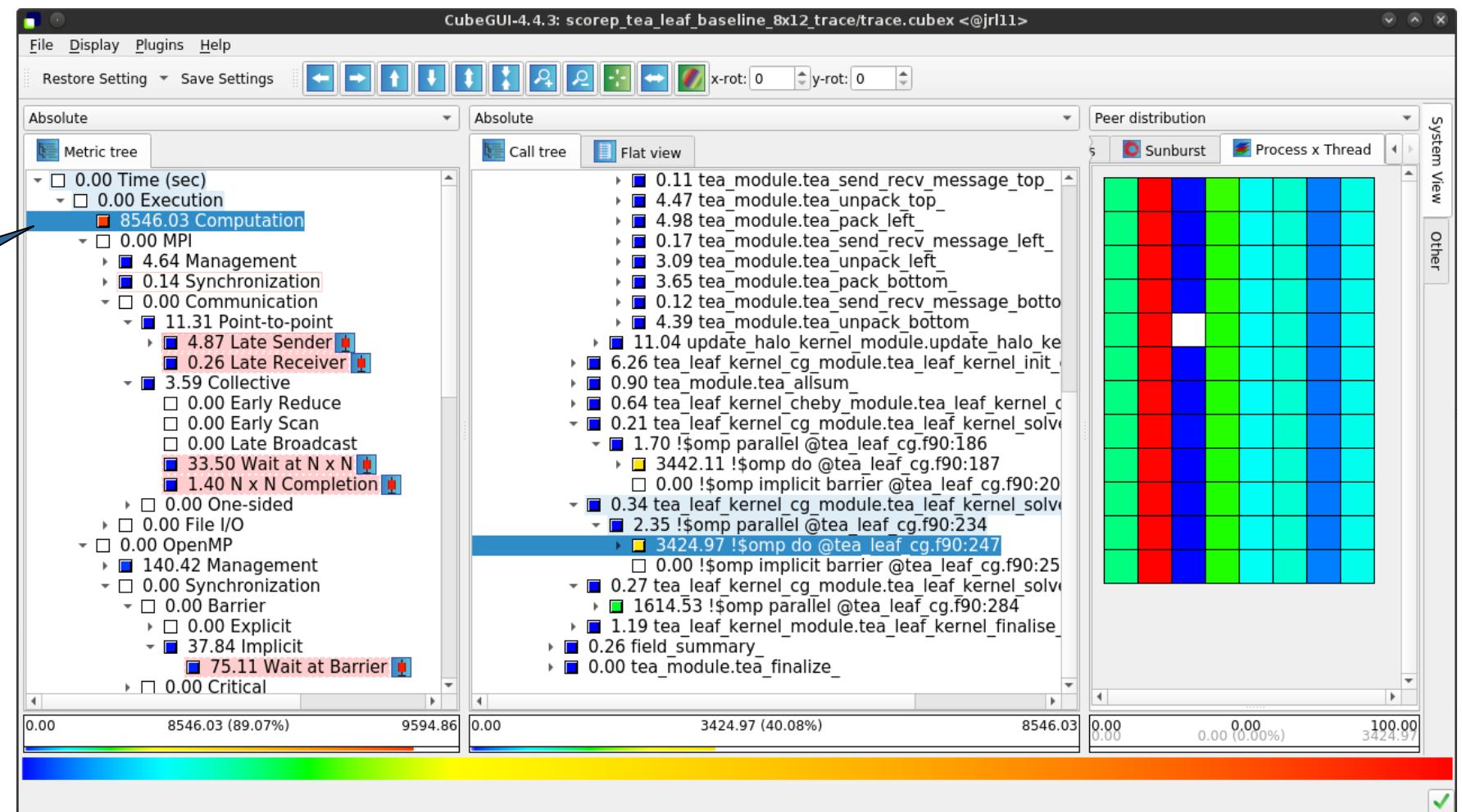
## Computation time of 1<sup>st</sup> ...



