Automatic trace analysis with the Scalasca Trace Tools

Marc Schlütter
Jülich Supercomputing Centre

trace tools
scalasca

Scalasca Trace Tools

DOI 10.5281/zenodo.15125898

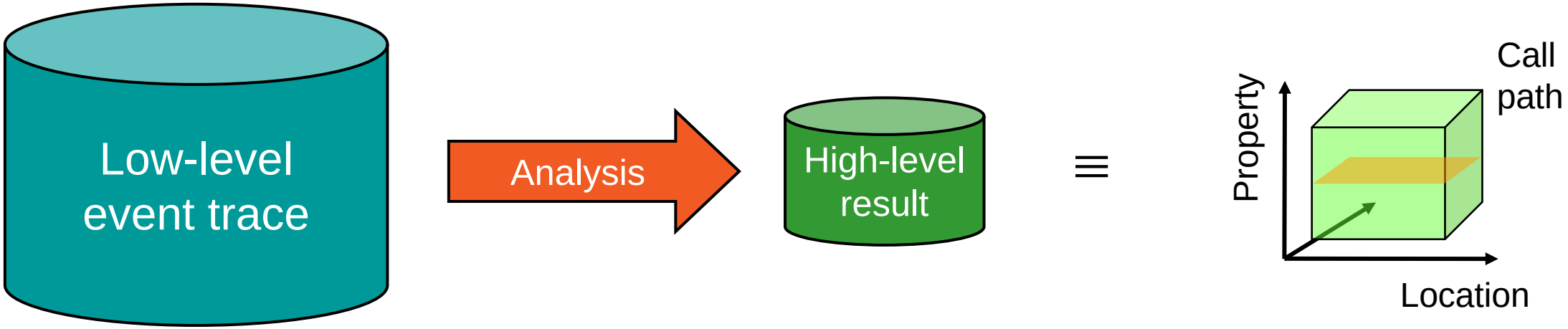
- **Scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
  - Analysis of traces including *only host-side events* from applications using CUDA, OpenCL, or OpenACC (also in combination 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

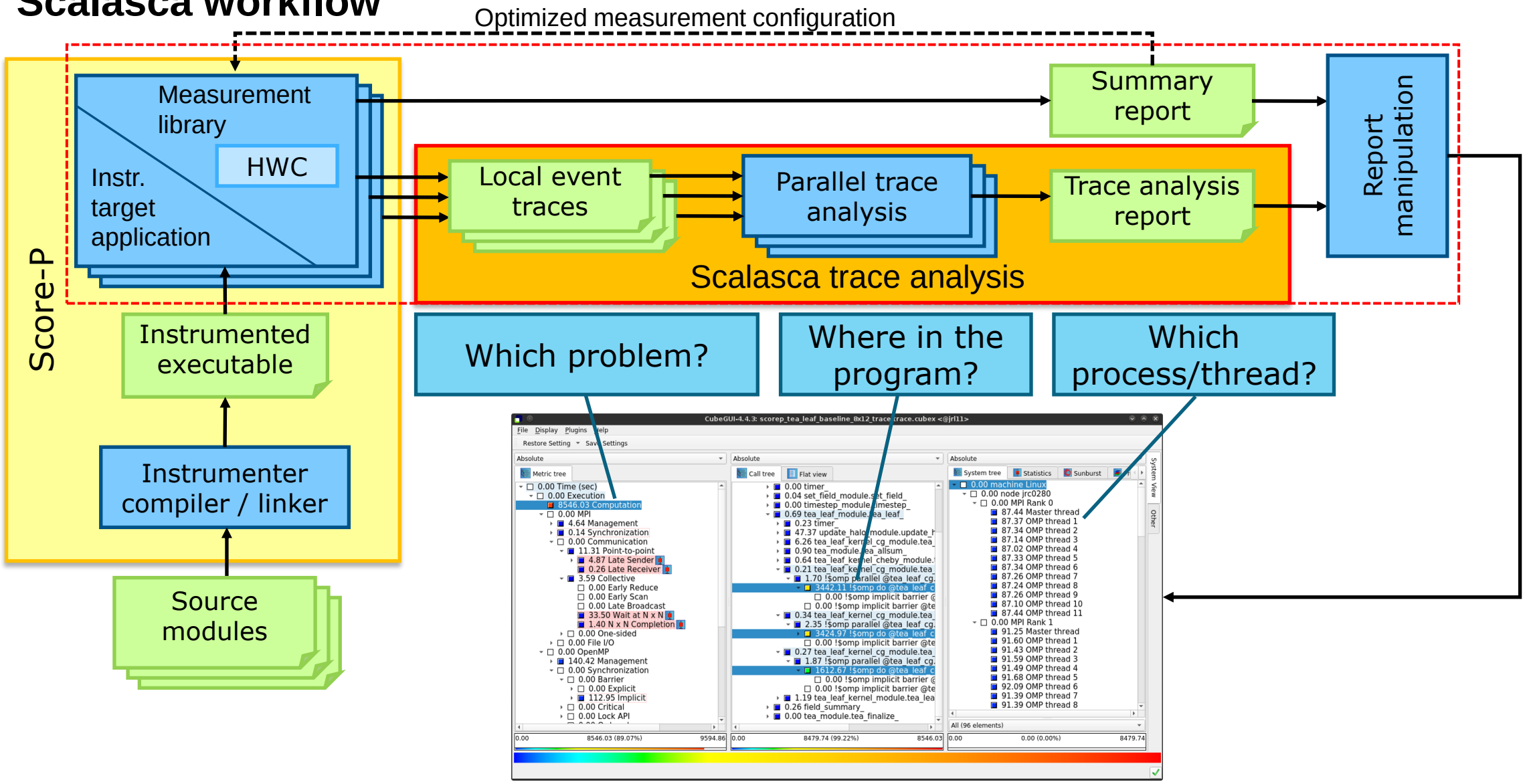
Scalasca Trace Tools: Features

- Open source, 3-clause BSD license
- Supports all major HPC platforms

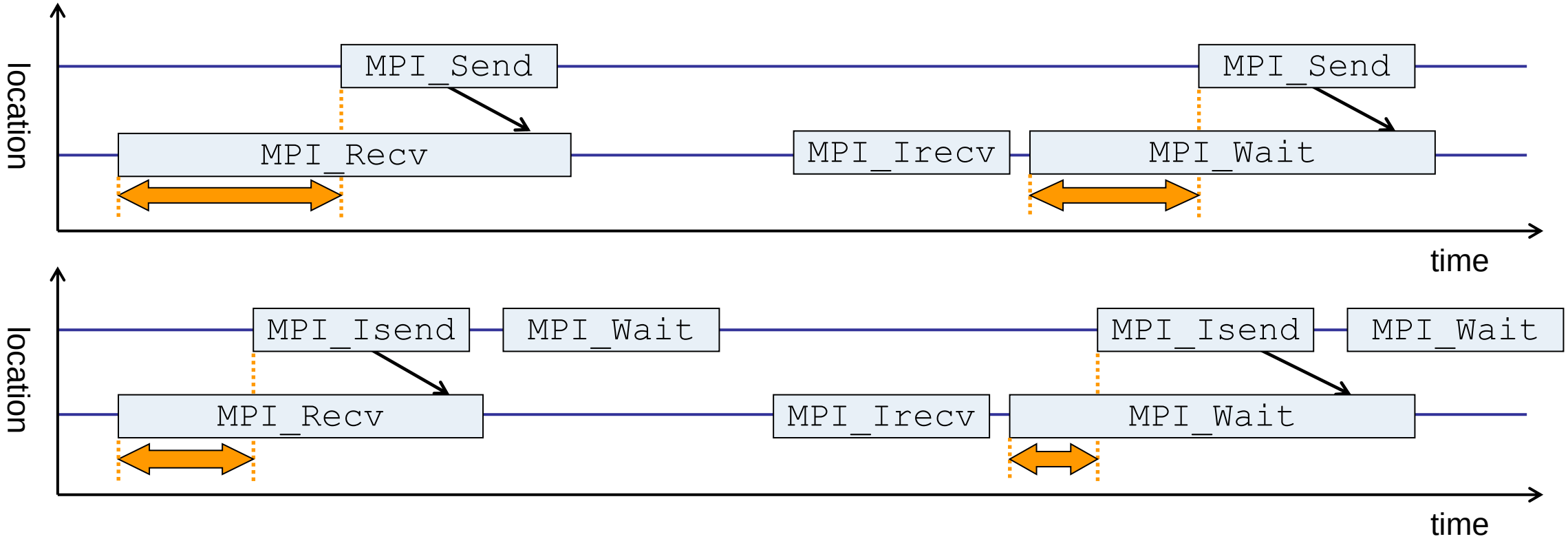
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Provides convenience commands for measurement, analysis, and post-processing
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 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

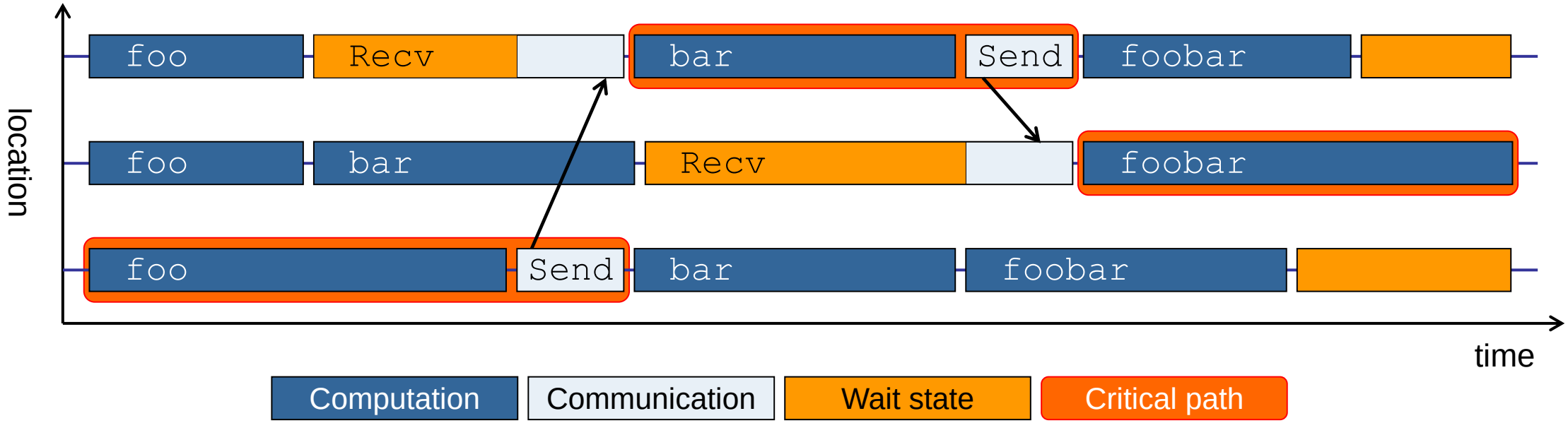


Example: "*Late Sender*" wait state



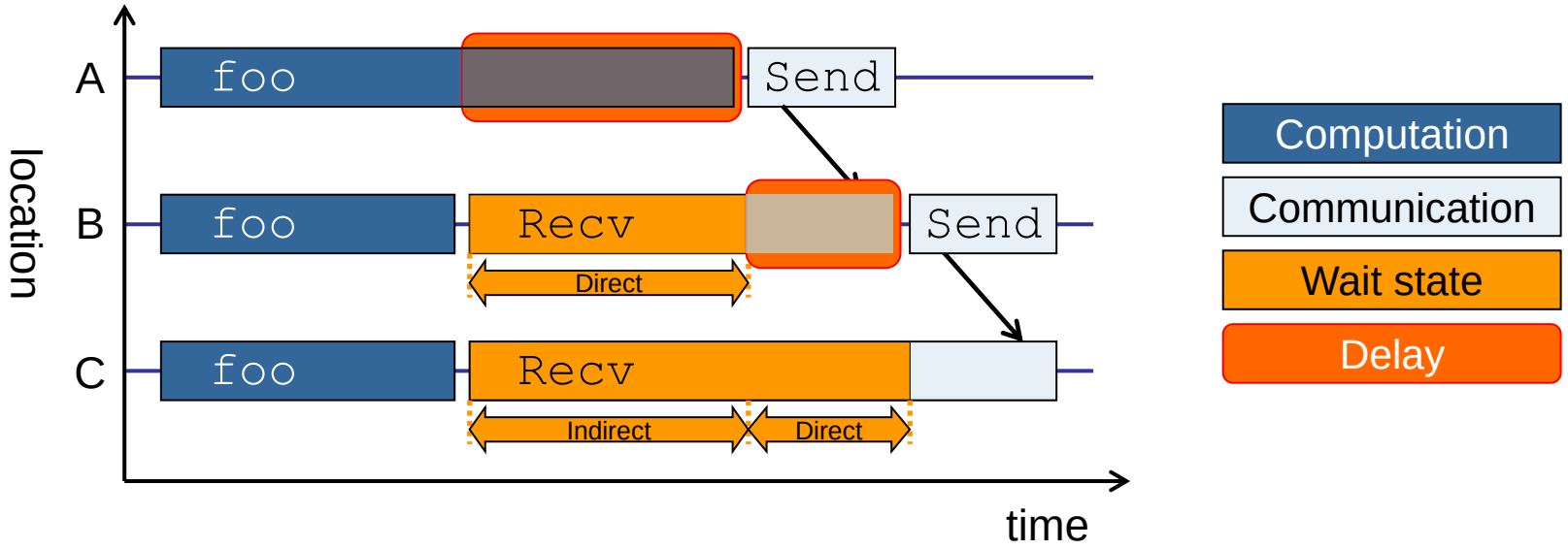
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

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*

Demo:
NPB-MZ-MPI / BT

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

Prefix scheduler cmd with scan:
`scan` = `scalasca -analyze`
`-s` = profile/summary (default)

- 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 … 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
```

Prefix scheduler cmd with scan:
scan = scalasca -analyze
-t = trace collection and automatic trace analysis

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

Multi-run measurements with *scan*

- Multiple runs through configurations, repetitions or a combination of both
  - Aggregation into singular Cube result via *square*

- Configurations:
  - Set of environment variables for a run; User and Score-P/Scalasca variables
  - For consistency, used Scalasca and Score-P variables have to be stated in the config file
  - **Use case**: different application/measurement settings, e.g., varying hardware counters, input sets
  - **Note**: Runs with too much variance in application behaviour might prevent combination

- Runs per configuration:
  - Number of runs with the same configuration (default environment or a specific run configuration)
  - **Use case**: statistical stability/analysis

Multi-run measurements: POP Preset

```
% scan
Scalasca 2.6.2: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
       where {options} may include:
...
  -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.
```

```
% square
Scalasca 2.6.2: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
...
  -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
```

Multi-run measurements: Config file format

Global Section '--'
First one, max. one

Run Section '-'
Everything after '-' is comment

KEY=VALUE Pair
Everything after first = is taken as value

Comments '#' and blank lines are ignored

Run Settings supersede Global settings

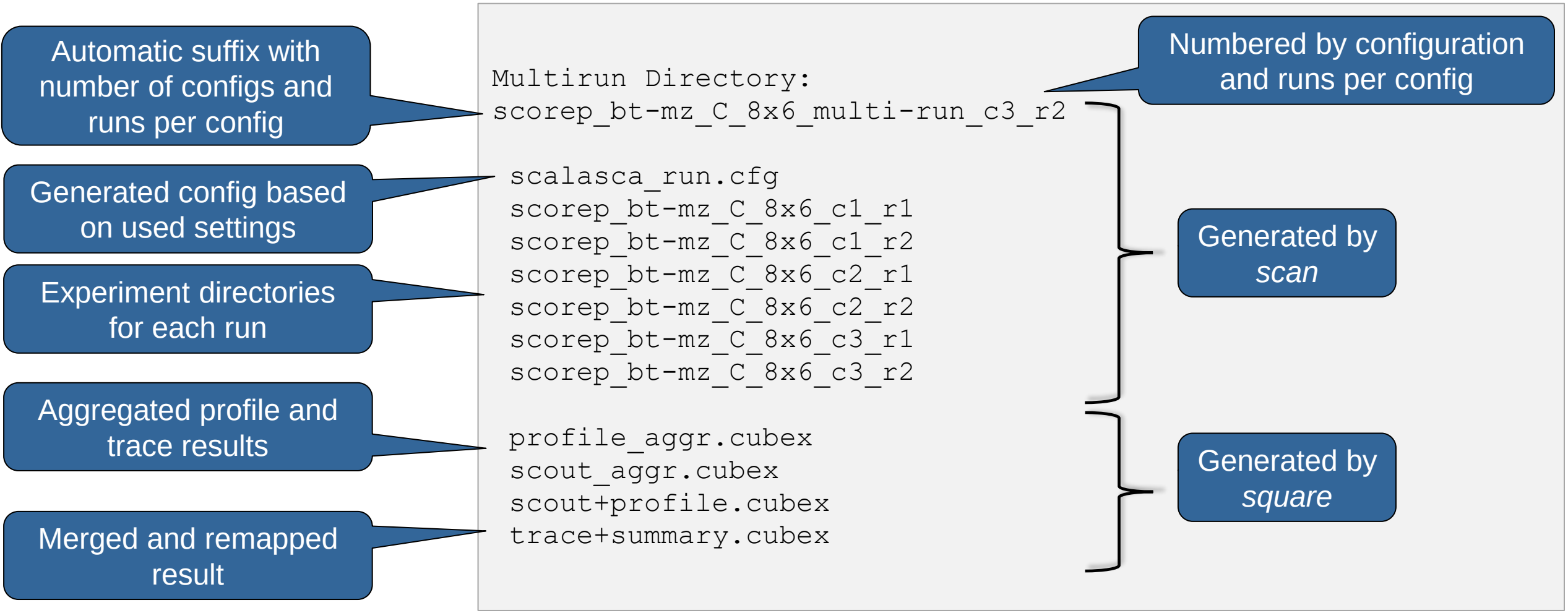
```
-- Global variable section marked by '--', comment optional
SCOREP_ENABLE_PROFILING=true
SCOREP_ENABLE_TRACING=false
# for all runs, not doing so will result in un-mergable results
SCOREP_FILTERING_FILE=../config/scorep.filt

# Begin of the per run settings each marked with a '-'
- Profiling run with PAPI counters
SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_RES_STL

- Profiling without counters to get a low overhead measurement
# no additional Score-P variables needed

