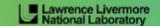
# **Automatic trace analysis with the Scalasca Trace Tools**

The Scalasca Team
Jülich Supercomputing Centre

















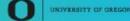












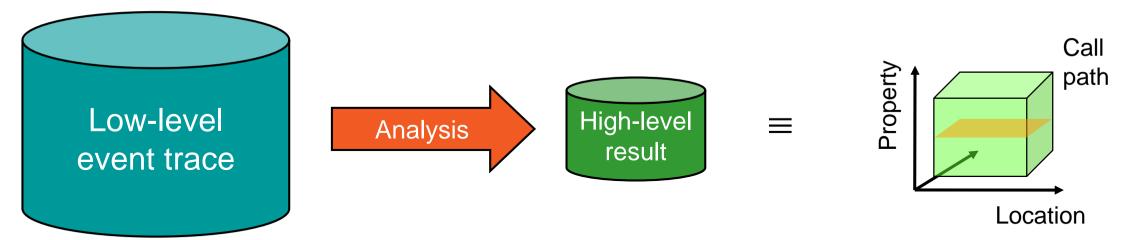




#### **Automatic trace analysis**

#### 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



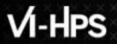
#### **Scalasca Trace Tools: Objective**

- Development of a scalable trace-based performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
- 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 v2.6 coordinated with Score-P v7.0 (April 2021), also works with later versions