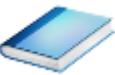
# TeaLeaf Scalasca report analysis (IX)



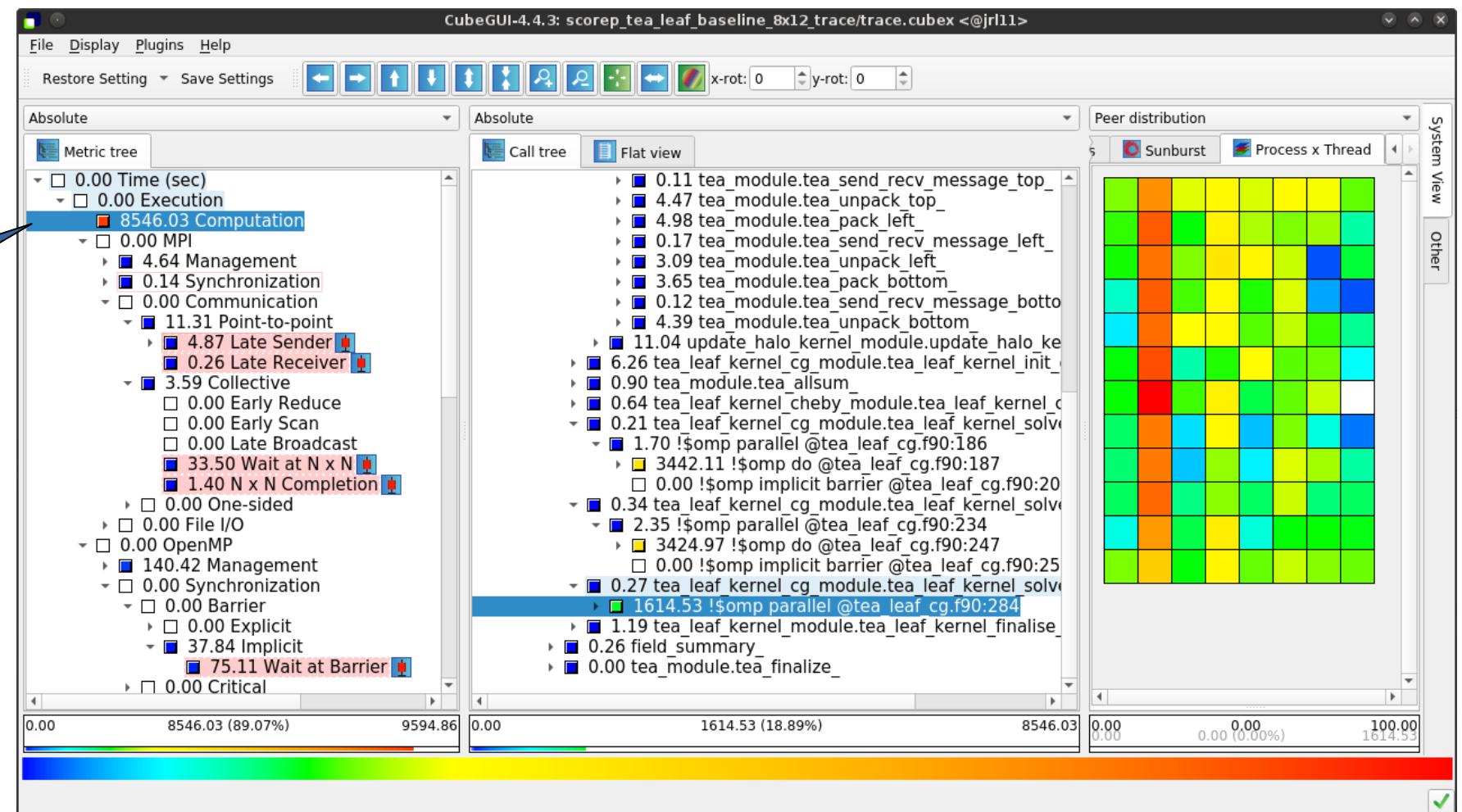
...and 2<sup>nd</sup> do loop  
mostly balanced within  
each rank, but vary  
considerably across  
ranks...