- Scalasca tracing run
SCOREP_TOTAL_MEMORY=80M
SCOREP_ENABLE_PROFILING=false
SCOREP_ENABLE_TRACING=true
SCAN_ANALYZE_OPTS=--time-correct
```

Multi-run measurements: Results

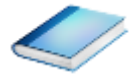


Case study:
TeaLeaf MPI+OpenMP

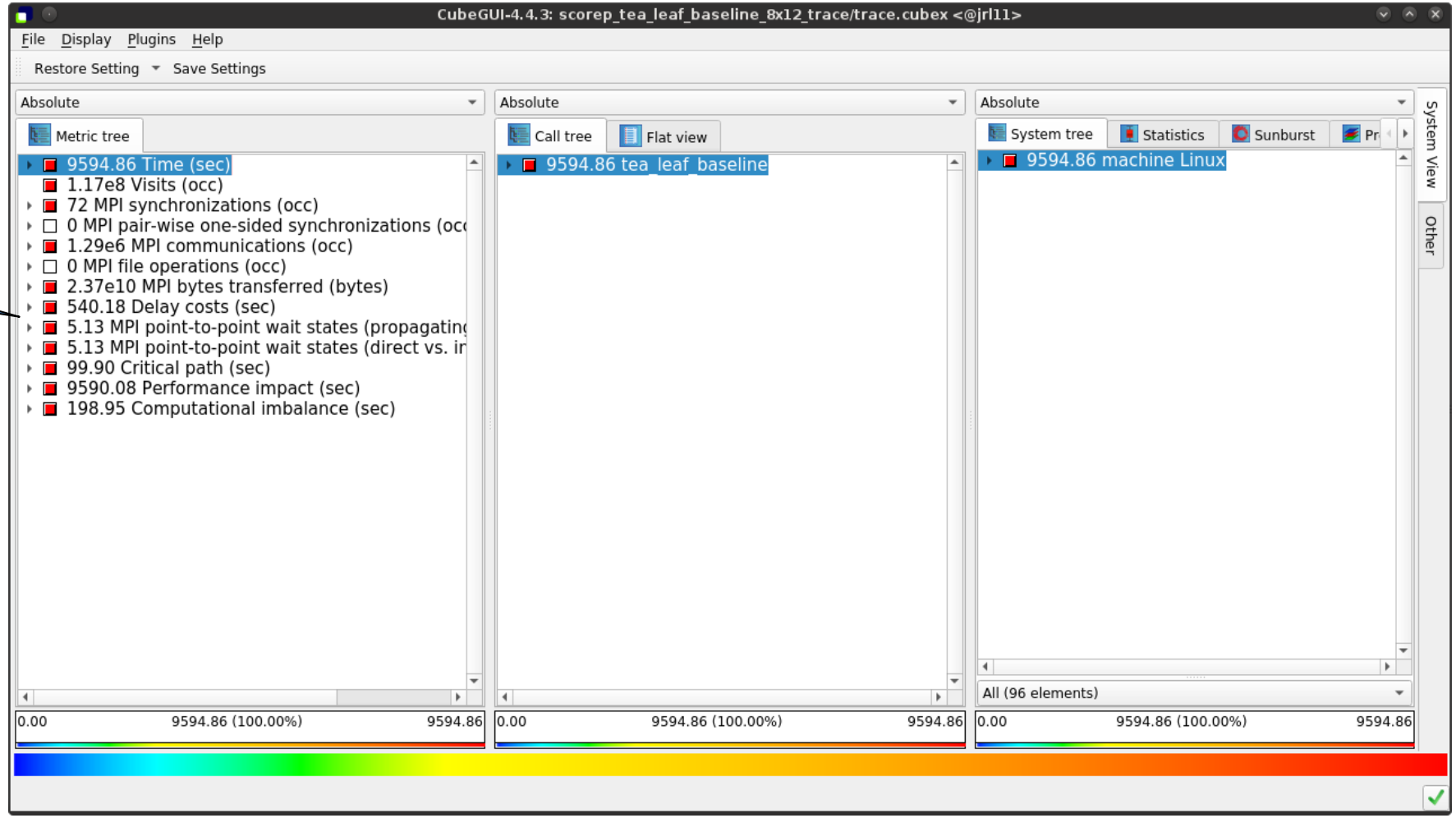
Case study: TeaLeaf MPI+OpenMP

- 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 × 4000 cells, CG solver

Scalasca analysis report exploration (opening view)

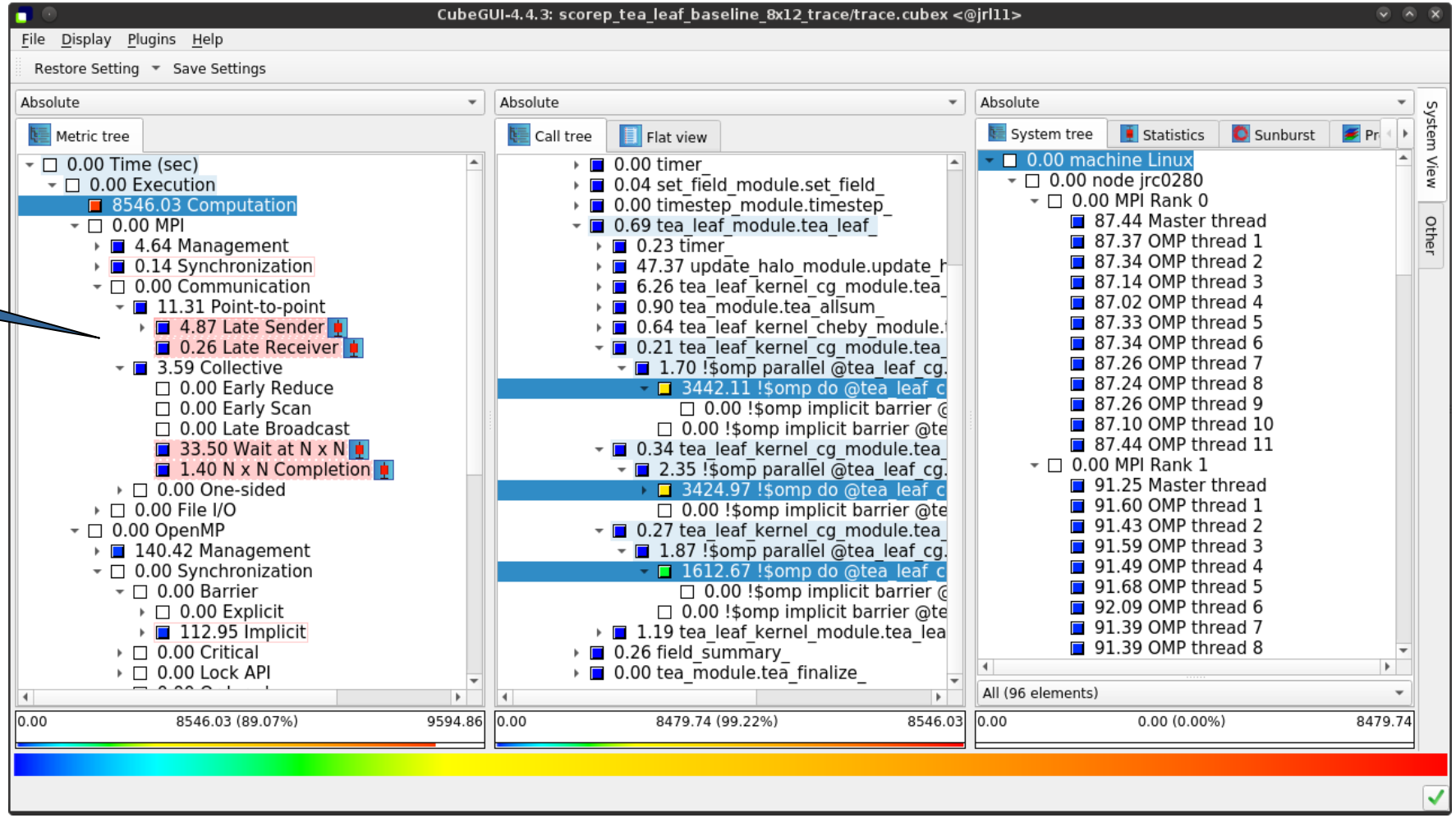


Additional top-level metrics produced by the trace analysis…



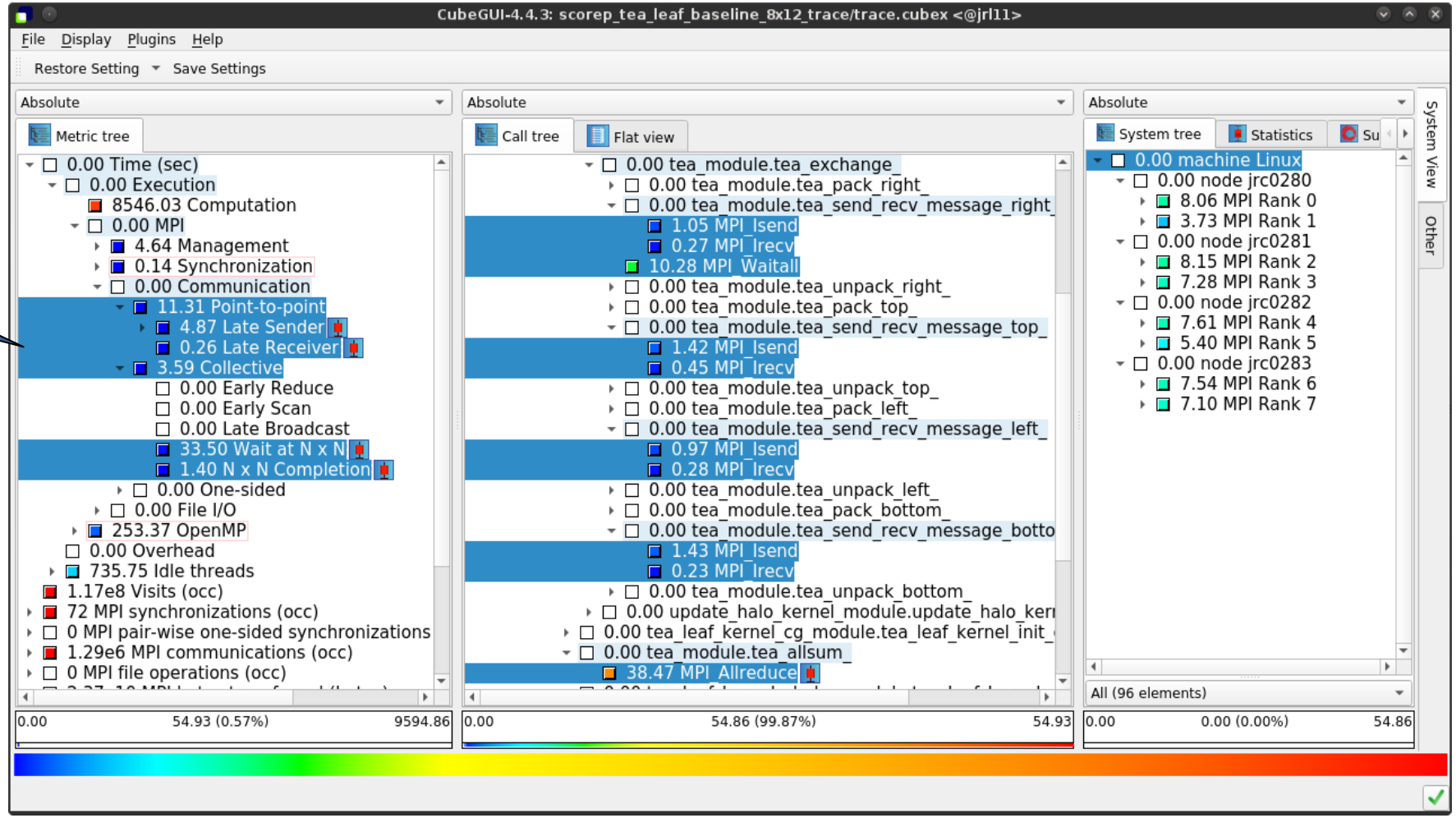
Scalasca wait-state metrics

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

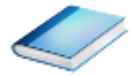


TeaLeaf Scalasca report analysis (I)

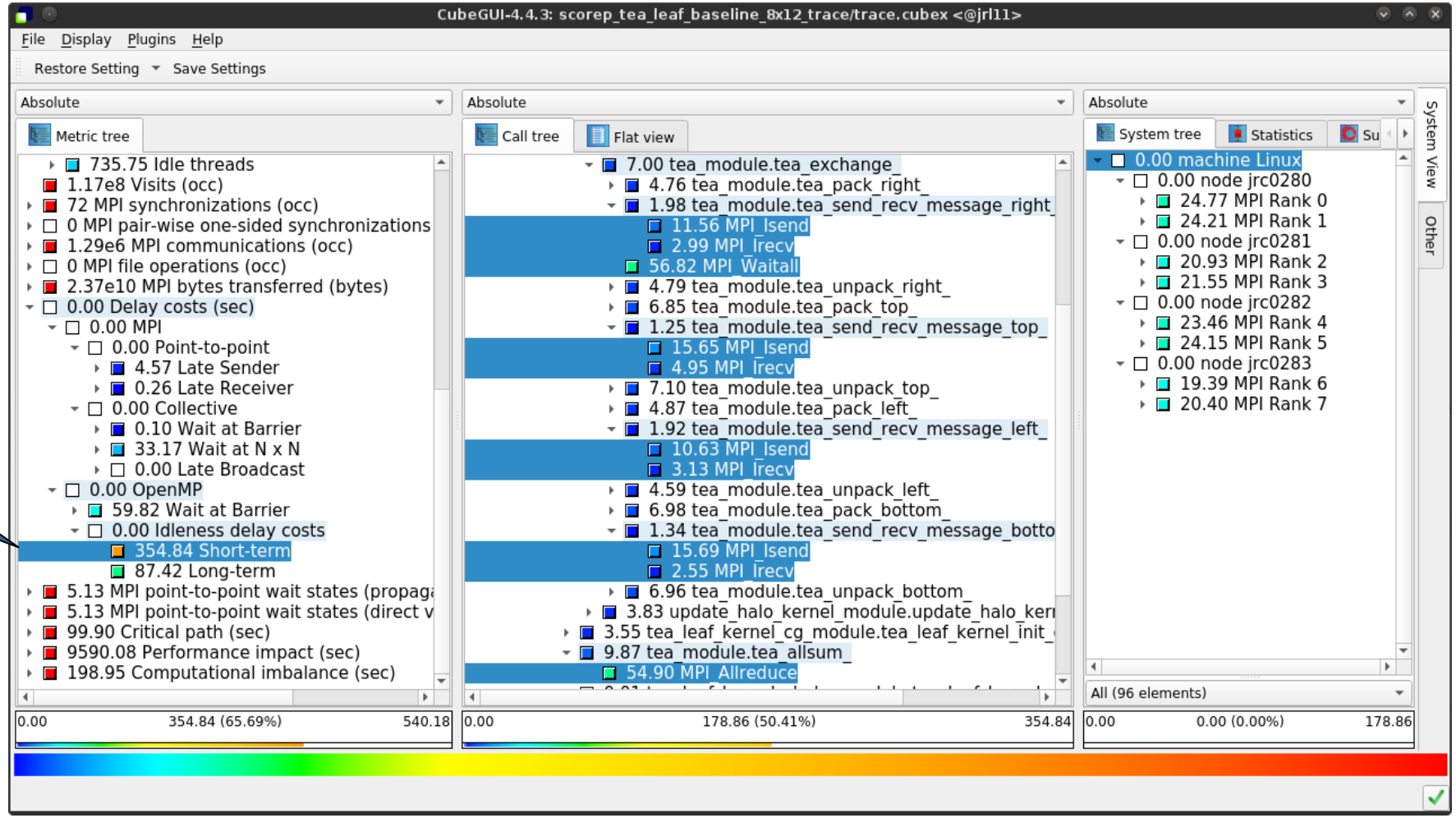
While MPI communication time and wait states are small (~0.6% of the total execution time)…



TeaLeaf Scalasca report analysis (II)

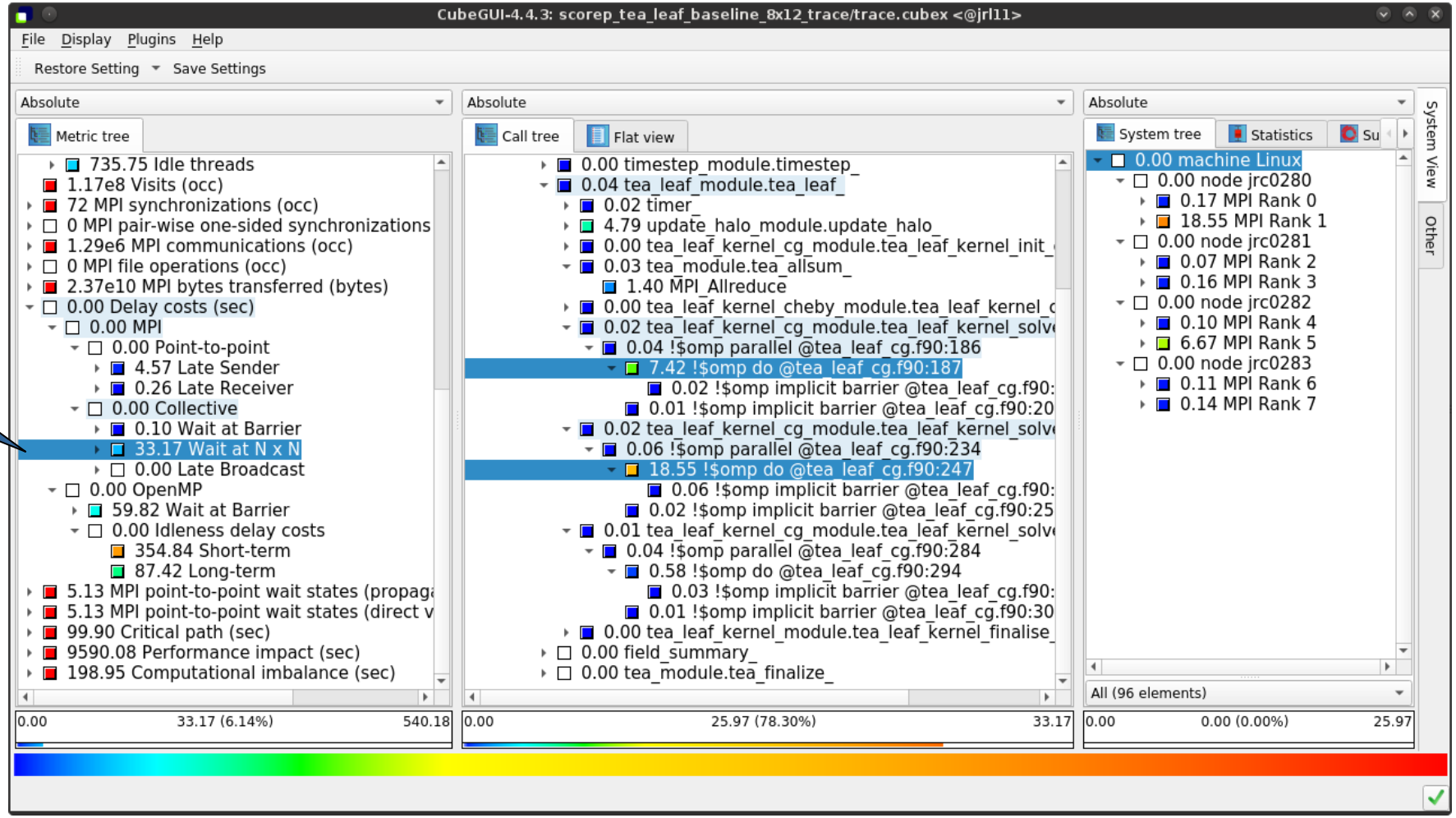


…they directly cause a significant amount of the OpenMP thread idleness

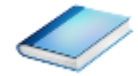


TeaLeaf Scalasca report analysis (III)

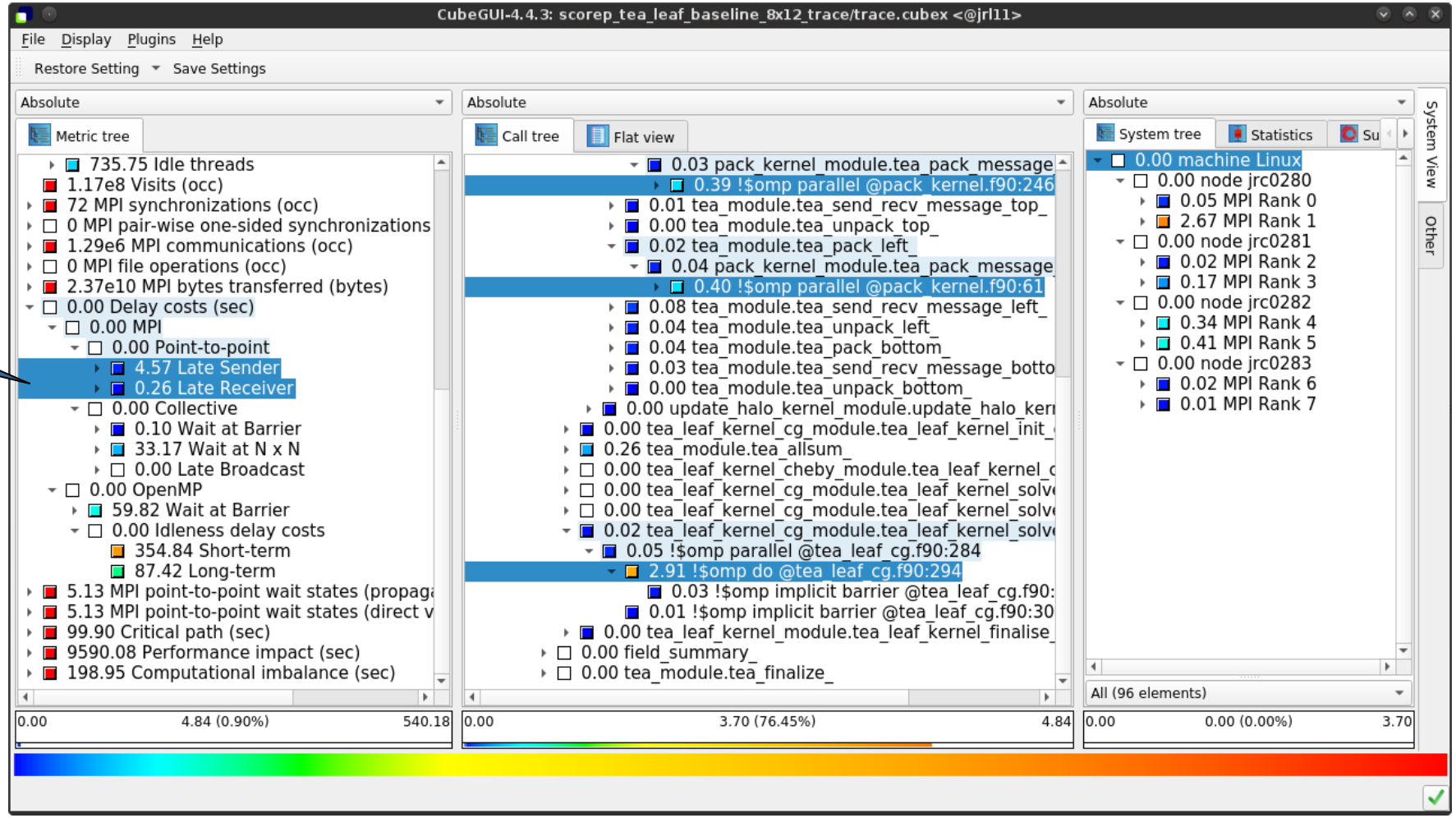
The “Wait at NxN” collective wait states are mostly caused by the first 2 OpenMP `do` loops of the solver (on ranks 5 & 1, resp.)…