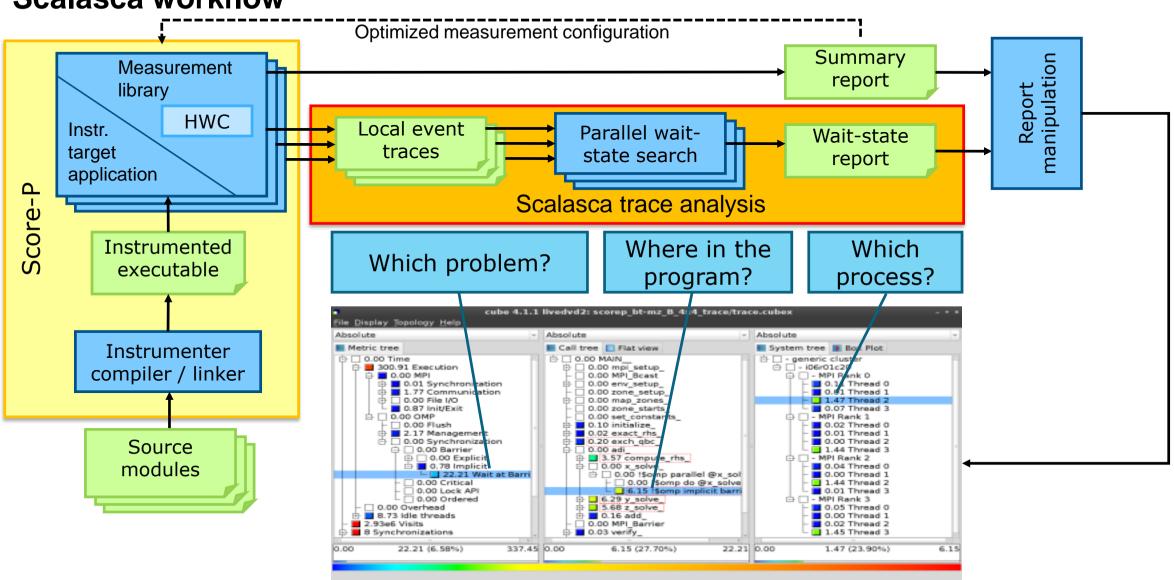


#### **Scalasca Trace Tools: Features**

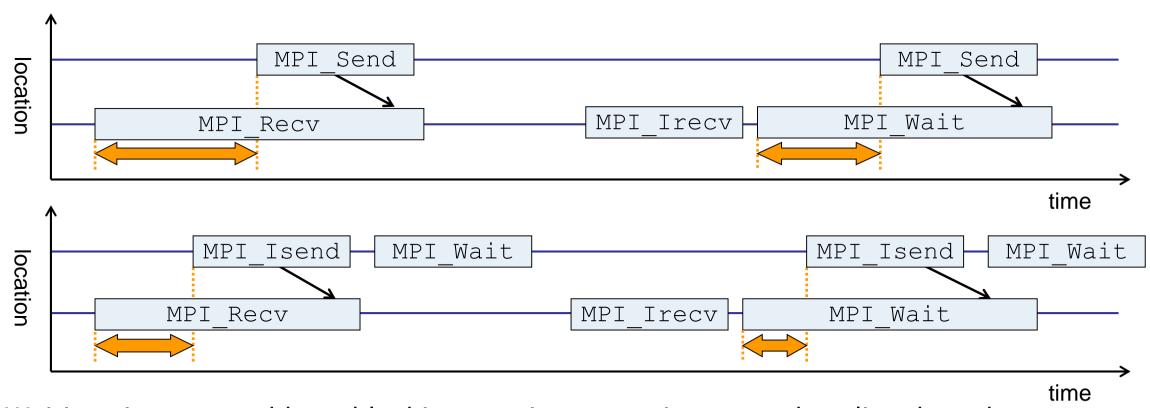
- Open source, 3-clause BSD license
- Fairly portable
  - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX systems, Linux clusters (x86, Power, ARM), Intel Xeon Phi, ...
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Supports 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 Memory events, CUDA/OpenCL device events (kernel, memcpy), SHMEM, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored



#### Scalasca workflow



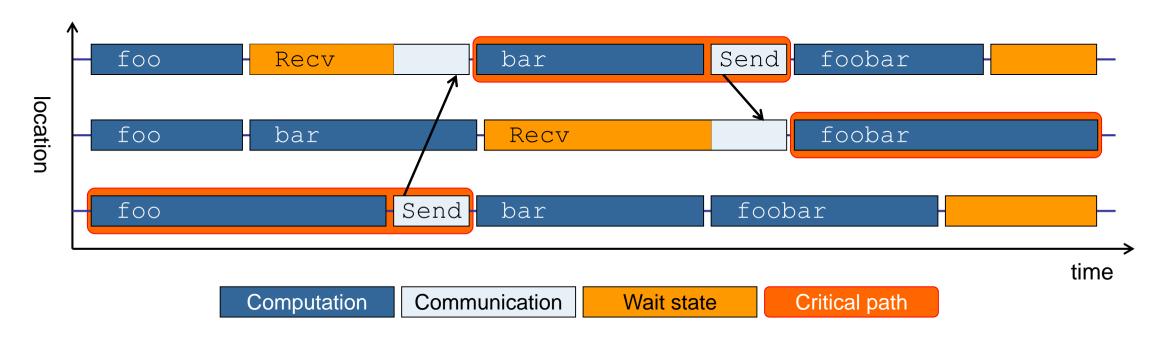
#### Example: "Late Sender" wait state



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



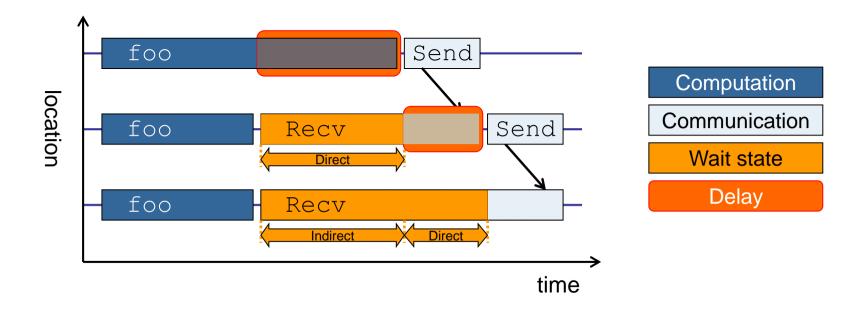
#### **Example: Critical path**



- 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**



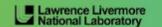
- Classifies wait states into direct and indirect (i.e., caused by other wait states)
- Identifies *delays* (excess computation/communication) as root causes of wait states
- Attributes wait states as delay costs

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



































#### Scalasca command - One command for (almost) everything

```
% scalasca
Scalasca 2.6
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
                         show quick reference quide and exit
      --auickref
      --remap-specfile show path to remapper specification file and exit
   -v, --verbose
                         enable verbose commentary
                         show version information and exit
   -V, --version
```

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



#### Scalasca convenience command: scan / scalasca -analyze

```
% scan
Scalasca 2.6: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
      where {options} may include:
       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.
       Analvze
                  : 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 cfafile
                  : Specify a config file for a multi-run measurement.
```