# TeaLeaf Scalasca report analysis (X)



...while the 3<sup>rd</sup> 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%**

# 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:
  - <https://www.scalasca.org>
- Contact:
  - mailto: [scalasca@fz-juelich.de](mailto:scalasca@fz-juelich.de)

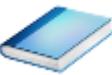


## Reference material

---

trace tools   
scalasca

# Scalasca command – One command for (almost) everything



```
% scalasca
Scalasca 2.6.2
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 will be removed in the next major release  
⇒ use Score-P instrumenter directly

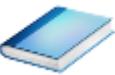
# Scalasca convenience command: scan / scalasca -analyze



```
% scan
Scalasca 2.6.2: 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.
  -P preset    : Specify a preset for a multi-run measurement, e.g., 'pop'.
  -L          : List available multi-run presets.
  -D cfgfile   : Check a multi-run config file for validity and dump
                  the processed configuration for comparison.
```

## ▪ Scalasca measurement collection & analysis nexus

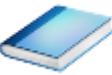
# Scalasca convenience command: square / scalasca -examine



```
% square
Scalasca 2.6.2: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
  -C <none | quick | full>      : Level of sanity checks for newly created reports
  -c <number>                   : Consider number of counters when doing scoring (-s)
  -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
  -I                           : Ignore structural sanity checks and force aggregation
                                of measurements in a multi-run experiment
  -x <scorep-score opt>       : Pass option(s) to scorep-score
```

- Scalasca analysis report explorer (Cube)

# Scalasca advanced command: scout - Scalasca automatic trace analyzer

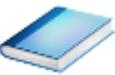


```
% scout.hyb --help
SCOUT      (Scalasca 2.6.2)
Copyright (c) 1998-2022 Forschungszentrum Juelich GmbH
Copyright (c) 2014-2021 RWTH Aachen University
Copyright (c) 2009-2014 German Research School for Simulation Sciences GmbH

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`

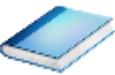


- Scalasca trace event timestamp consistency correction

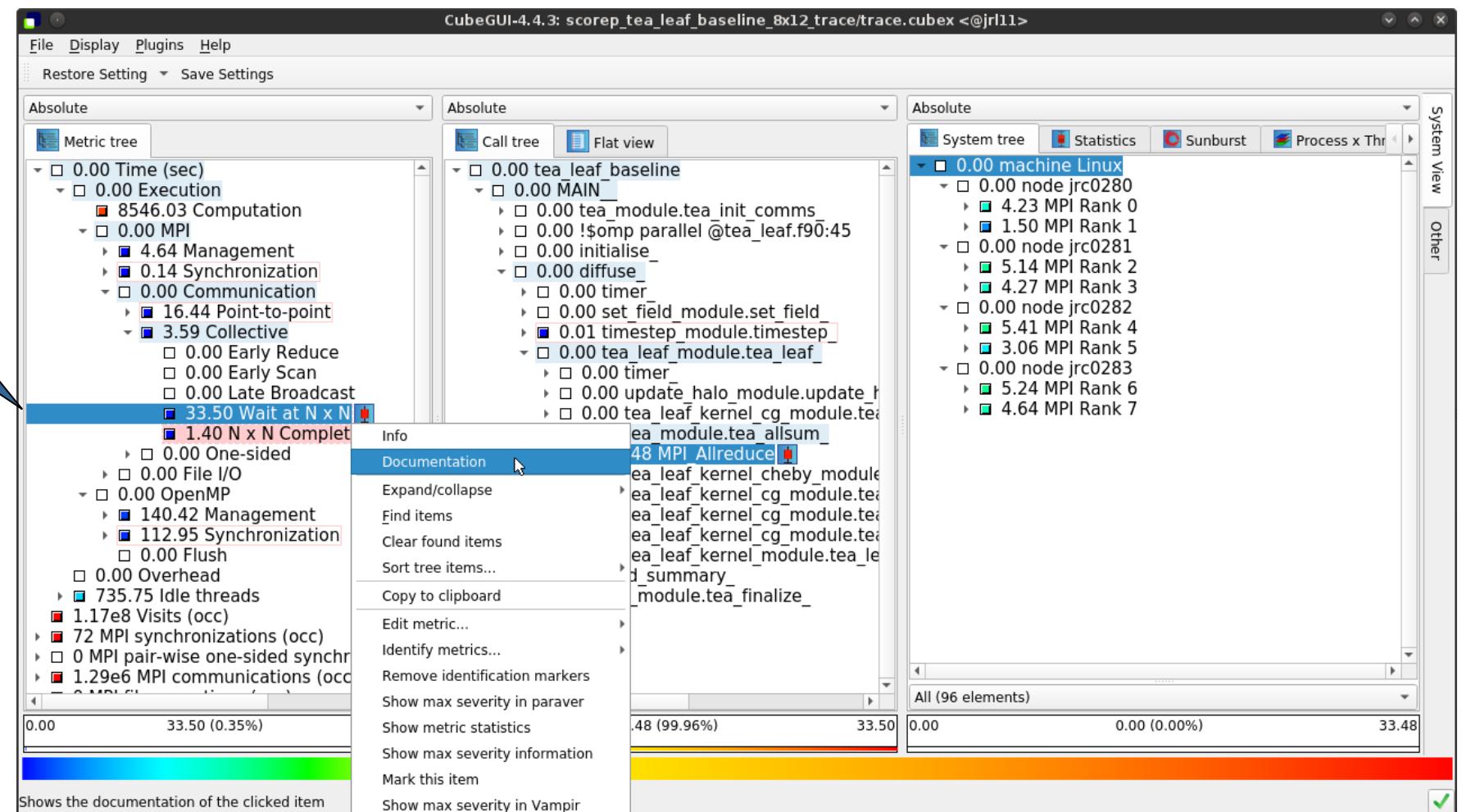
```
Usage: <launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>
```