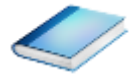
TeaLeaf Scalasca report analysis (IV)



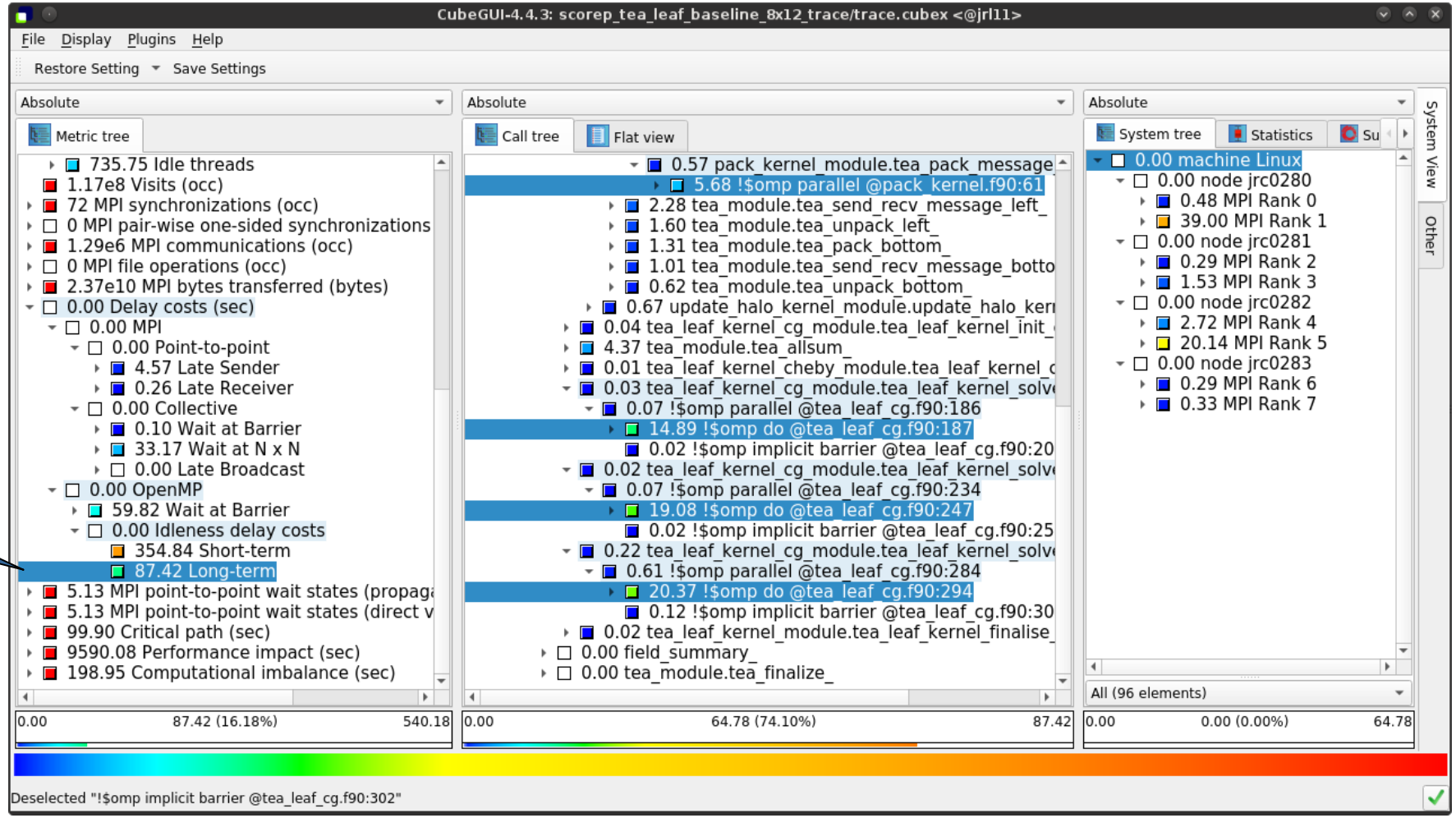
…while the MPI point-to-point wait states are caused by the 3rd solver `do` loop (on rank 1) and two loops in the halo exchange



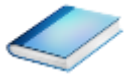
TeaLeaf Scalasca report analysis (V)



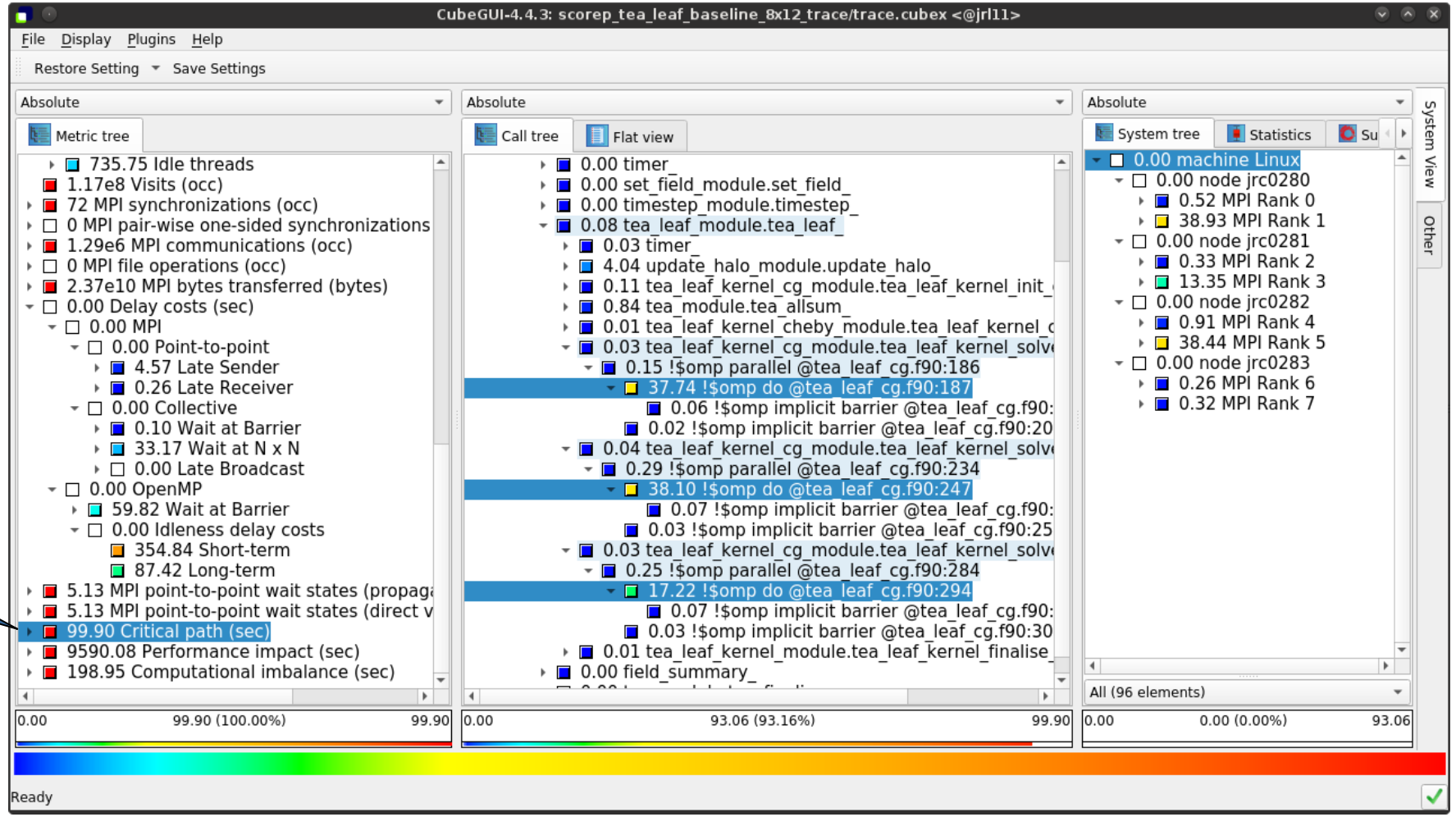
Various OpenMP `do` loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation



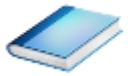
TeaLeaf Scalasca report analysis (VI)



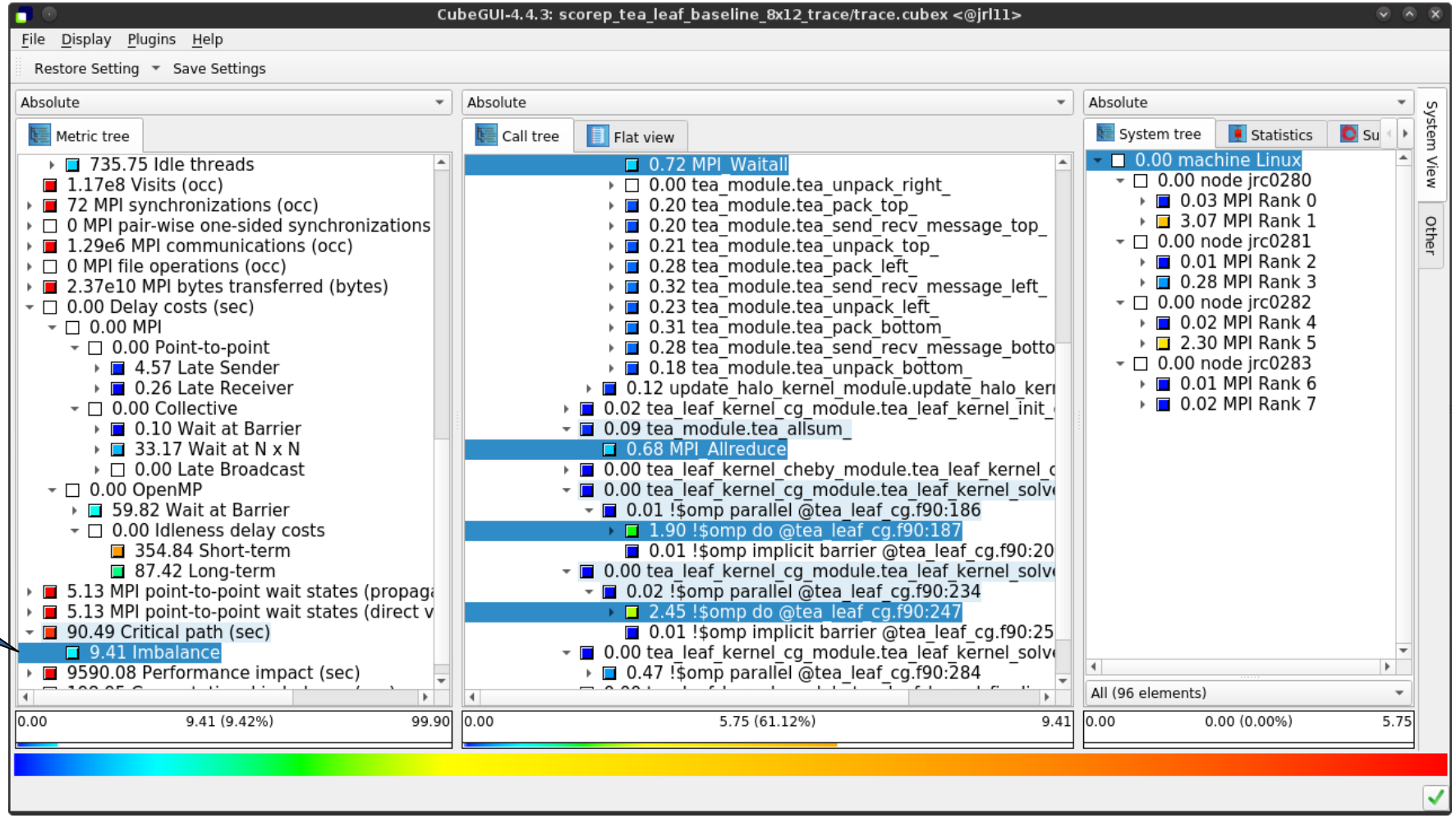
The Critical Path also highlights the three solver loops…



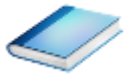
TeaLeaf Scalasca report analysis (VII)



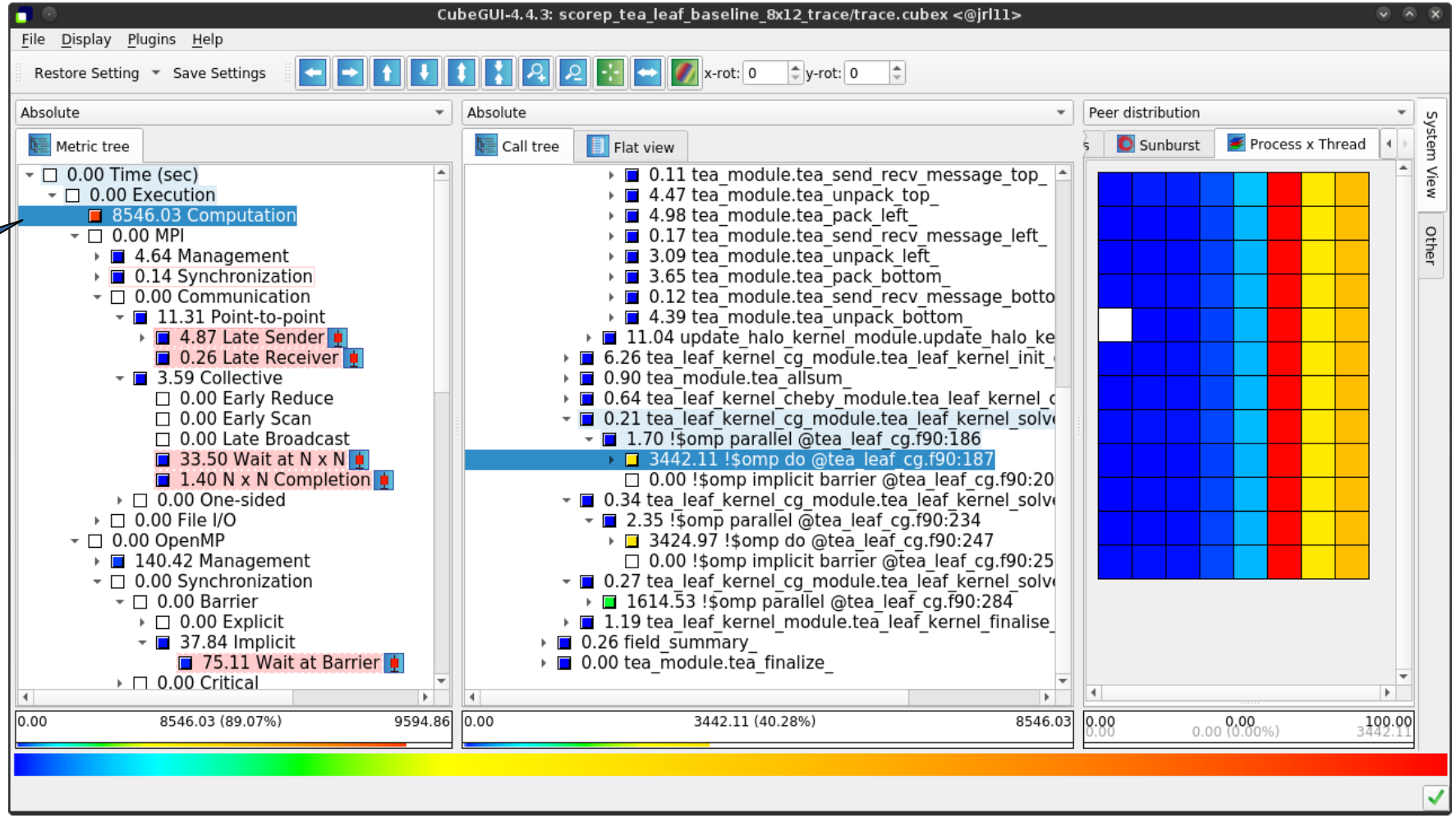
…with imbalance (time on critical path above average) mostly in the first two loops and MPI communication



TeaLeaf Scalasca report analysis (VIII)

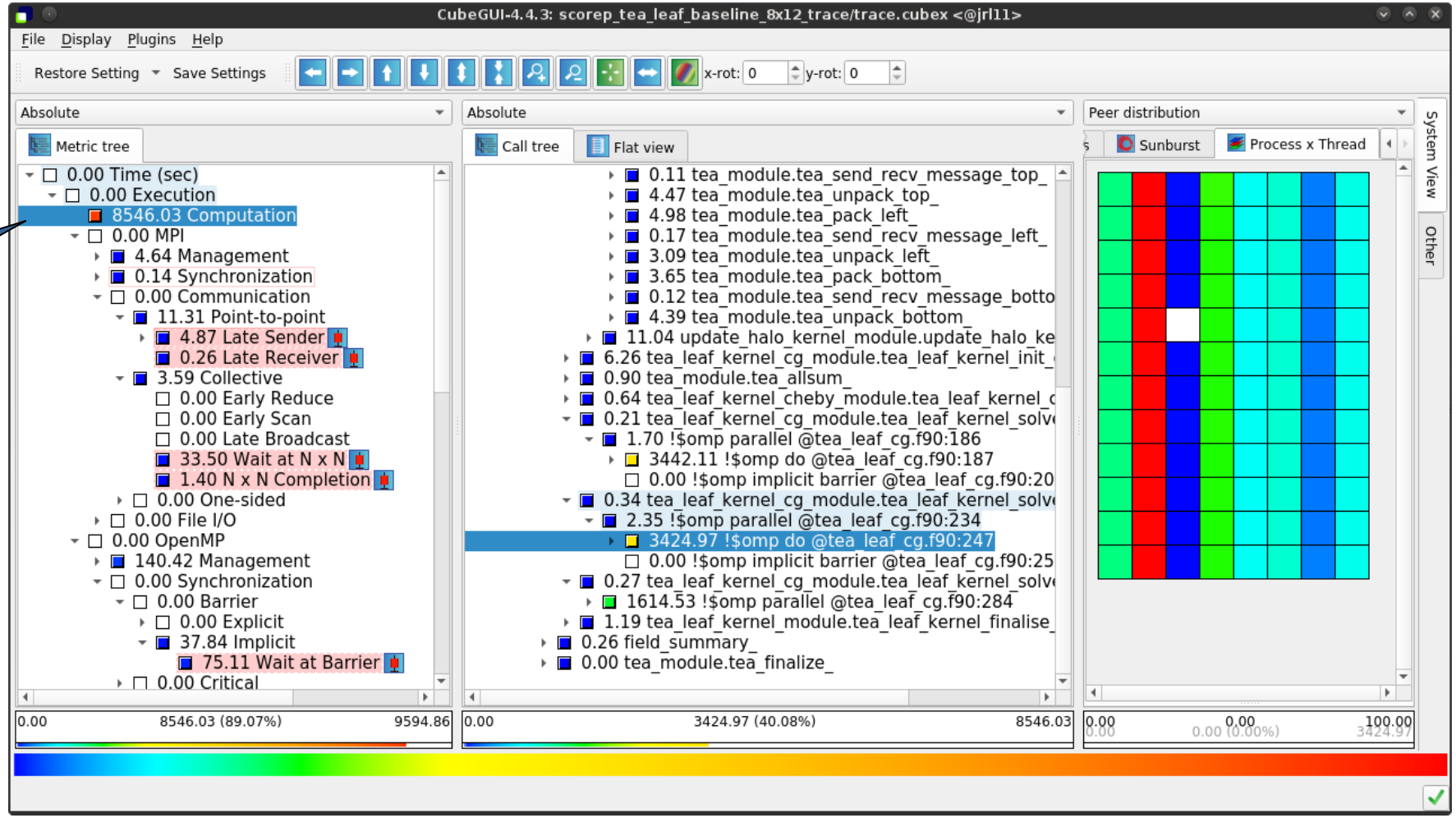


Computation time of 1st…



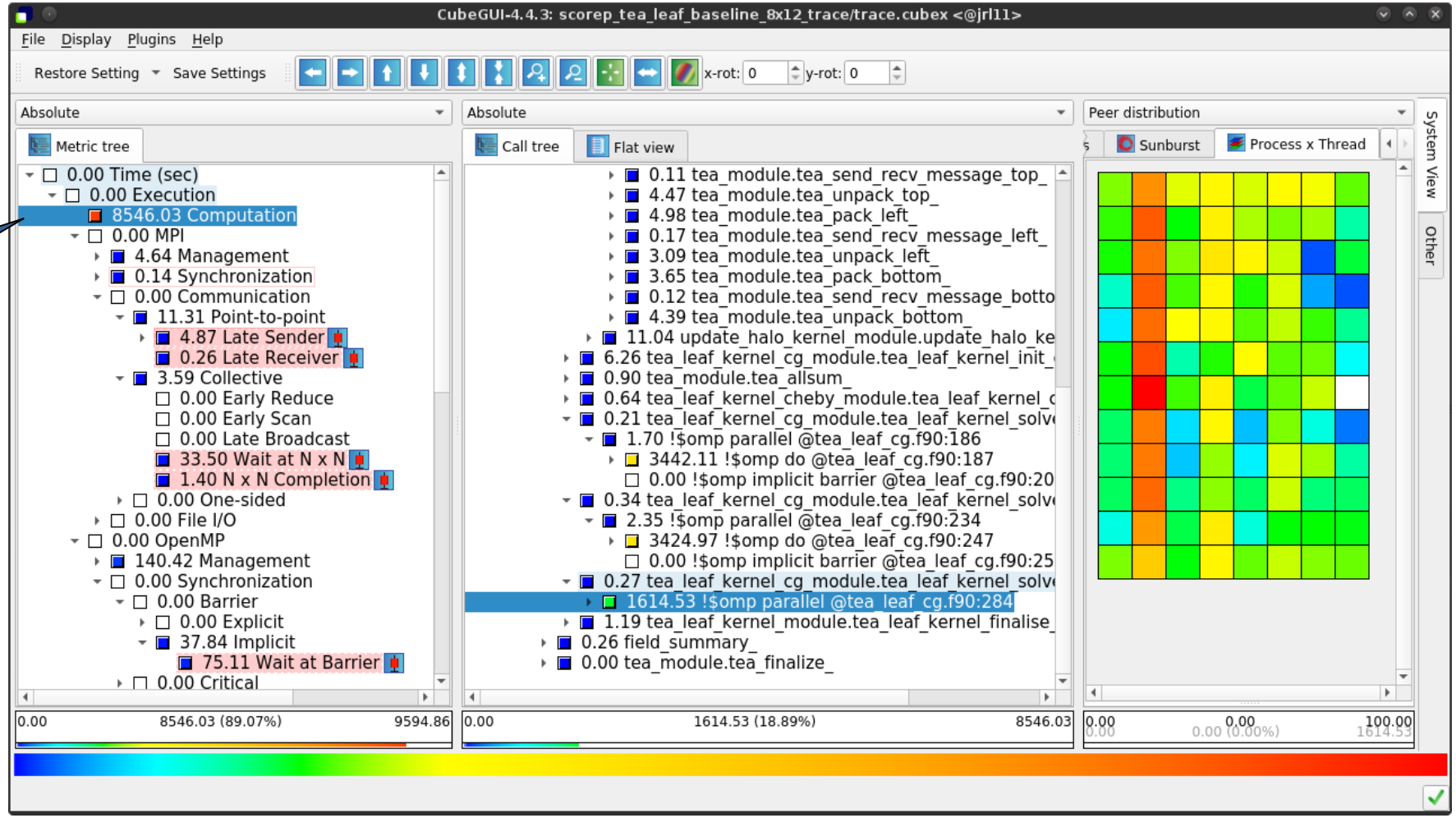
TeaLeaf Scalasca report analysis (IX)

…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%**

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

trace tools scalasca

Reference material

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 remove 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.
```