Scalasca measurement collection & analysis nexus



#### **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)



#### Scalasca convenience command: square / scalasca -examine

```
% square
Scalasca 2.6: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
   -c <none | quick | full> : Level of sanity checks for newly created reports
                            : Force remapping of already existing reports
   -F
  -f filtfile
                            : Use specified filter file when doing scoring (-s)
                            : Skip display and output textual score report
  -s
                            : Enable verbose mode
                            : Do not include idle thread metric
   -n
                            : Aggregation method for summarization results of
   -S <mean | merge>
                              each configuration (default: merge)
   -T <mean | merge>
                            : Aggregation method for trace analysis results of
                              each configuration (default: merge)
                            : Post-process every step of a multi-run experiment
   -A
```

Scalasca analysis report explorer (Cube)



## **Recap - Local installation (JUSUF)**

■ Load default environment (GCC + ParaStationMPI):

```
% module load GCC ParaStationMPI
```

Load the modules for the tool environment:

```
% module load Score-P Scalasca CubeGUI
```

- Copy tutorial sources to your WORK directory (or your personal workspace)
  - Only required if not done already (for opening exercise)

```
% cd $SCRATCH_training2123/$USER
% tar zxvf $PROJECT_training2208/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

## **BT-MZ** summary measurement collection...

```
% cd bin.scorep
% cp ../jobscript/jusuf/scalasca2.sbatch .
% cat scalasca2.sbatch
# Scalasca nexus configuration for profiling
#NEXUS="scalasca -analyze"
# Scalasca nexus configuration for profiling
#NEXUS="scalasca -analyze -t"
# Score-P measurement configuration
export SCOREP FILTERING FILE=../config/scorep.filt
#export SCOREP TOTAL MEMORY=250M
# run the application
scalasca -analyze mpiexec -n $SLURM NTASKS ./bt-mz $CLASS.$PROCS
```

Change to
 directory with the
 Score-P
 instrumented
 executable and
 edit the job script

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

Submit the job

% sbatch -account=training2208 scalasca2.sbatch

#### **BT-MZ summary measurement**

```
S=C=A=N: Scalasca 2.6 runtime summarization
S=C=A=N: ./scorep bt-mz C 8x6 sum experiment archive
S=C=A=N: Mon Apr 4 16:29:24 2022: Collect start
mpirun./bt-mz B.28
 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:
 [... More application output ...]
S=C=A=N: Mon Apr 4 16:29:41 2022: Collect done (status=0) 17s
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 report (profile.cubex)...
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]
```

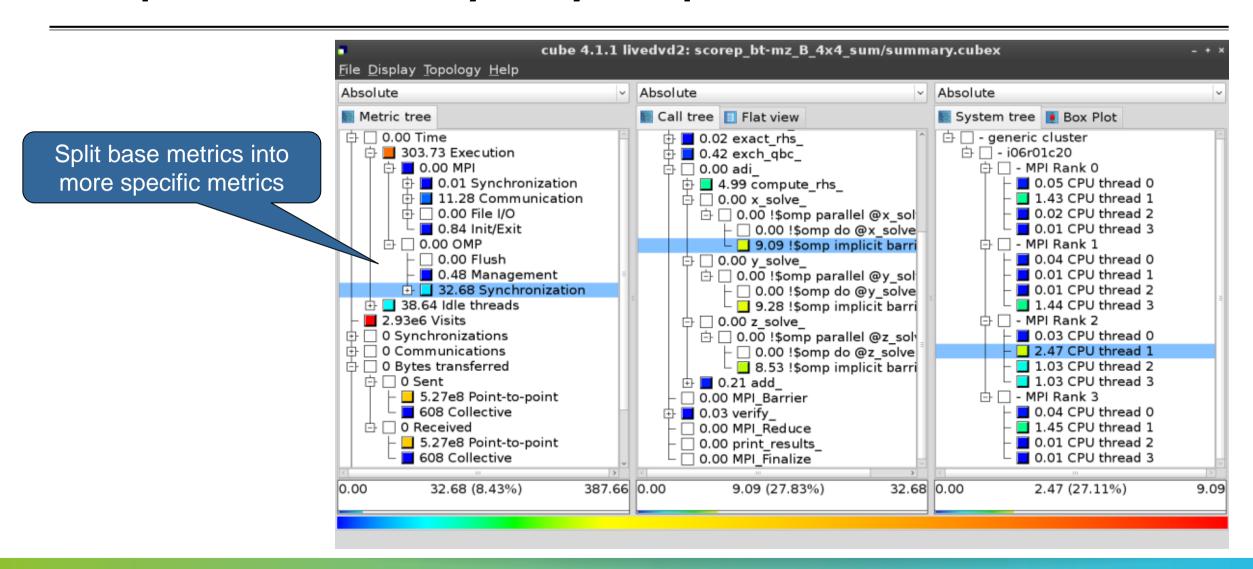
#### Hint:

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

# VI-HPS

#### Post-processed summary analysis report



## **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/jusuf/scalasca2.sbatch .
% vim scalasca2.sbatch
# Scalasca nexus configuration for profiling
#NEXUS="scalasca -analyze"
# Scalasca nexus configuration for profiling
#NEXUS="scalasca -analyze -t"
# Score-P measurement configuration
export SCOREP FILTERING FILE=../config/scorep.filt
export SCOREP TOTAL MEMORY=250M
# run the application
scalasca -analyze -t mpiexec -n $SLURM NTASKS ./bt-mz $CLASS.$PROCS
```