- 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

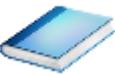
# Online metric description



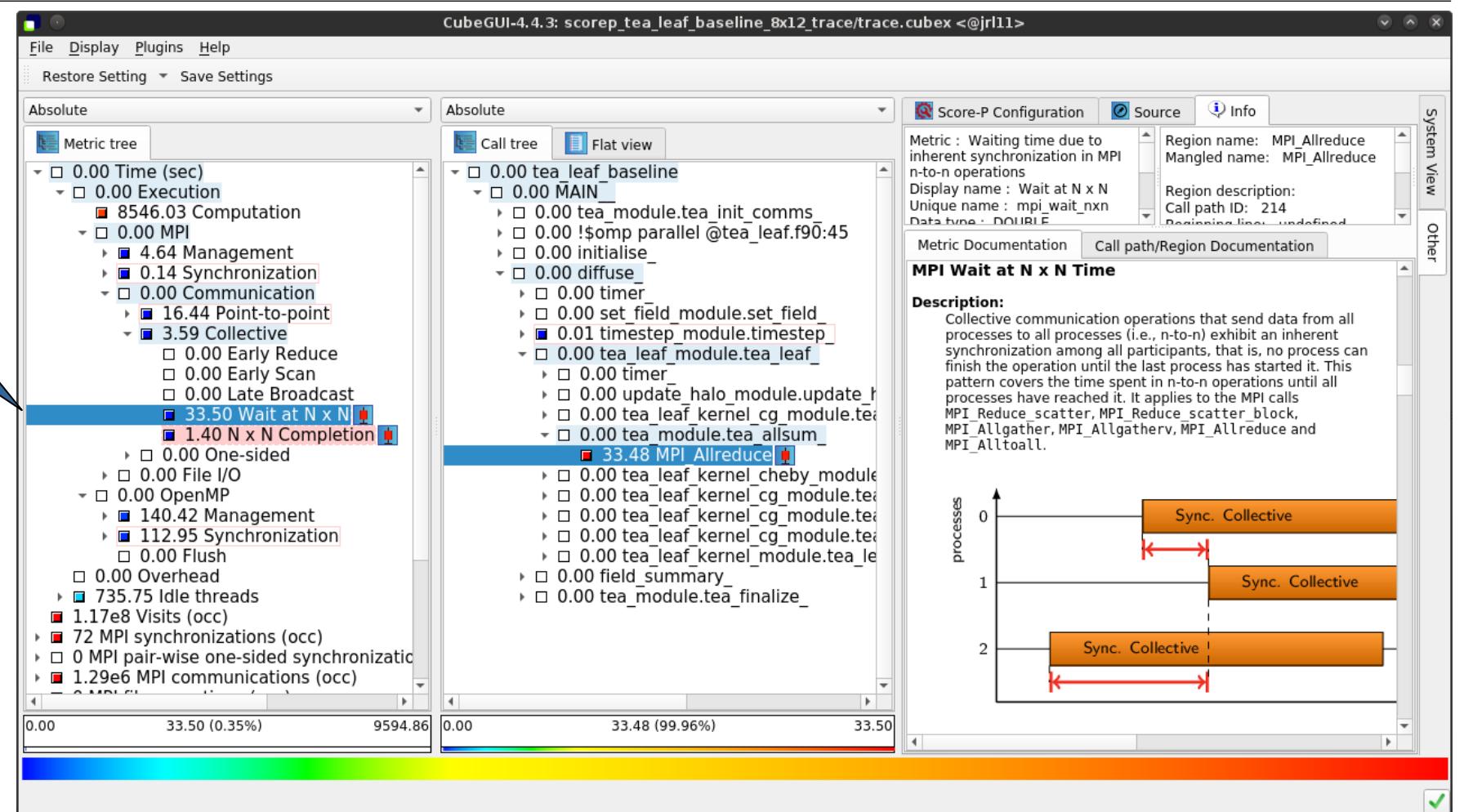
Access online metric description via context menu (right-click)