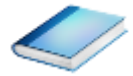
- Scalasca measurement collection & analysis nexus

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

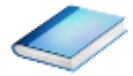


```
% scout.hyb --help
SCOUT   (Scalasca 2.6.2)
Copyright (c) 1998-2022 Forschungszentrum Juelich GmbH
Copyright (c) 2014-2021 RWTH Aachen University
Copyright (c) 2009-2014 German Research School for Simulation Sciences GmbH

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

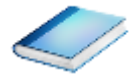


- Scalasca trace event timestamp consistency correction

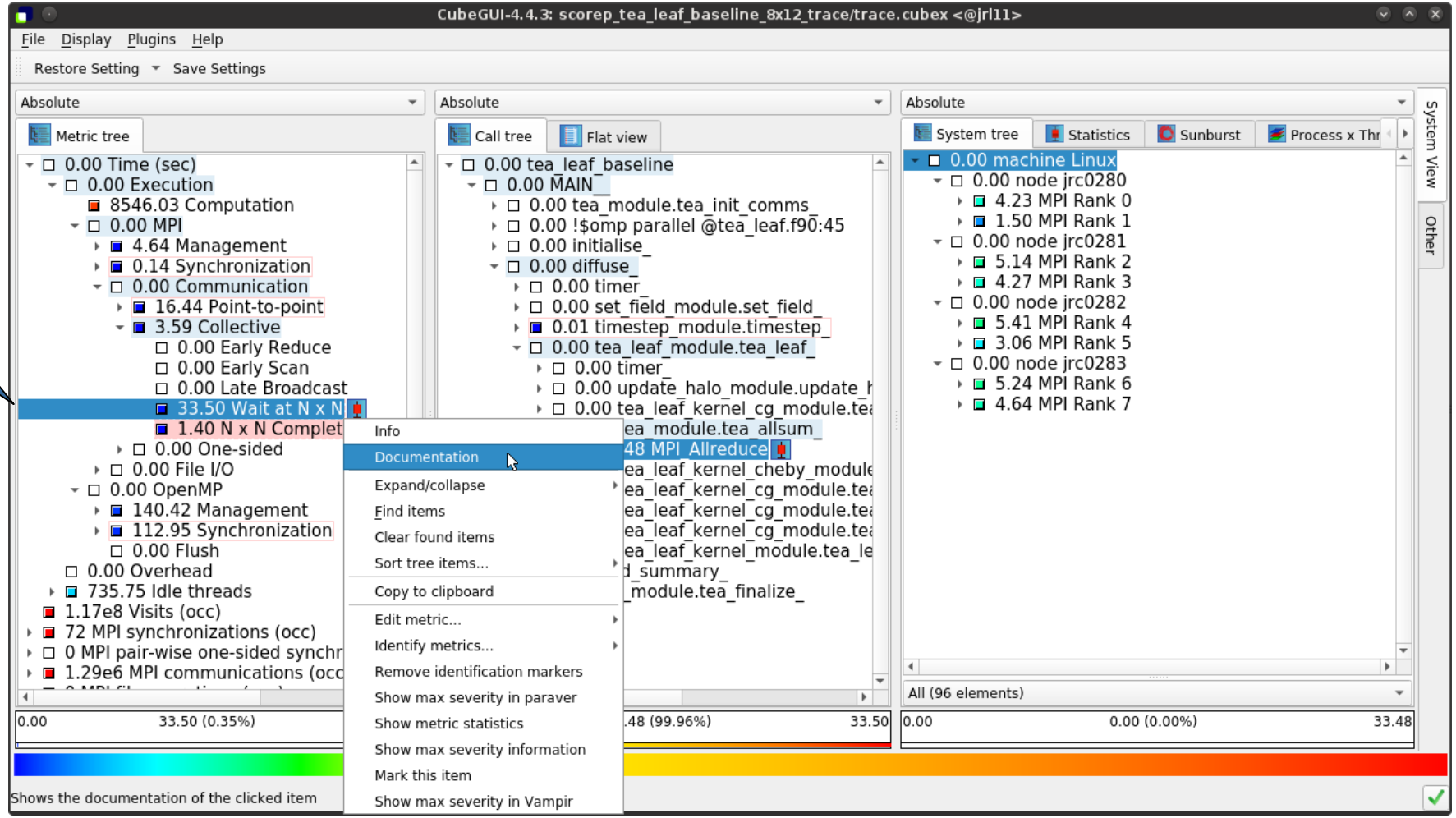
```
Usage: <launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>
```

  - 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

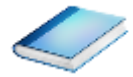
Online metric description



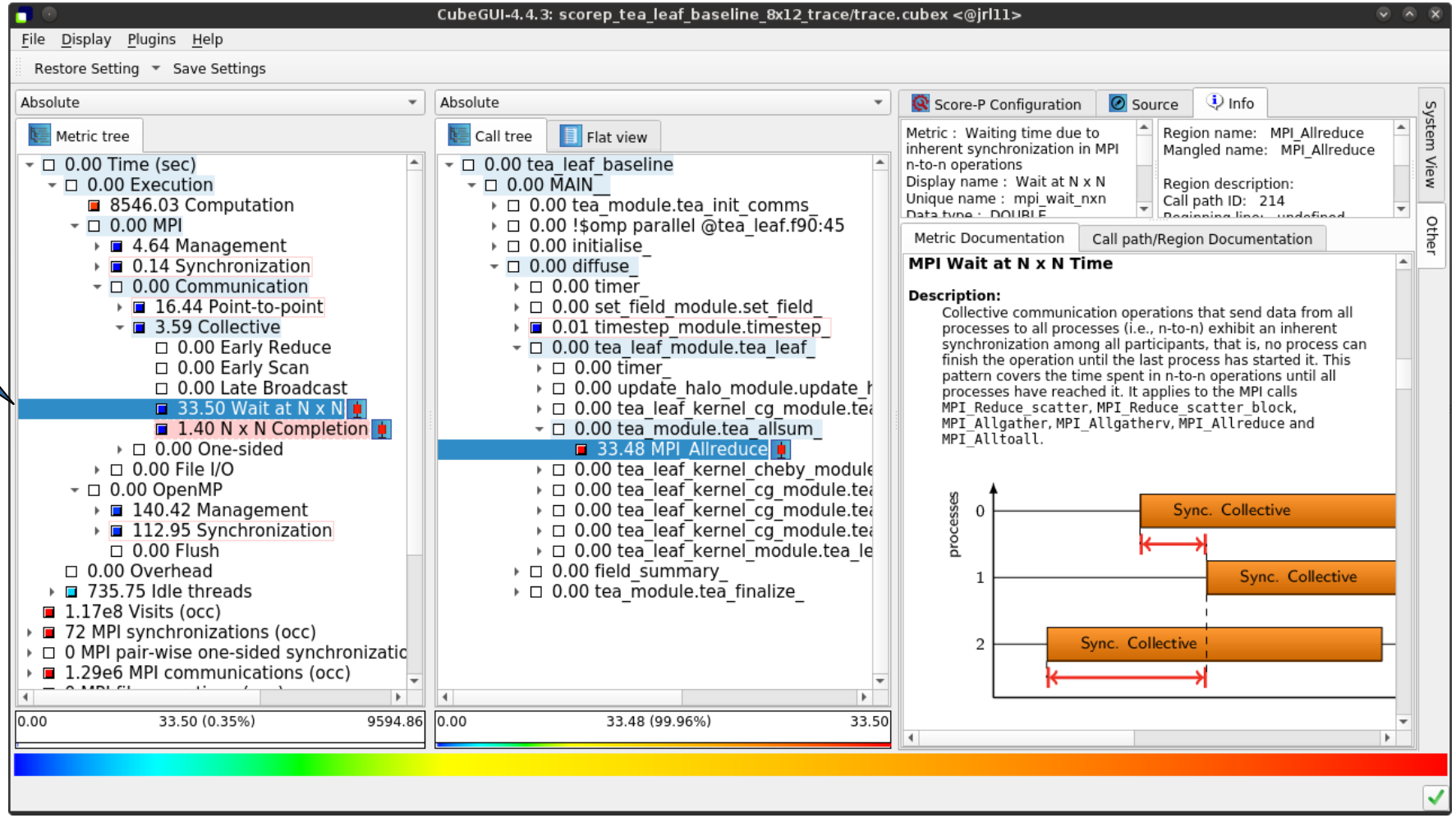
Access online metric description via context menu (right-click)



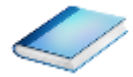
Online metric description (cont.)



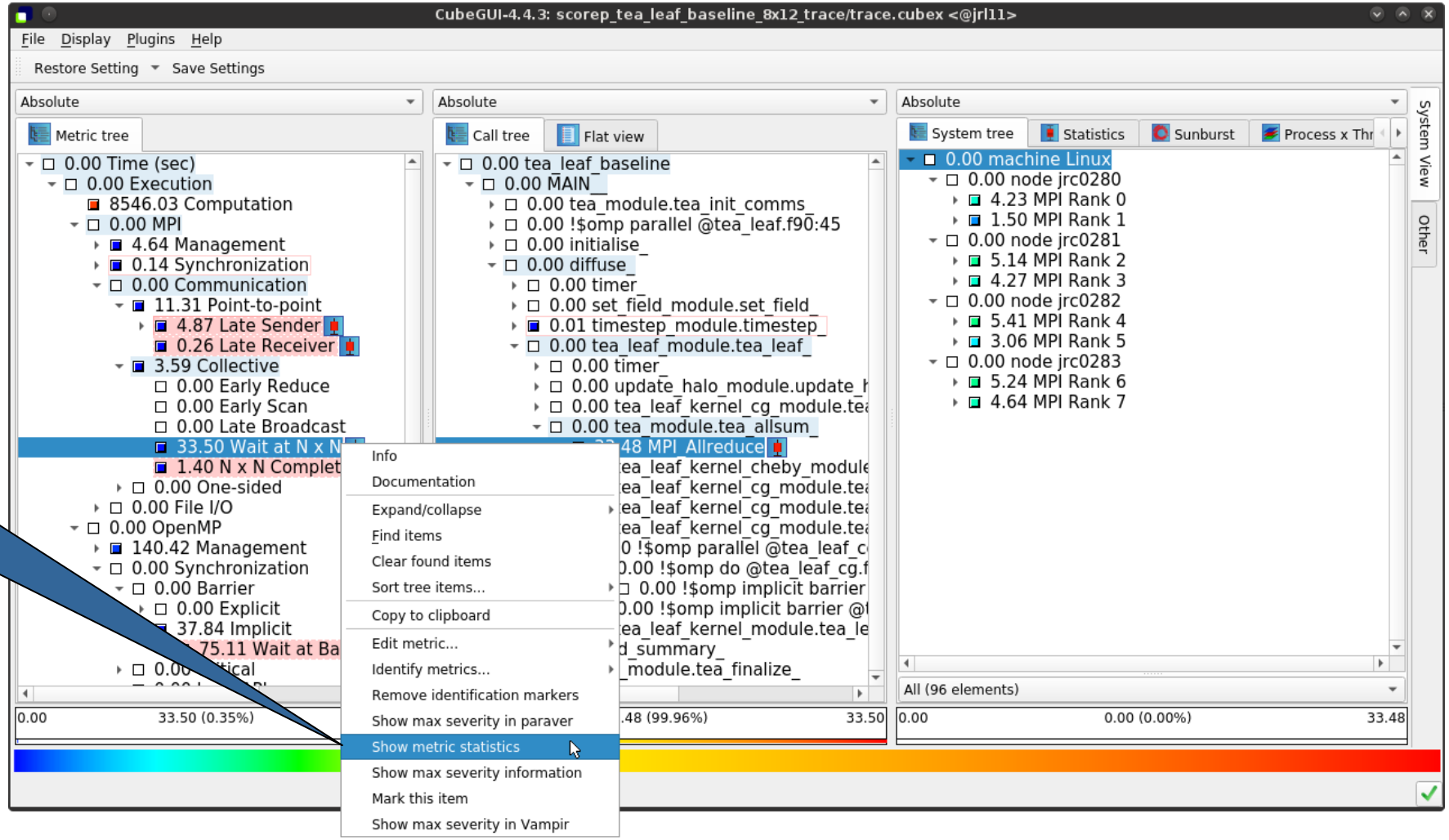
Selection of different metric automatically updates description



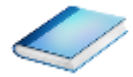
Metric statistics



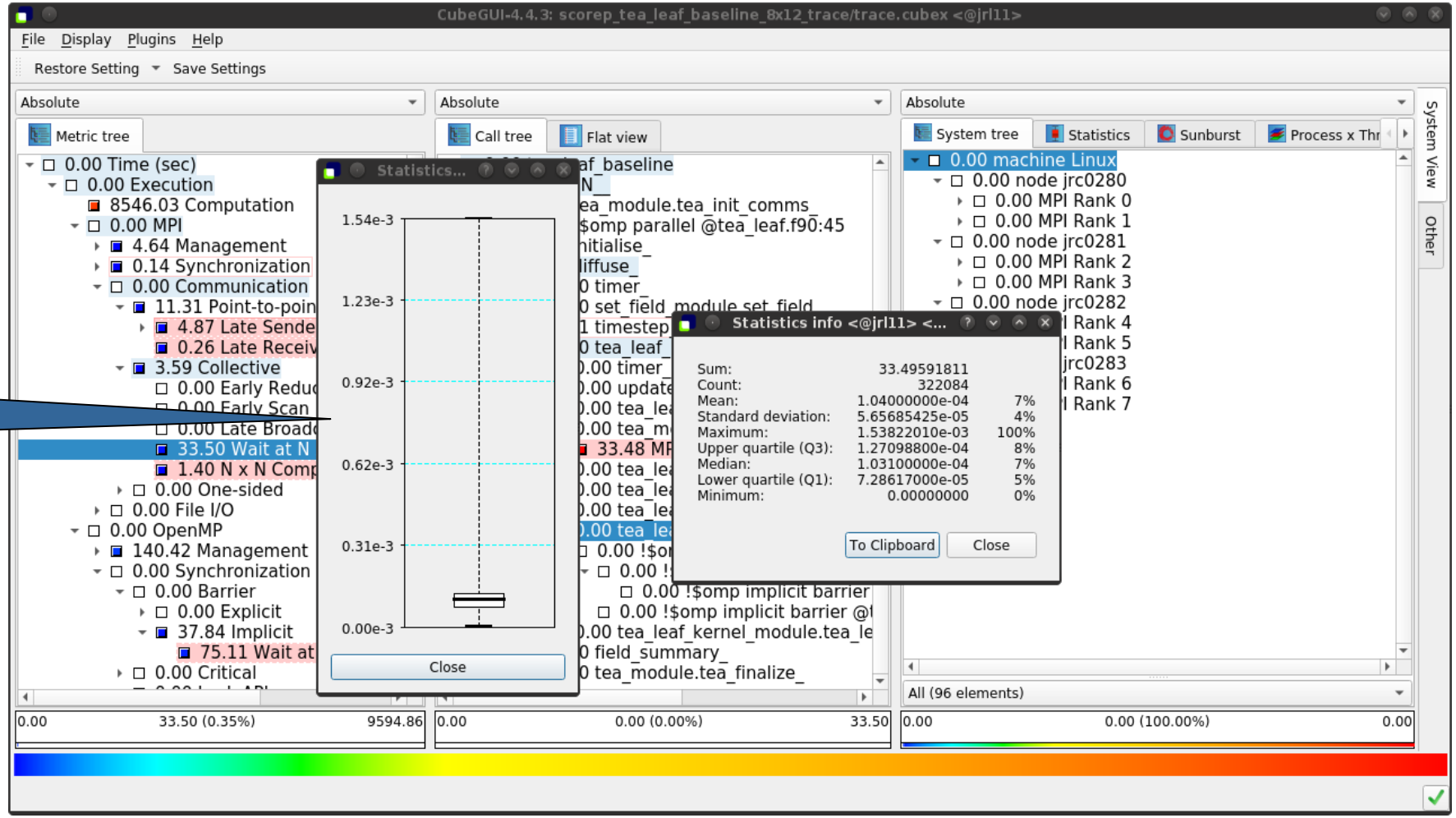
Access metric statistics for metrics marked with box plot icon from context menu