 Change to directory with the Score-P instrumented executable and edit the job script

- Add "-t" to the scan command
- Submit the job

% sbatch -account=training2208 scalasca2.sbatch

#### BT-MZ trace measurement ... collection

```
S=C=A=N: Scalasca 2.6 trace collection and analysis
S=C=A=N: Mon Apr 4 16:45:15 2022: Collect start
mpirun./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 Apr 4 16:45:34 2022: Collect done (status=0) 19s
```

 Starts measurement with collection of trace files



#### BT-MZ trace measurement ... analysis

```
S=C=A=N: Mon Apr 4 16:45:34 2022: Analyze start
mpirun scout.hyb --time-correct \
        ./scorep bt-mz C 8x6 trace/traces.otf2
         (Scalasca 2.6)
SCOUT
Analyzing experiment archive ./scorep bt-mz C 8x6 trace/traces.otf2
Opening experiment archive ... done (0.007s).
Reading definition data ... done (0.007s).
Reading event trace data ... done (0.185s).
Preprocessing ... done (0.306s).
Analyzing trace data ... done (7.524s).
Writing analysis report ... done (0.343s).
Max. memory usage : 870.453MB
Total processing time : 8.513s
S=C=A=N: Mon Apr 4 16:45:43 2022: Analyze done (status=0) 9s
```

 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 report (profile.cubex)...
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_bt-mz_C_8x6_trace/trace.cubex...