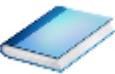
# Online metric description (cont.)



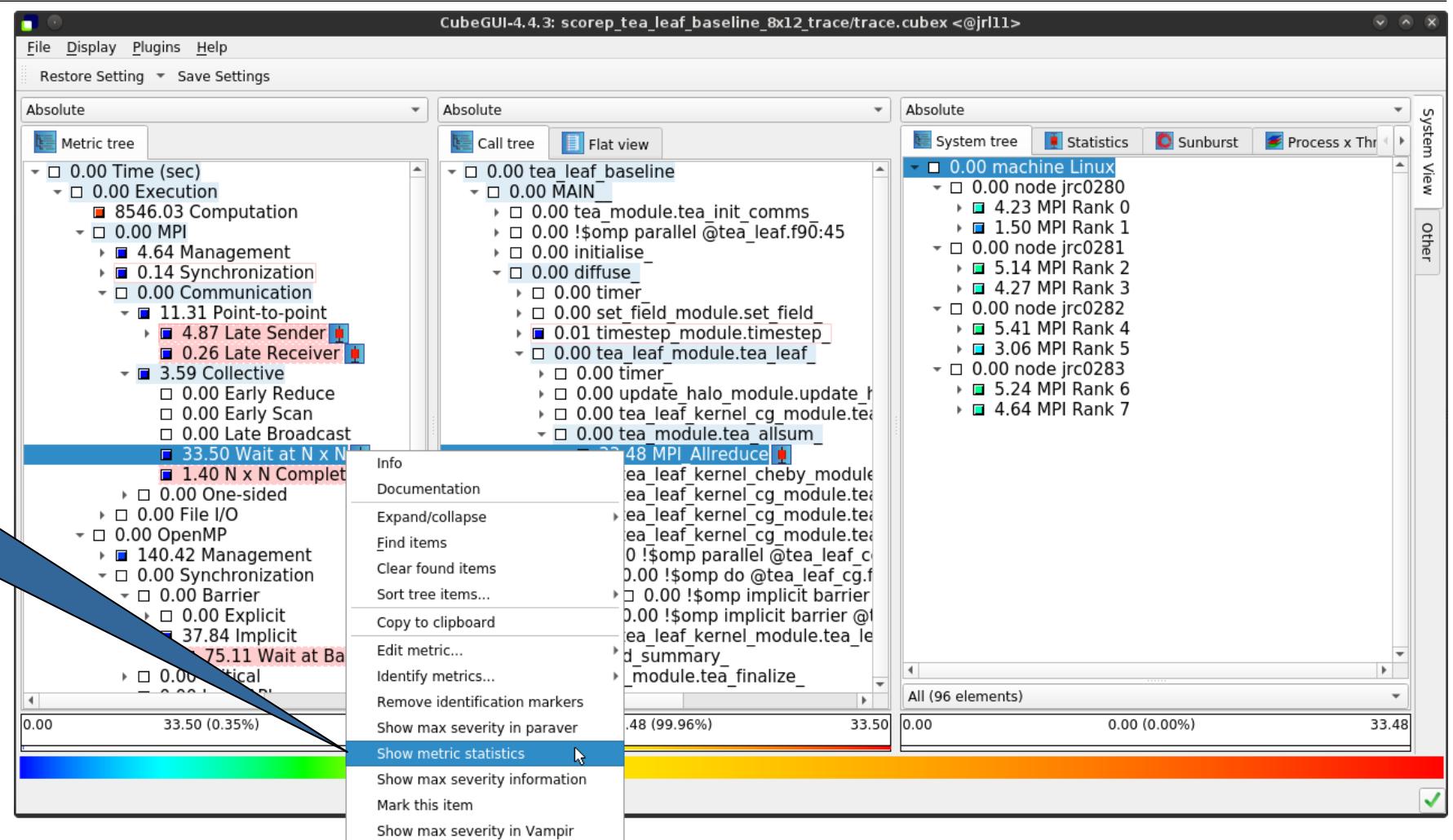
Selection of different metric automatically updates description