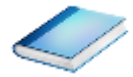
Metric statistics (cont.)



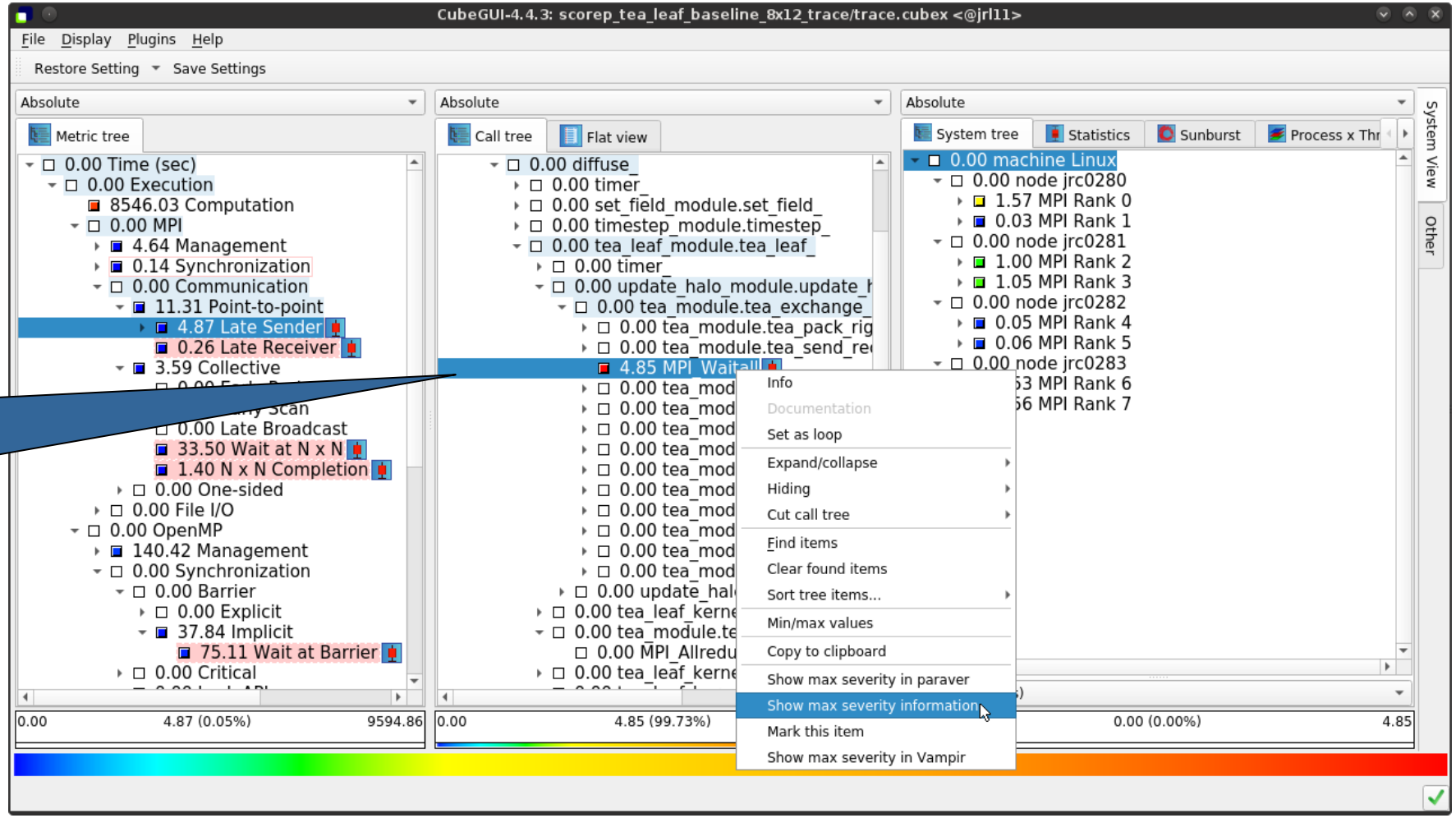
Shows instance statistics box plot, click to get details



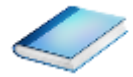
Metric instance statistics



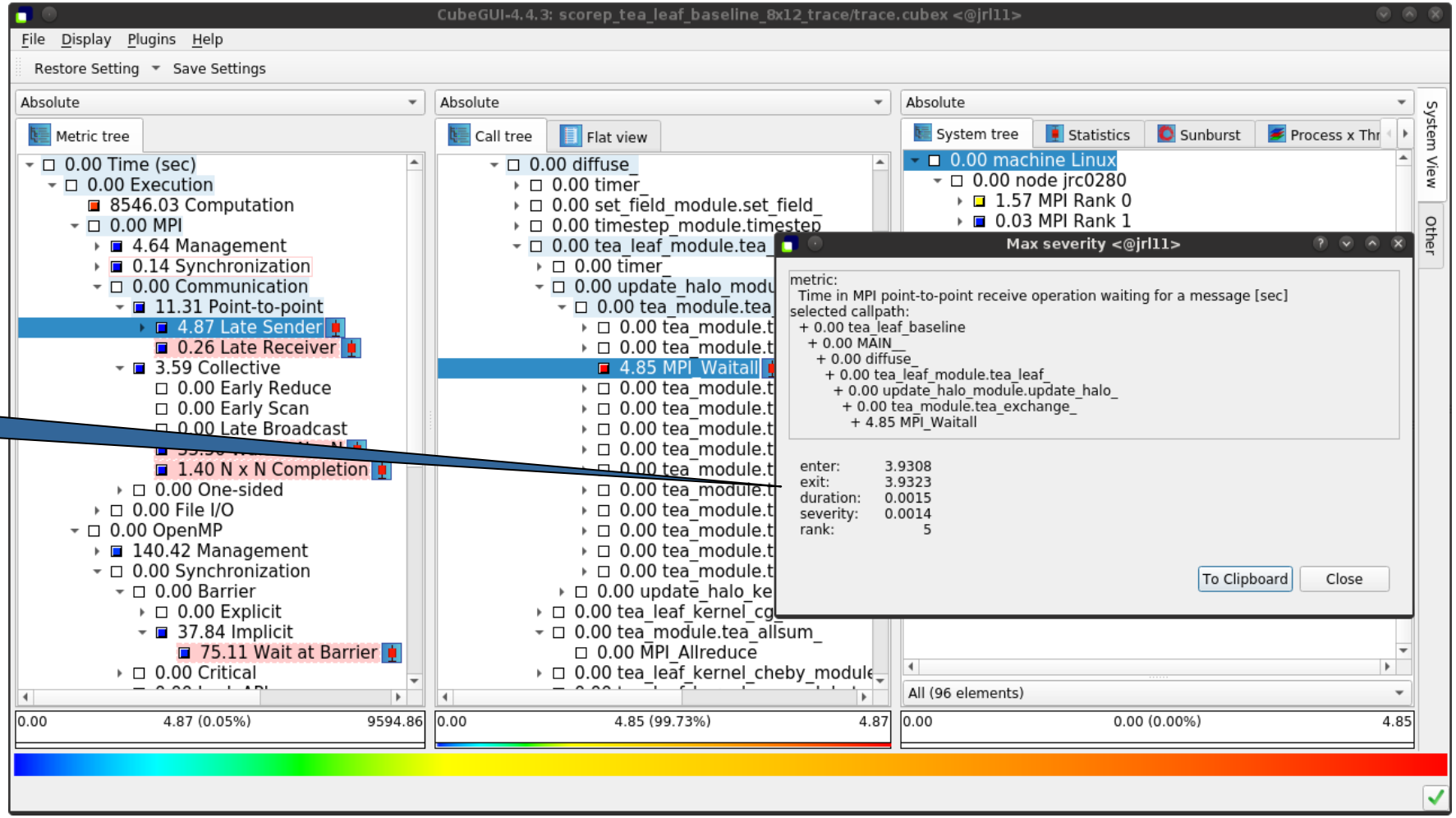
Access most-severe instance information for call paths marked with box plot icon via context menu



Metric instance statistics (cont.)

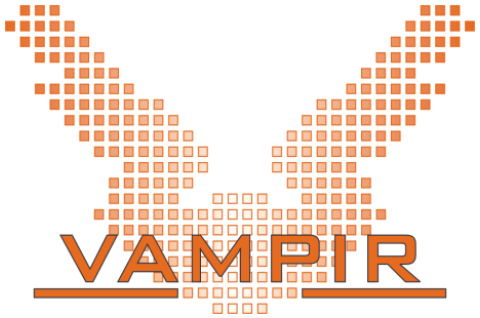


Shows instance details



Interactive visualization and time-interval statistics with Vampir

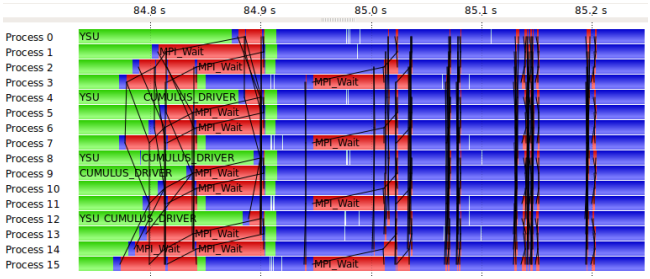
Marc Schlütter, JSC



GWT Gesellschaft für Wissens- und Technologietransfer

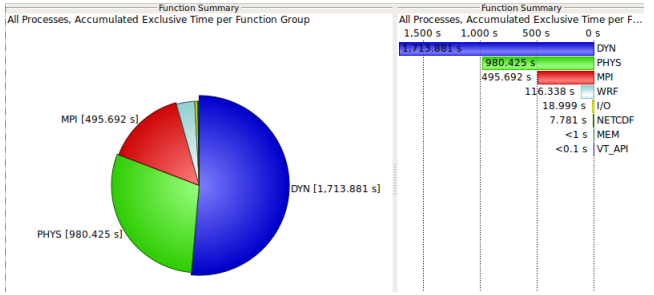
Event Trace Visualization with Vampir

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically at any level of detail
- Provide statistics and performance metrics



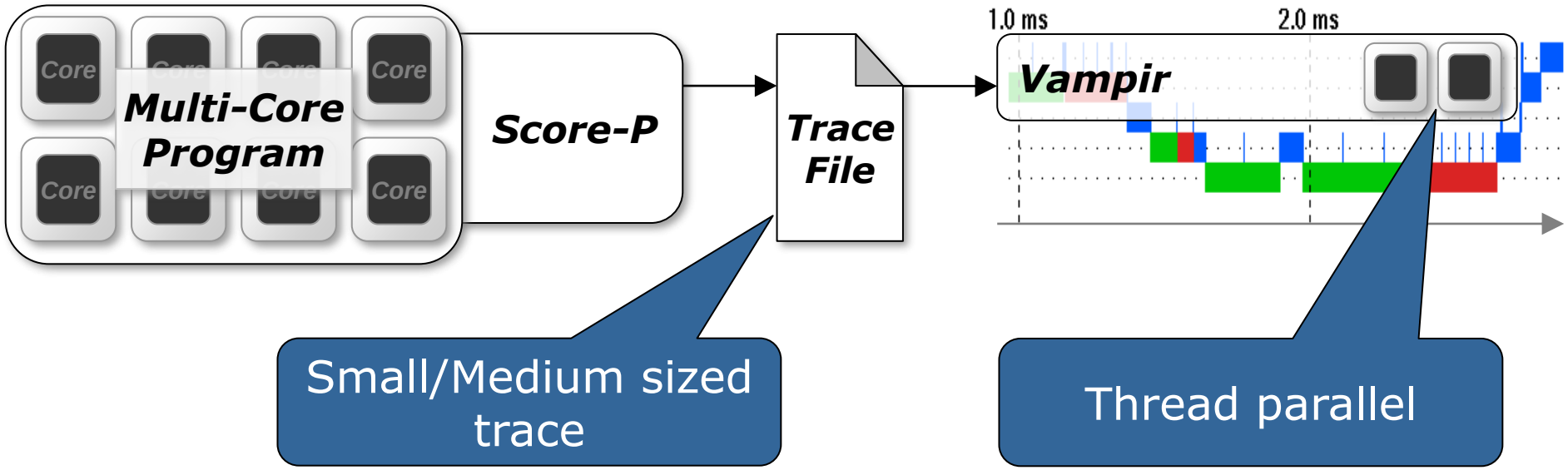
- **Timeline charts**
  - Show application activities and communication along a time axis

- **Summary charts**
  - Provide quantitative results for the currently selected time interval



Visualization Modes (1)
Directly on front end or local machine

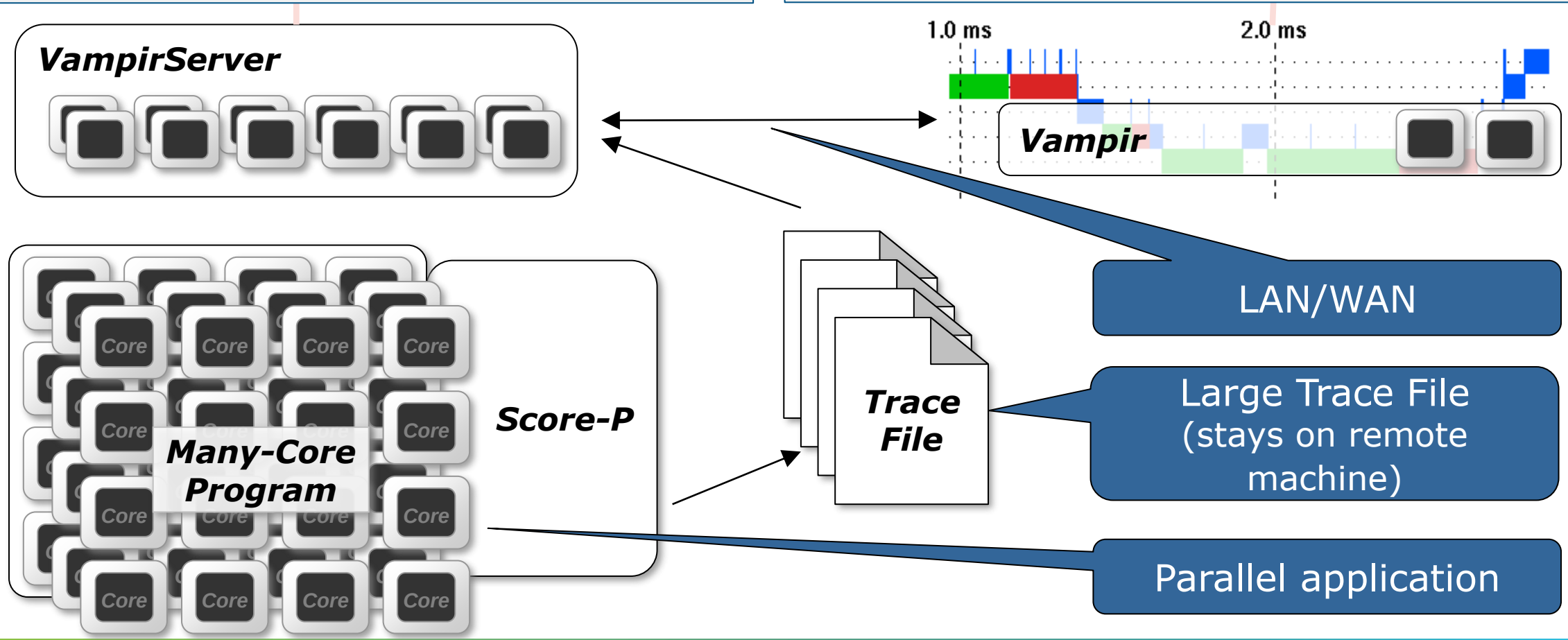
```
% ml vampir
% vampir
```



Visualization Modes (2)
On local machine with remote VampirServer