[GUI showing trace analysis report]
```

#### Hint:

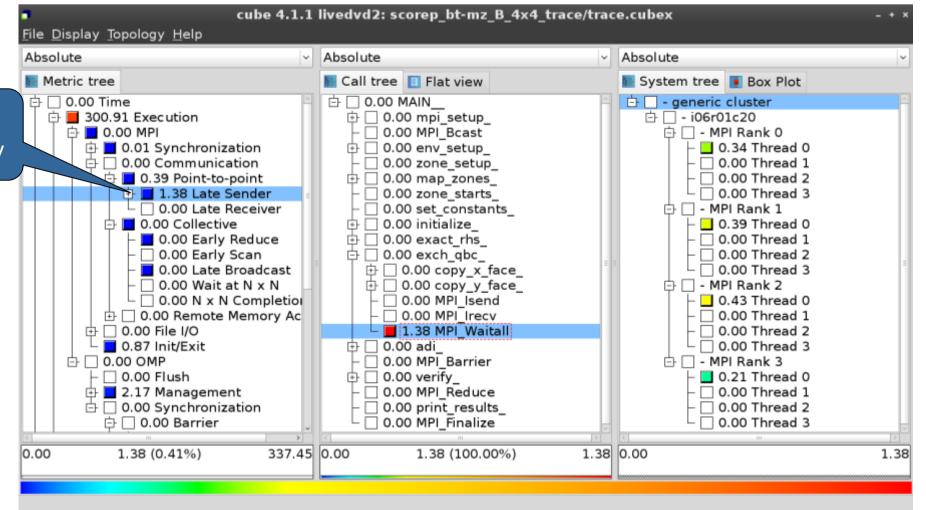
Run 'square -s' first and then copy 'trace.cubex' to local system (laptop) using 'scp' to improve responsiveness of GUI



#### Post-processed trace analysis report



Additional trace-based metrics in metric hierarchy

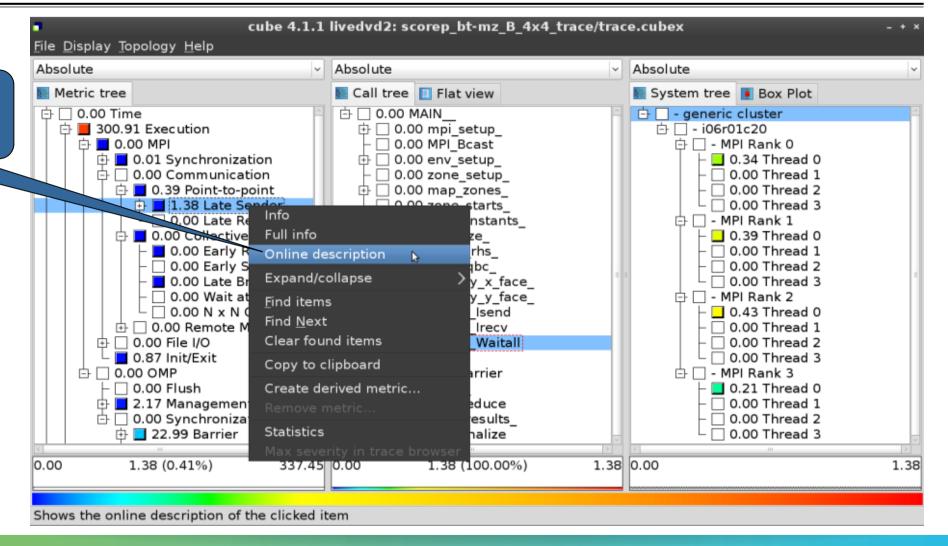




#### **Online metric description**

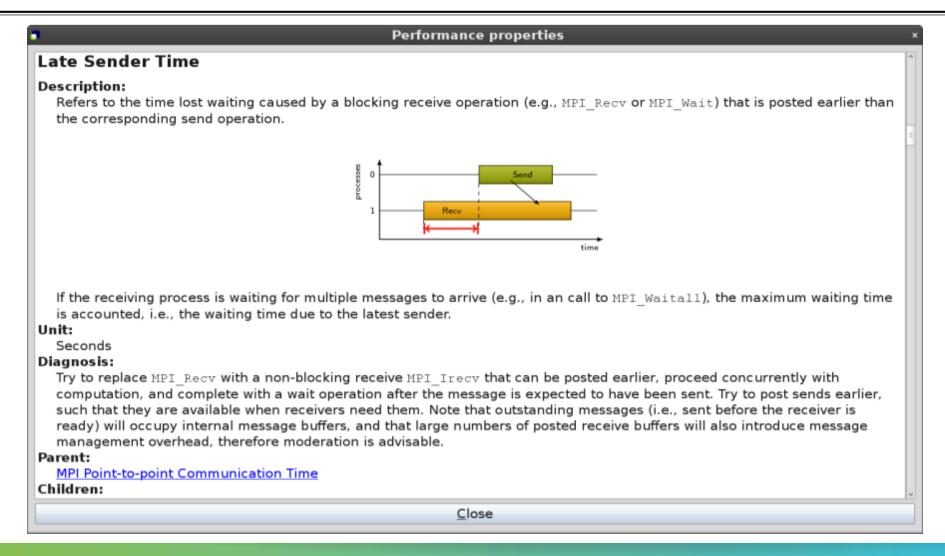


Access online metric description via context menu



#### **Online metric description**



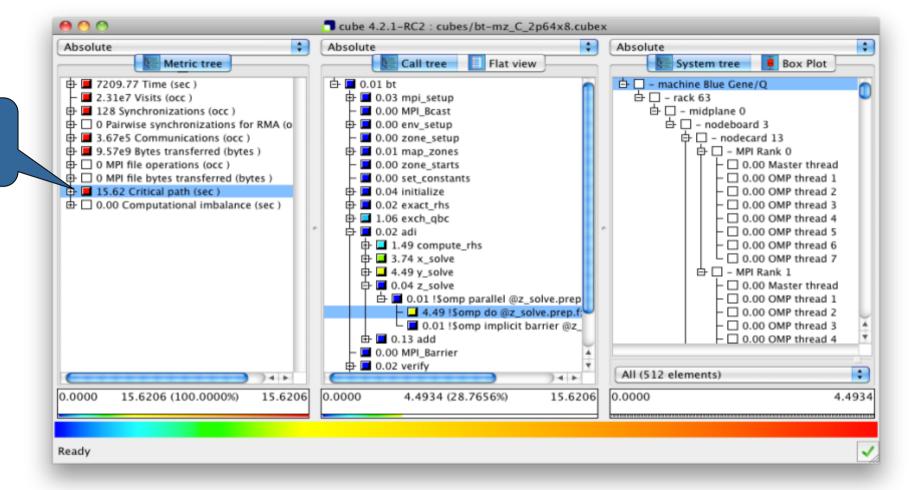