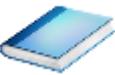


# Metric statistics



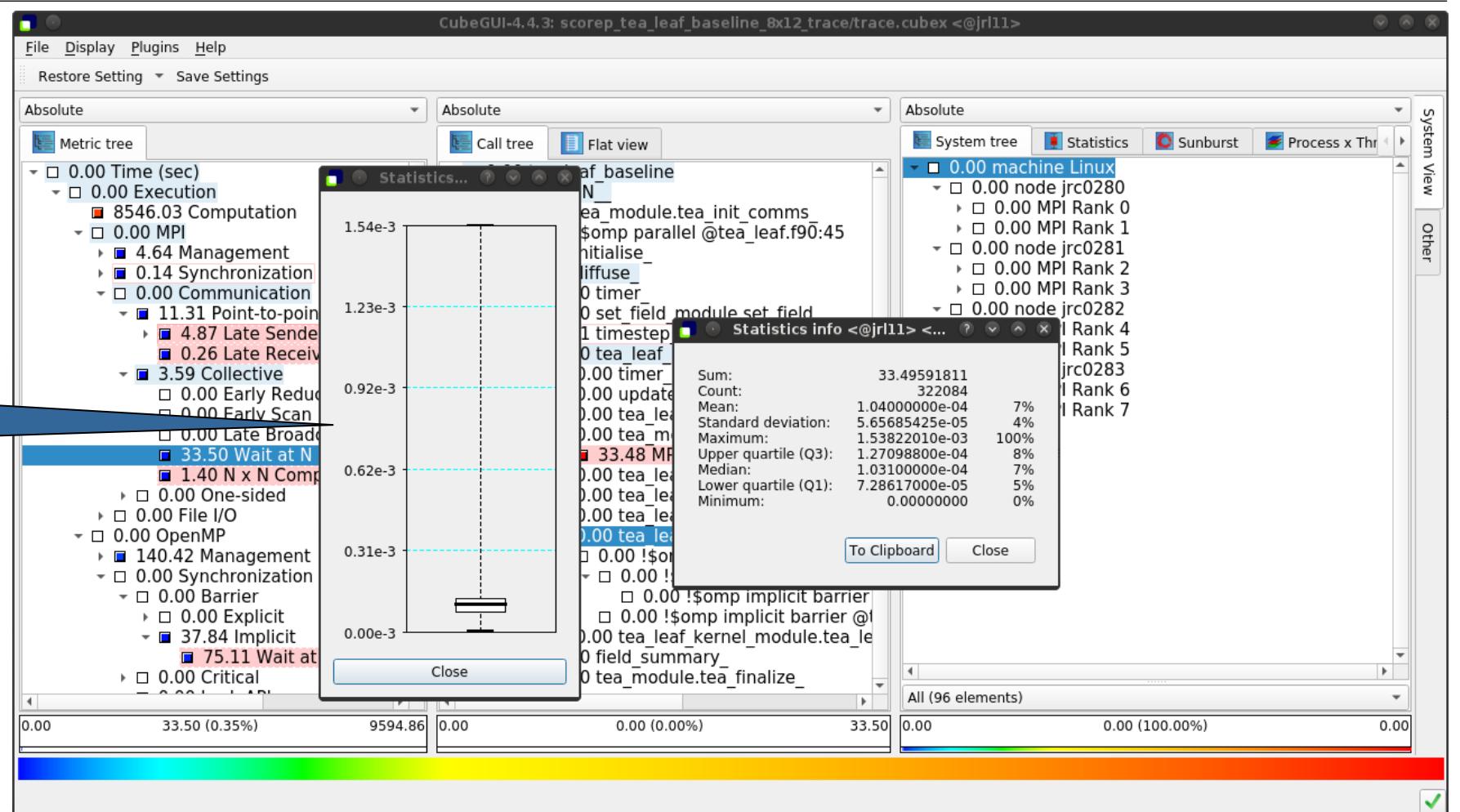
Access metric statistics for metrics marked with box plot icon from context menu