```
% ml vampir
% vampirserver start
```

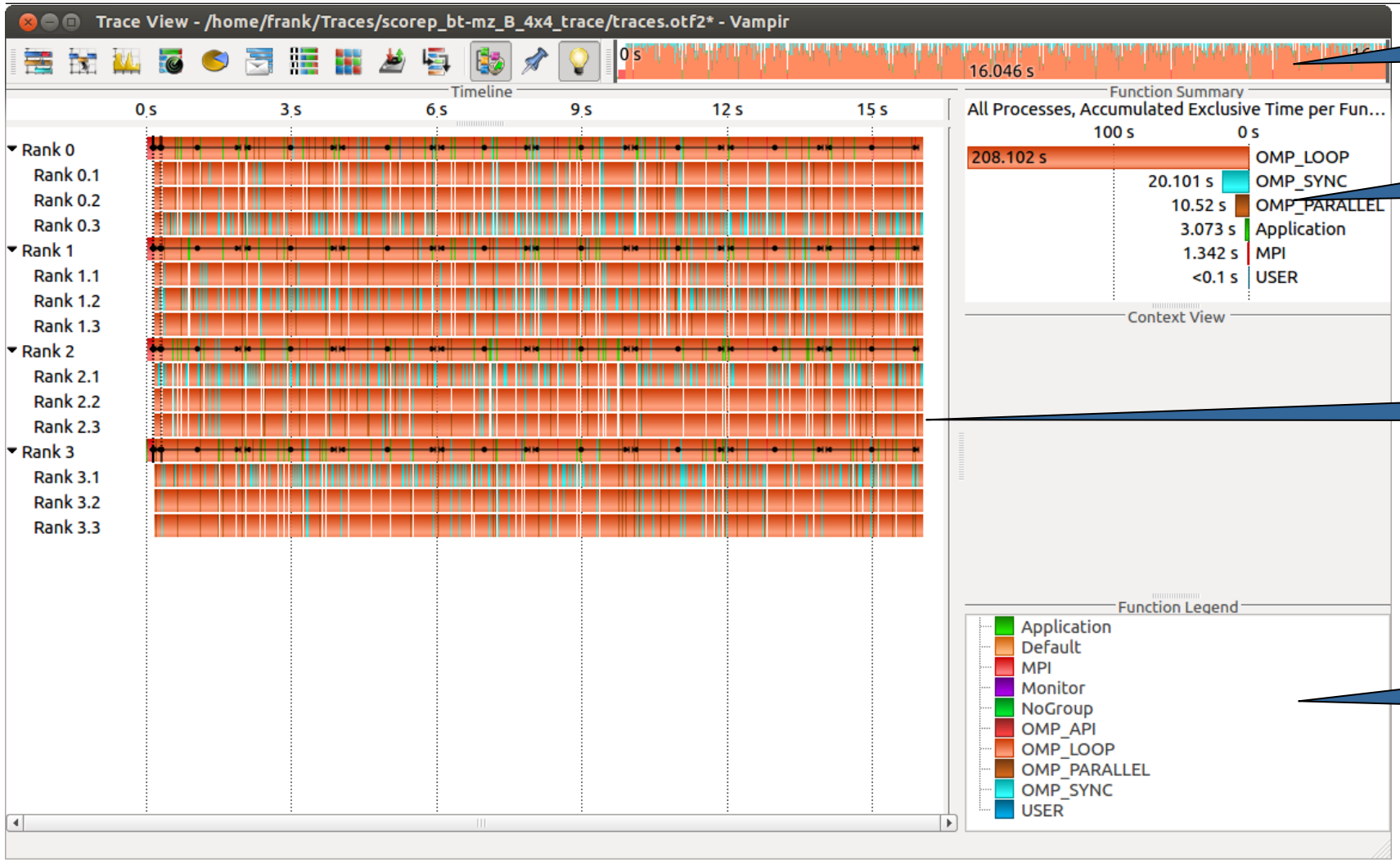
```
% ml vampir
% vampir
```



The main displays of Vampir

- Timeline Charts:
  -  Master Timeline
  -  Process Timeline
  -  Counter Data Timeline
  -  Performance Radar
- Summary Charts:
  -  Function Summary
  -  Message Summary
  -  Process Summary
  -  Communication Matrix View

Visualization of the NPB-MZ-MPI / BT trace



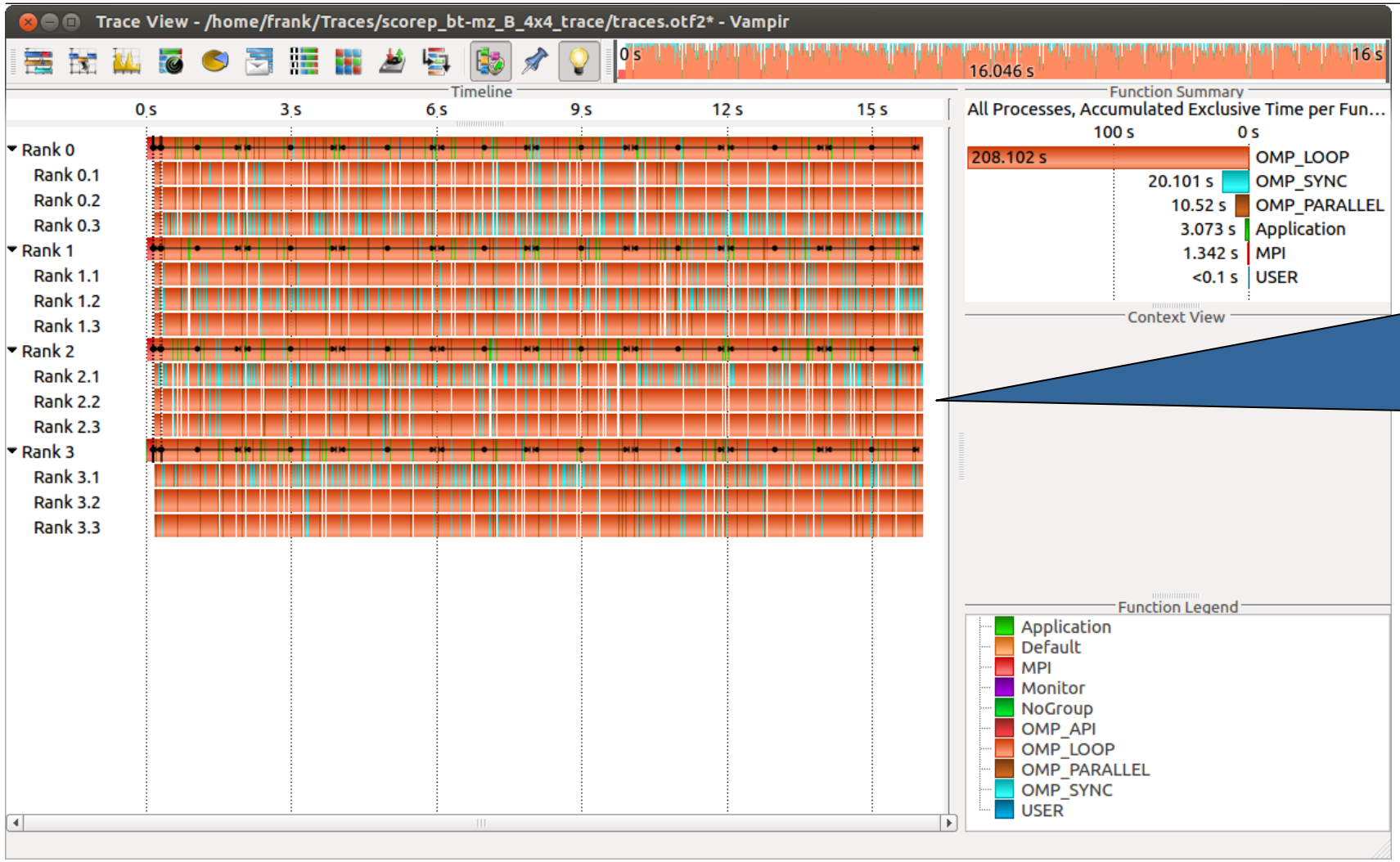
Navigation Toolbar

Function Summary

Master Timeline

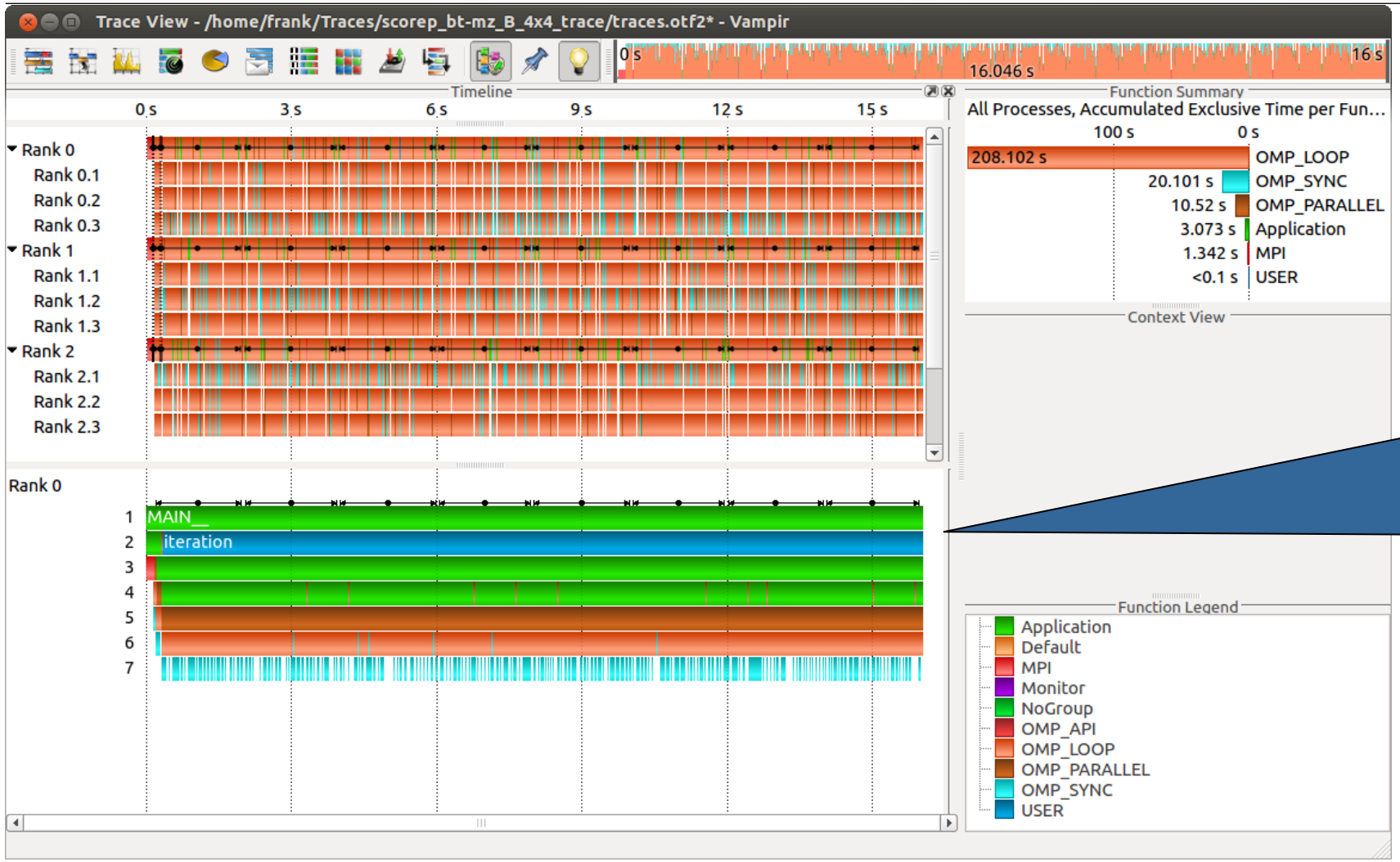
Function Legend

Visualization of the NPB-MZ-MPI / BT trace
Master Timeline



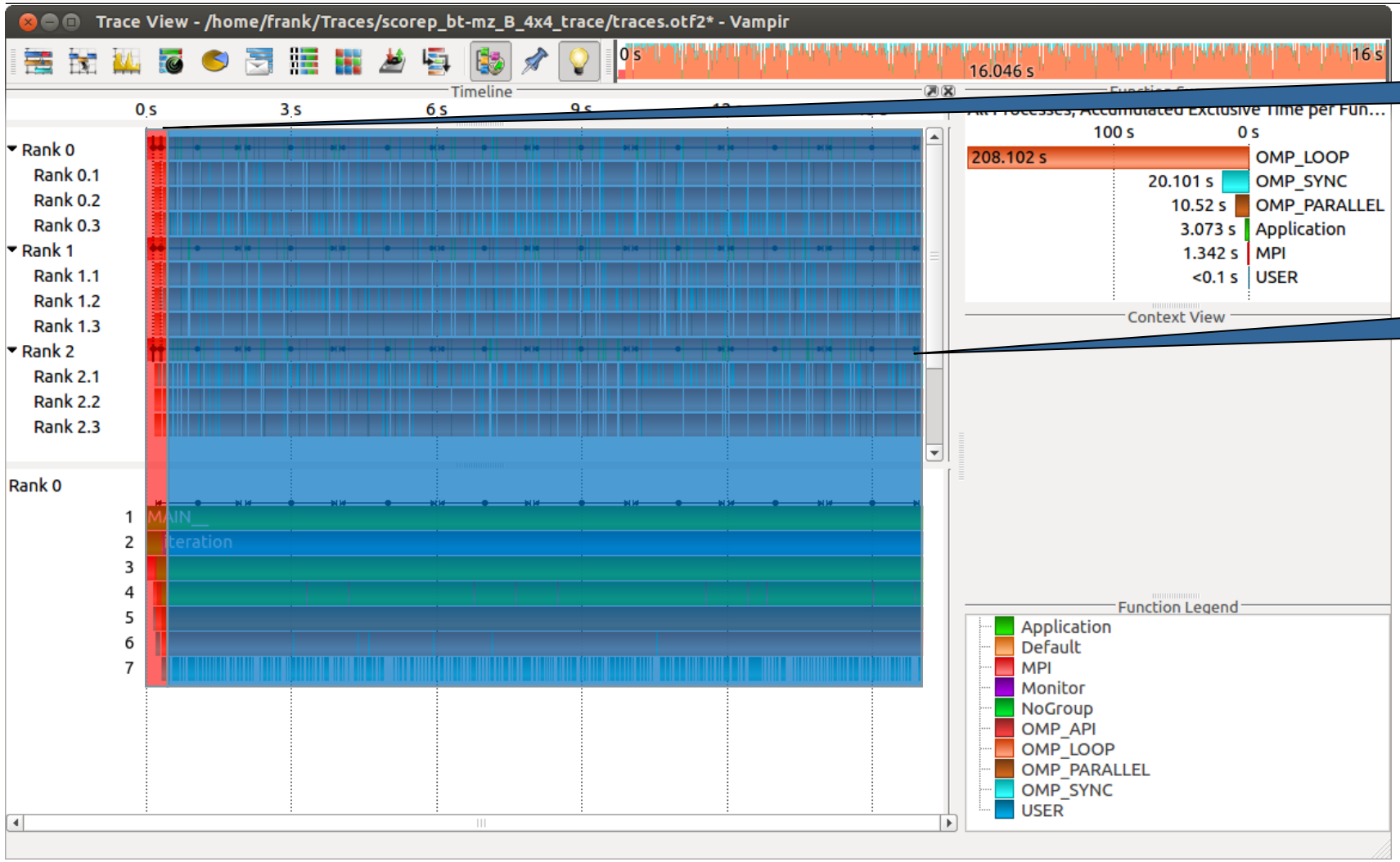
Detailed information about functions, communication and synchronization events for collection of processes.

Visualization of the NPB-MZ-MPI / BT trace
Process Timeline



Detailed information about different levels of function calls in a stacked bar chart for an individual process.

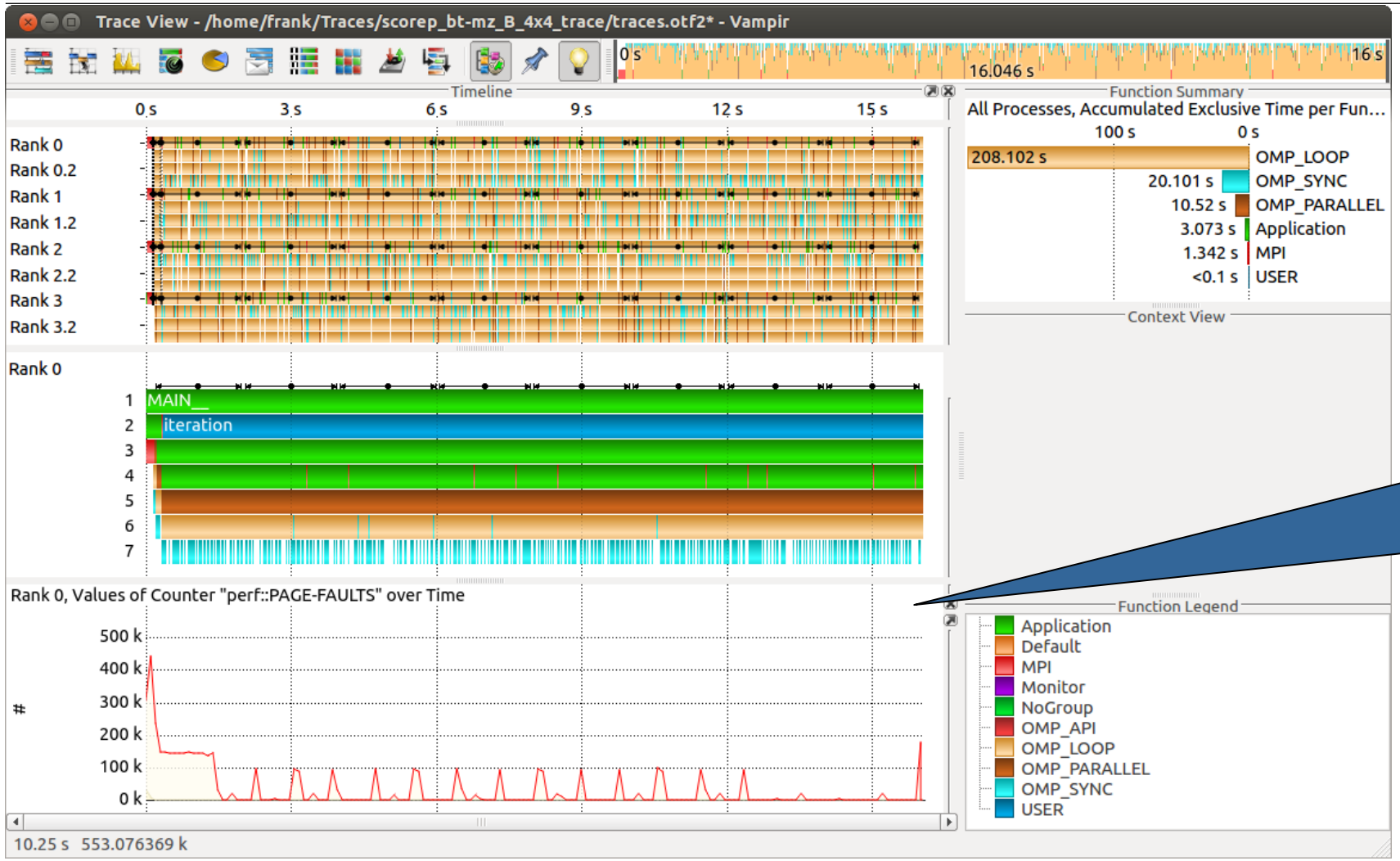
Visualization of the NPB-MZ-MPI / BT trace
Typical program phases



Initialisation Phase

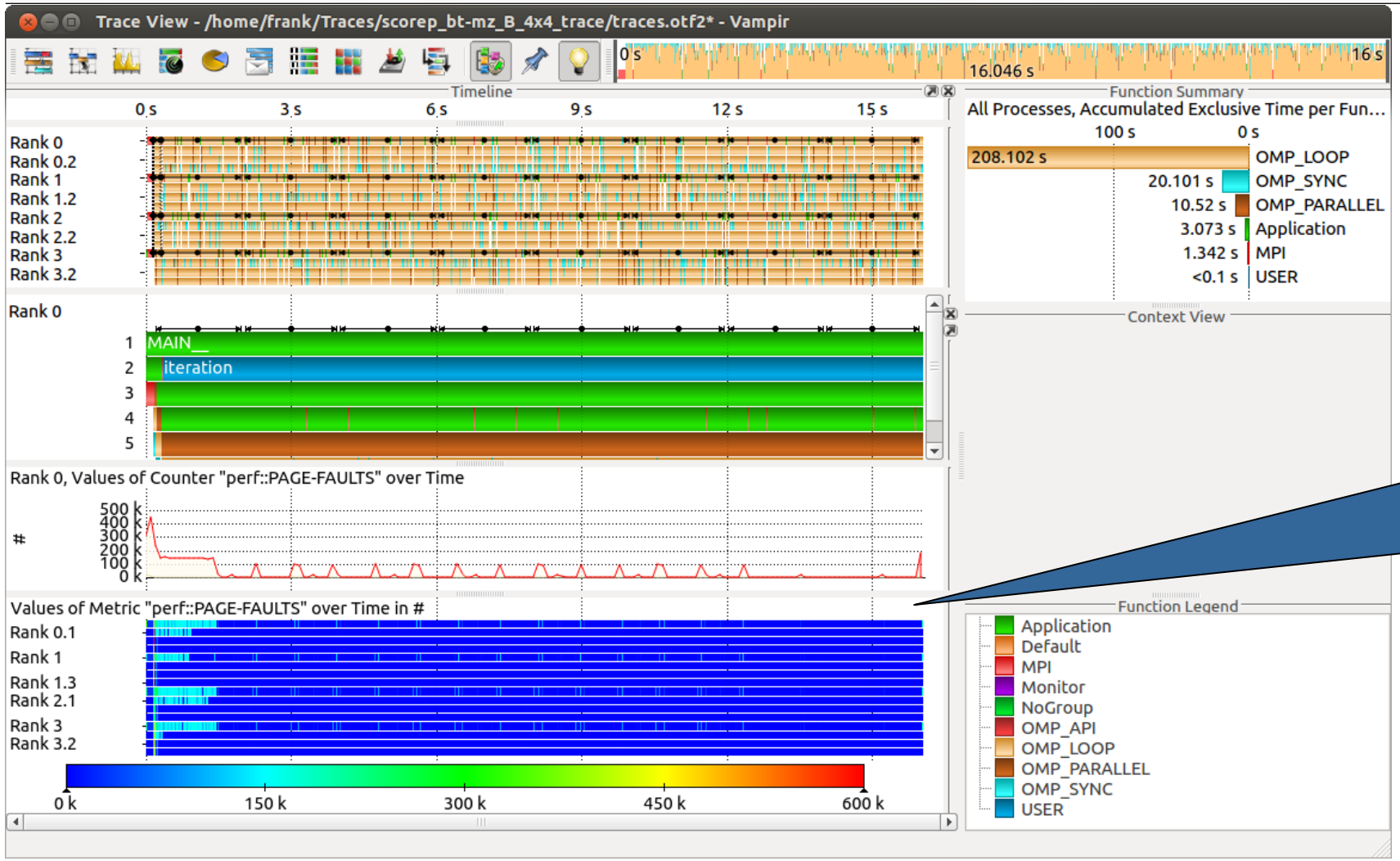
Computation Phase

Visualization of the NPB-MZ-MPI / BT trace
Counter Data Timeline



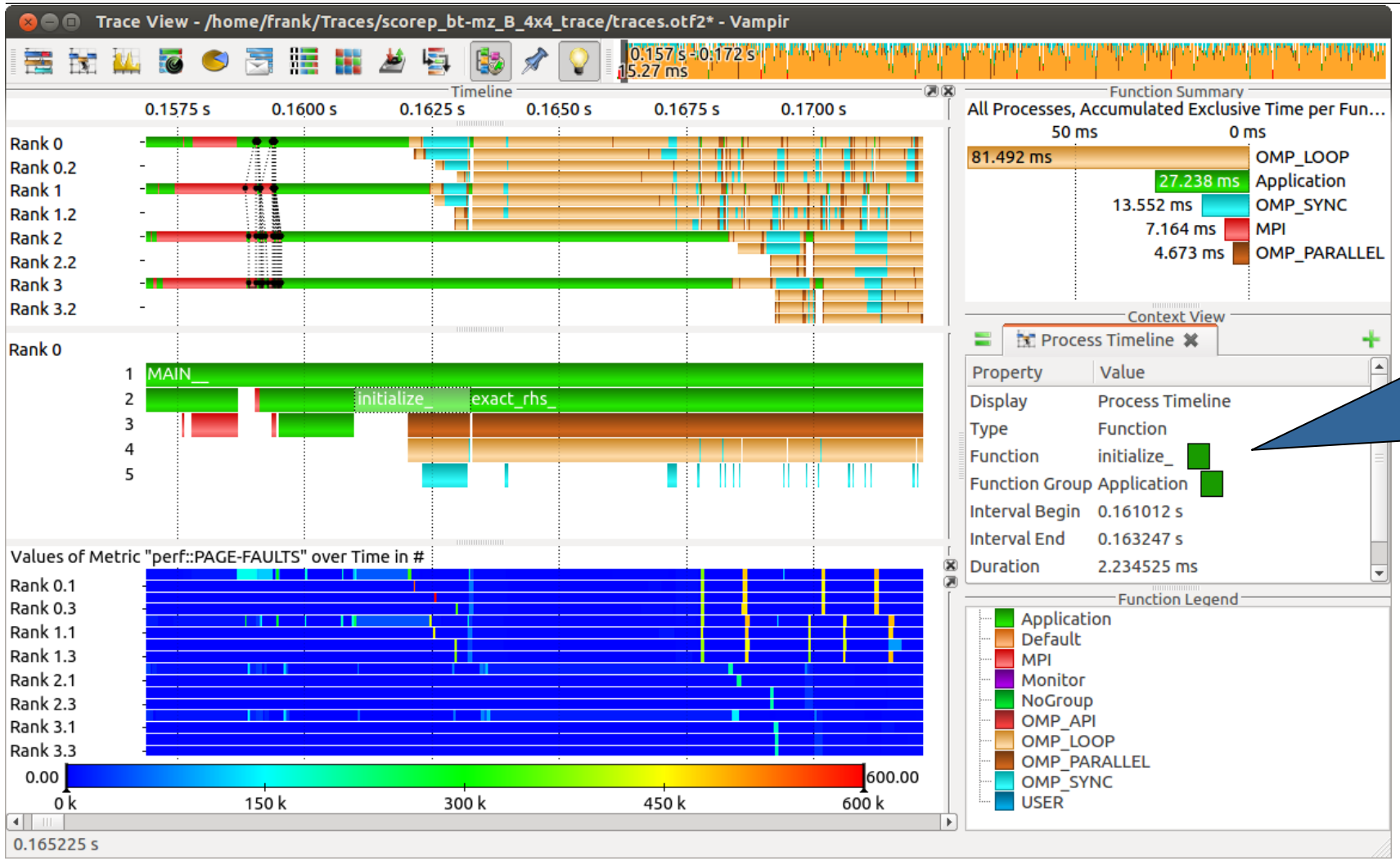
Detailed counter information over time for an individual process.

Visualization of the NPB-MZ-MPI / BT trace
Performance Radar



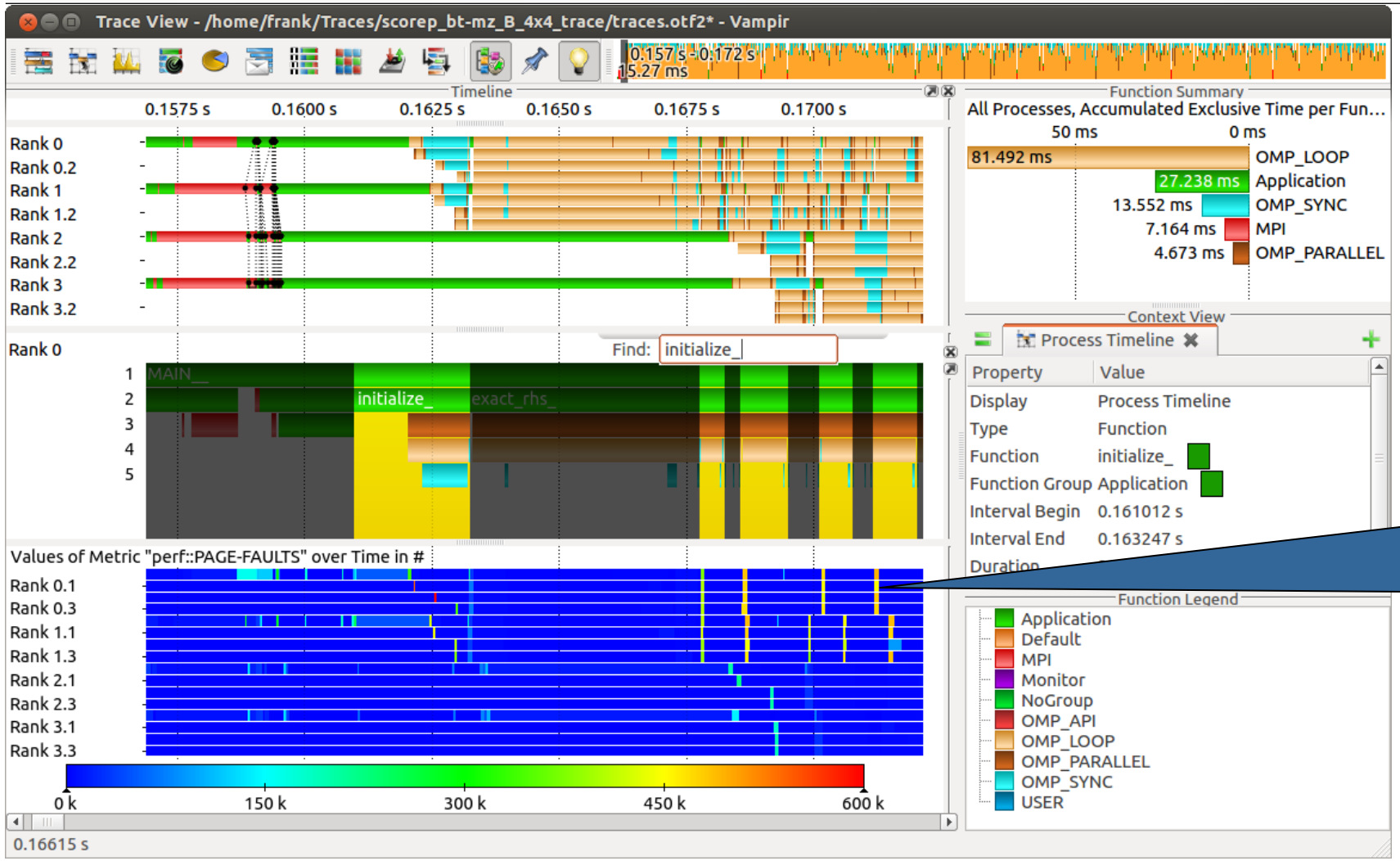
Detailed counter information over time for a collection of processes.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Inititialisation Phase



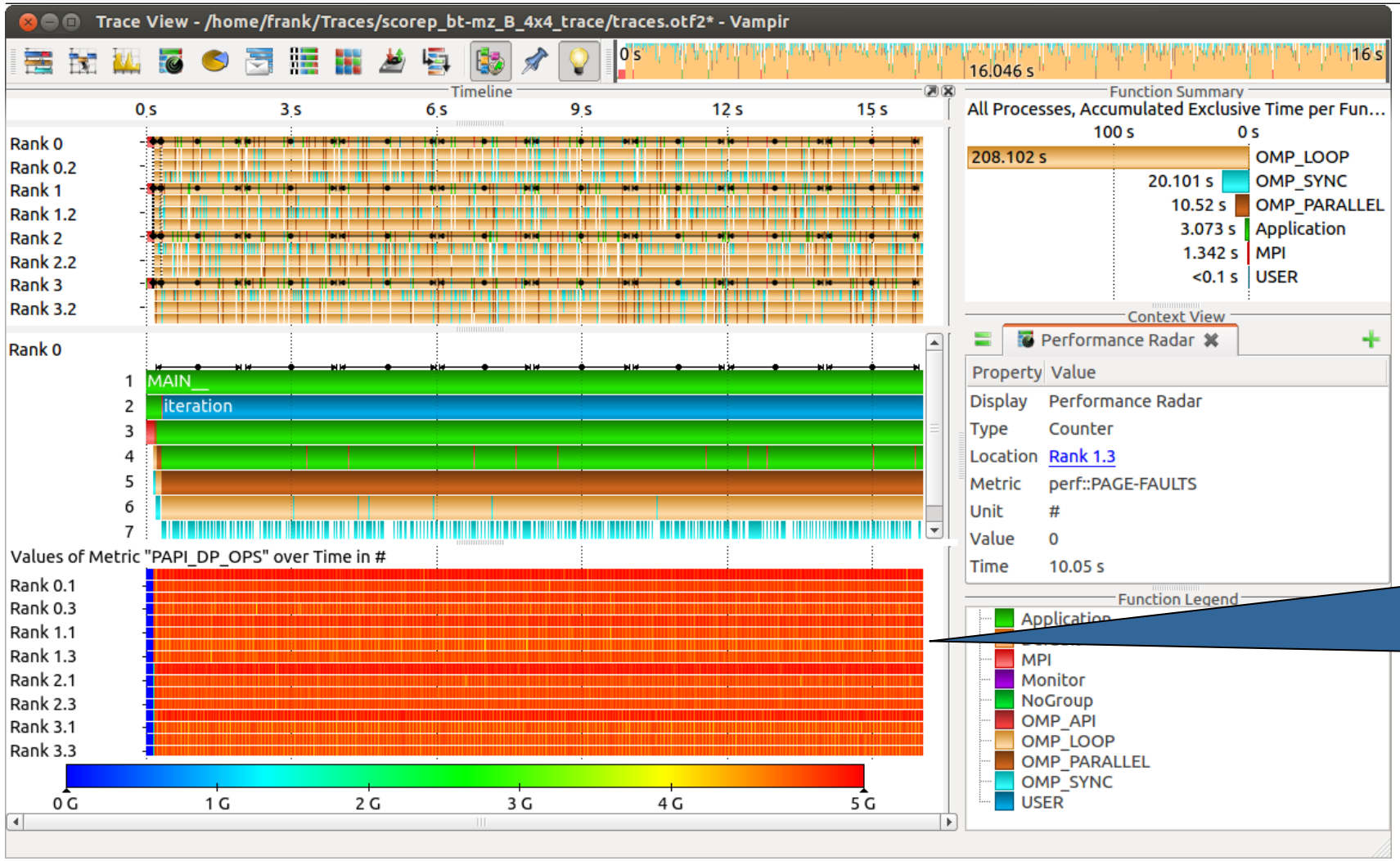
Context View: Detailed information about function “initialize_”.

Visualization of the NPB-MZ-MPI / BT trace
Find Function



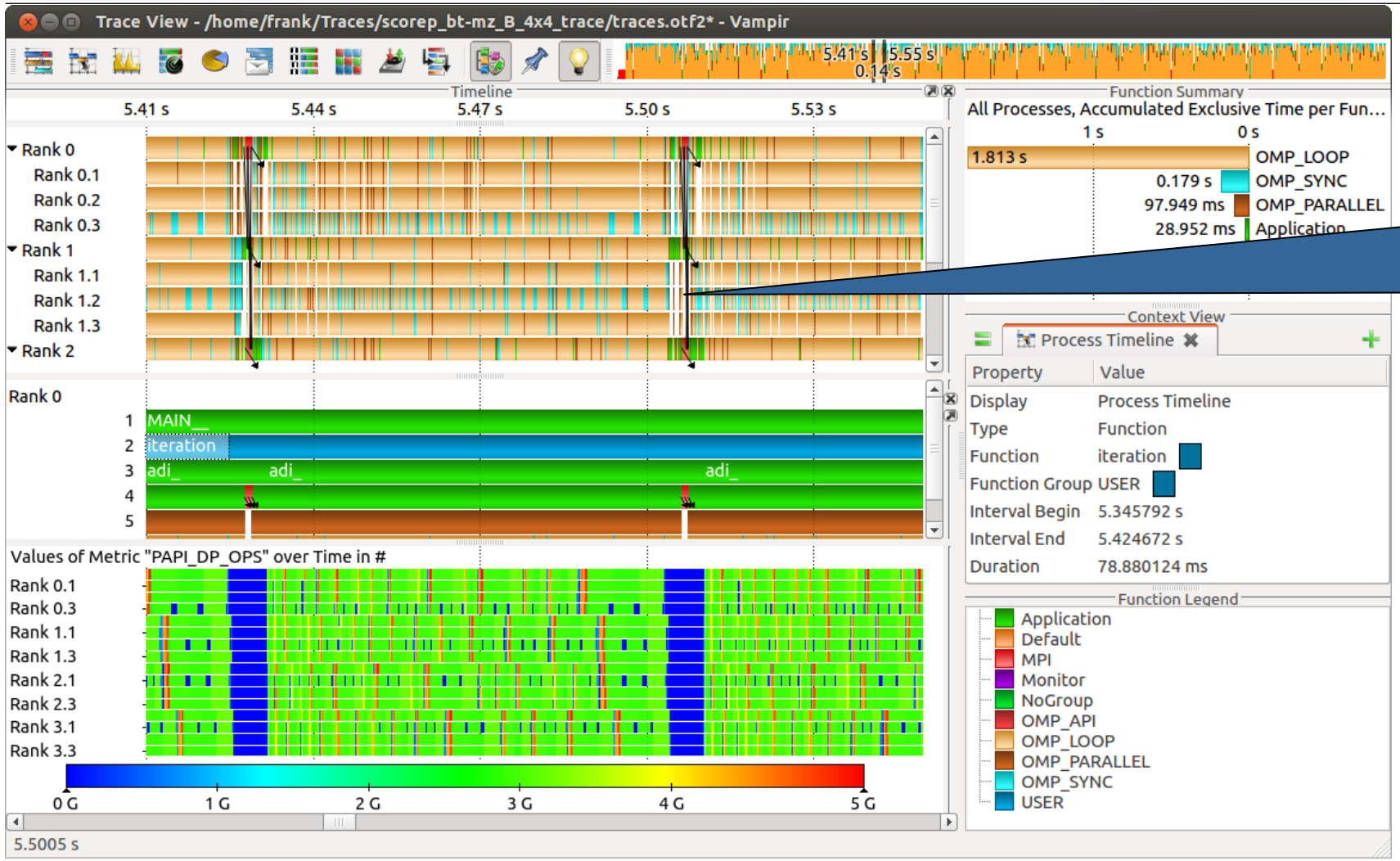
Execution of function "initialize_" results in higher page fault rates.

Visualization of the NPB-MZ-MPI / BT trace
Computation Phase



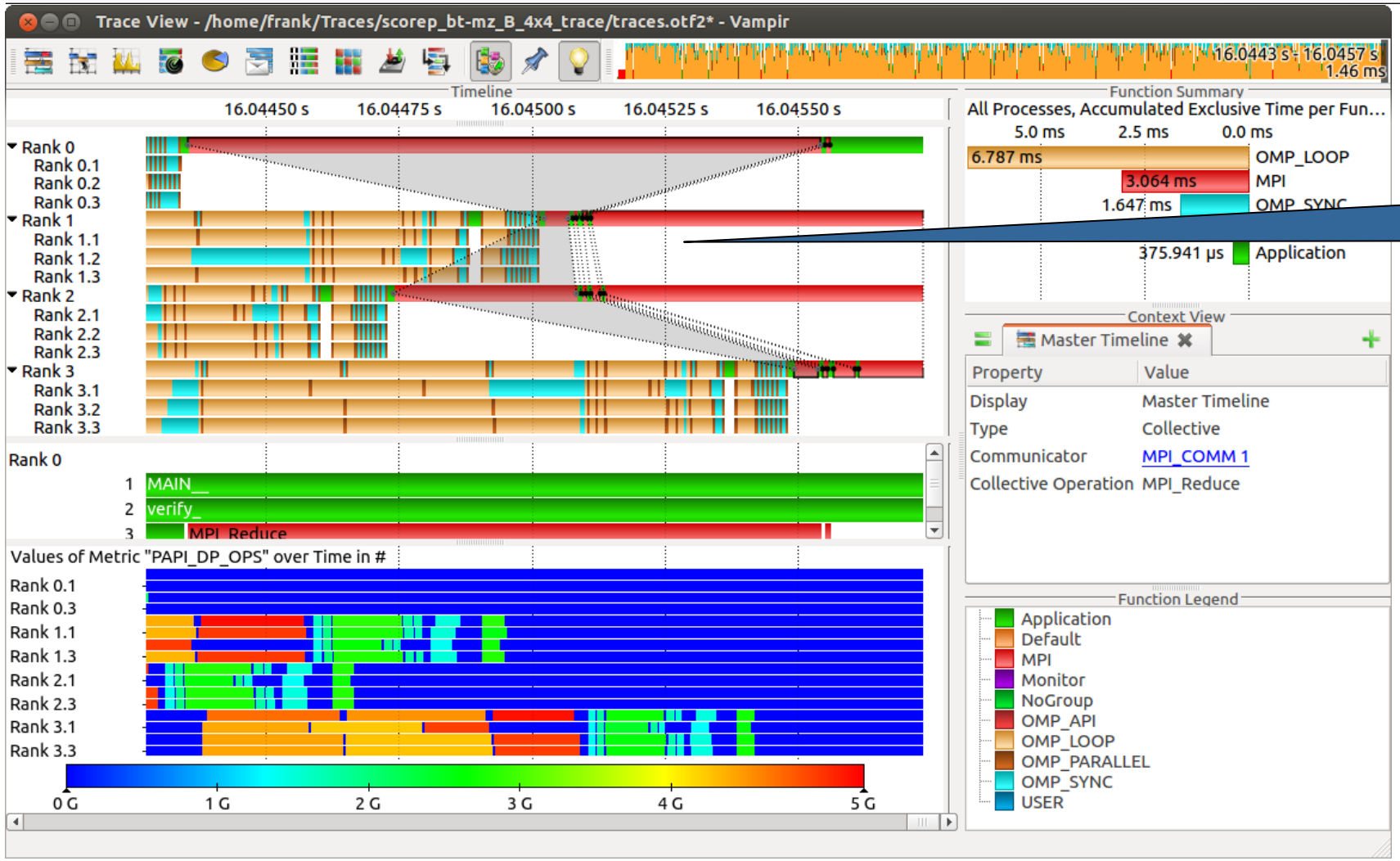
Computation phase results in higher floating point operations.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Computation Phase



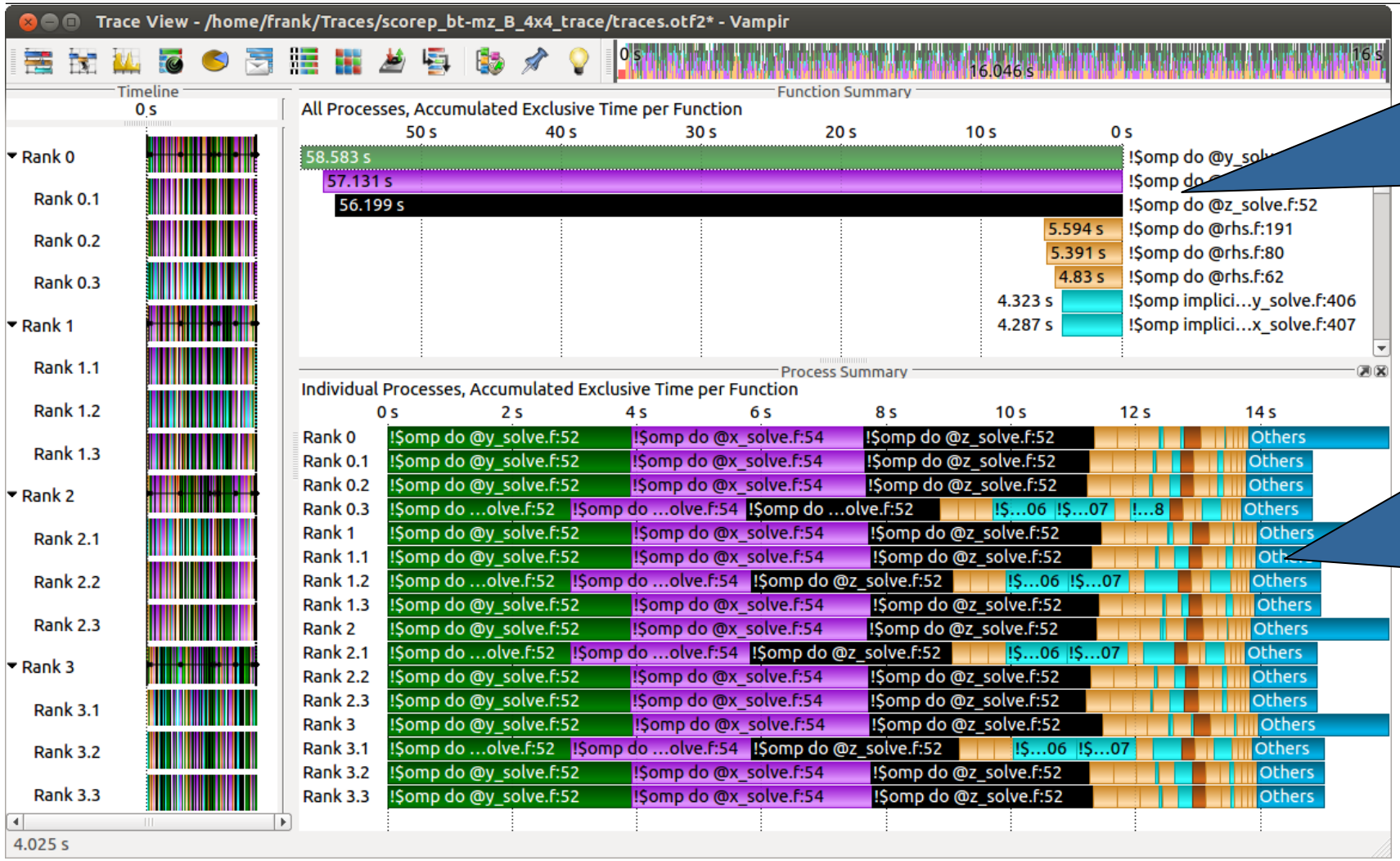
MPI communication results in lower floating point operations.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Finalisation Phase



"Early reduce" bottleneck.

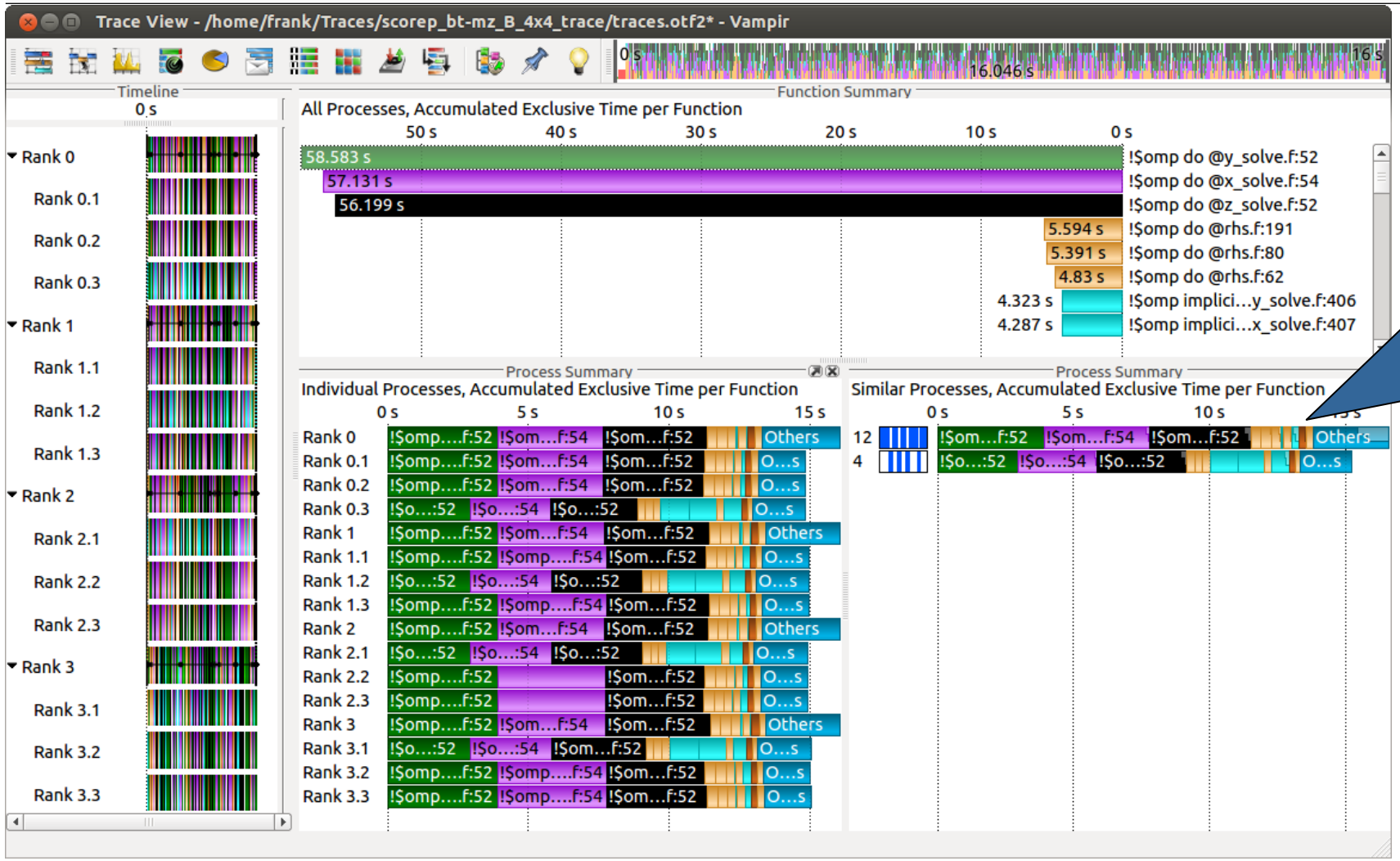
Visualization of the NPB-MZ-MPI / BT trace