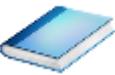




## Metric statistics (cont.)

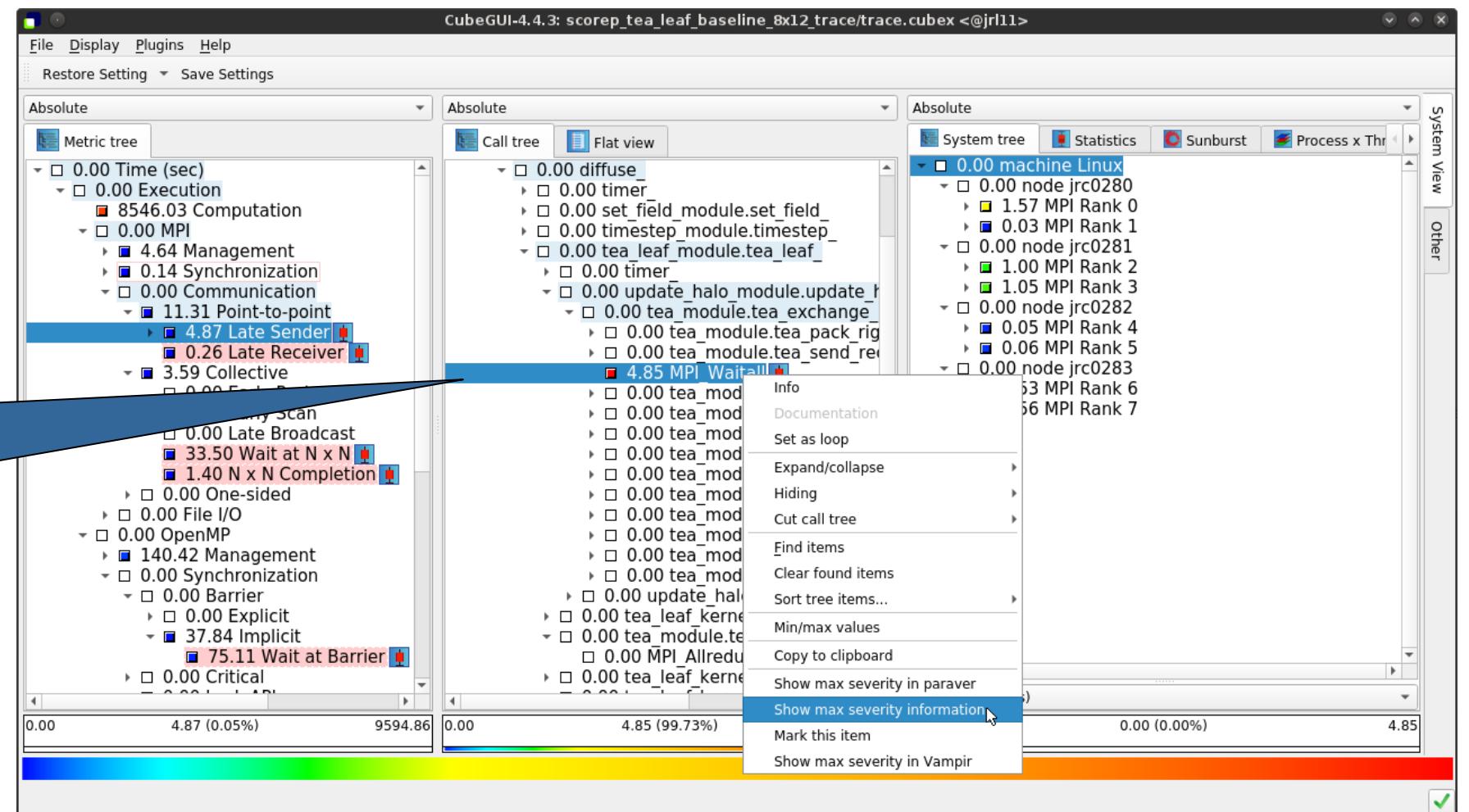
Shows instance statistics box plot, click to get details

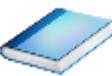




## Metric instance statistics

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





## Metric instance statistics (cont.)

## Shows instance details