Process Summary



Function Summary: Overview of the accumulated information across all functions and for a collection of processes.

Process Summary: Overview of the accumulated information across all functions and for every process independently.

Visualization of the NPB-MZ-MPI / BT trace
Process Summary



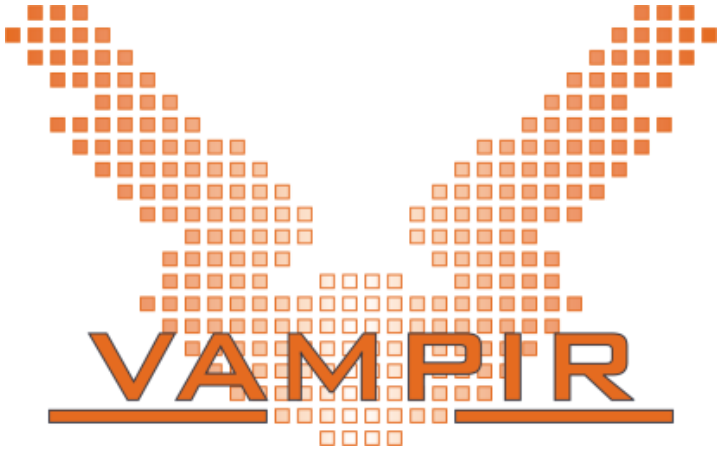
Find groups of similar processes and threads by using summarized function information.

Summary

- Scalable visualization of event traces
- Color coding activities to easily identify program structure
- Client-Server (MPI) architecture to utilize HPC resources
- Supports multiple trace formats produced by different measurement tools
  - OTF2
    - Score-P
    - lo2s
    - TAU
    - Intel Trace Analyzer[1]
    - The Structural Simulation Toolkit[2]
  - Chrome Trace Format
    - TensorFlow
    - PyTorch
    - Cmake build
  - WfCommons
    - Fireworks
    - RADICAL
    - Nextflow
    - Snakemake

[1] *https://www.intel.com/content/www/us/en/docs/trace-analyzer-collector/user-guide-reference/2022-2/otf2-format-support.html*
[2] *http://sst-simulator.org*

GWT Gesellschaft für Wissens- und Technologietransfer



https://vampir.eu

service@vampir.eu