#### **Critical-path analysis**



Critical-path profile shows wall-clock time impact

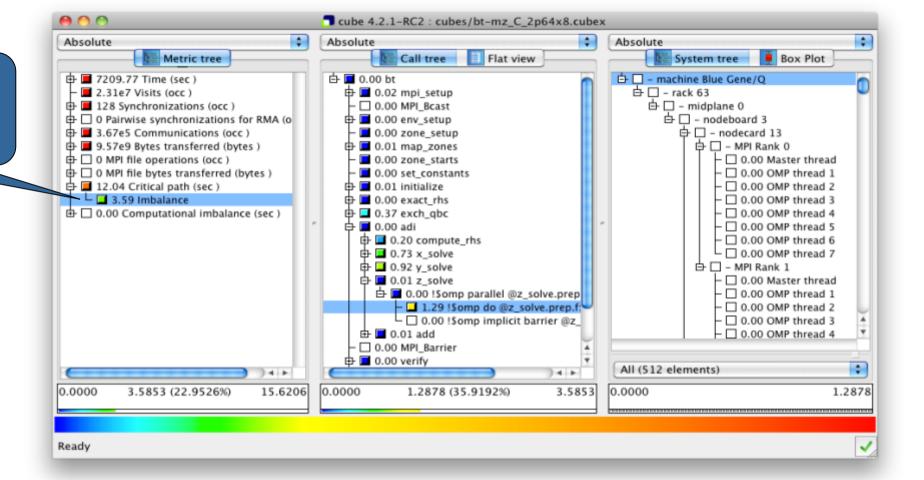


# VI-HPS

#### **Critical-path analysis**



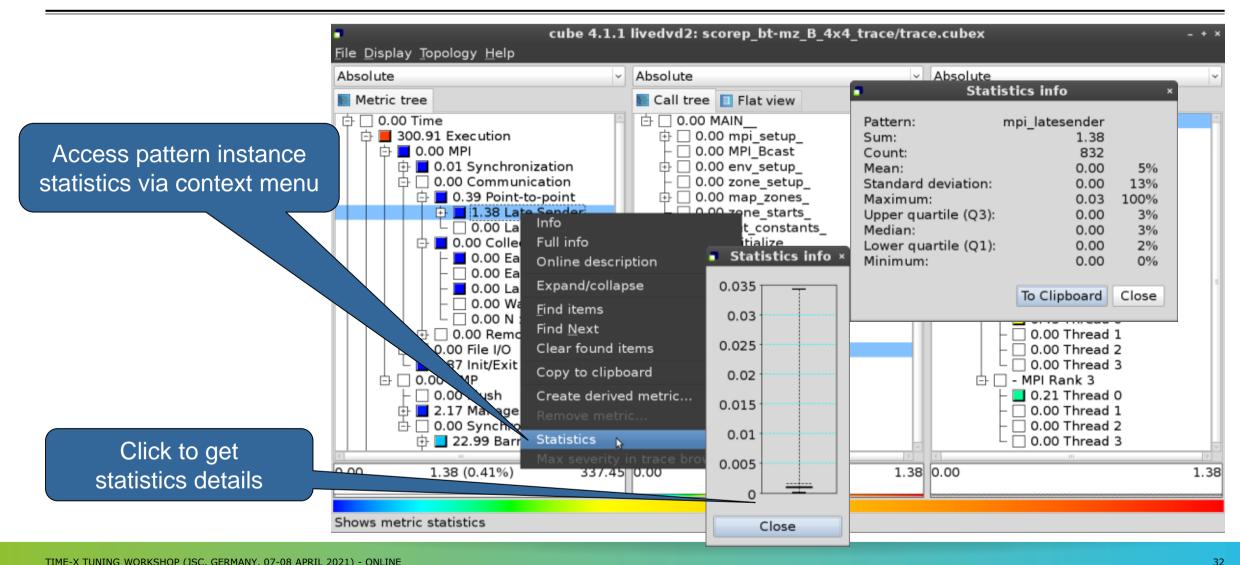
Critical-path imbalance highlights inefficient parallelism



# VI-HPS

#### Pattern instance statistics



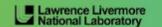


# Demo: **TeaLeaf case study**







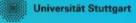










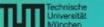


















#### Case study: TeaLeaf

- 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



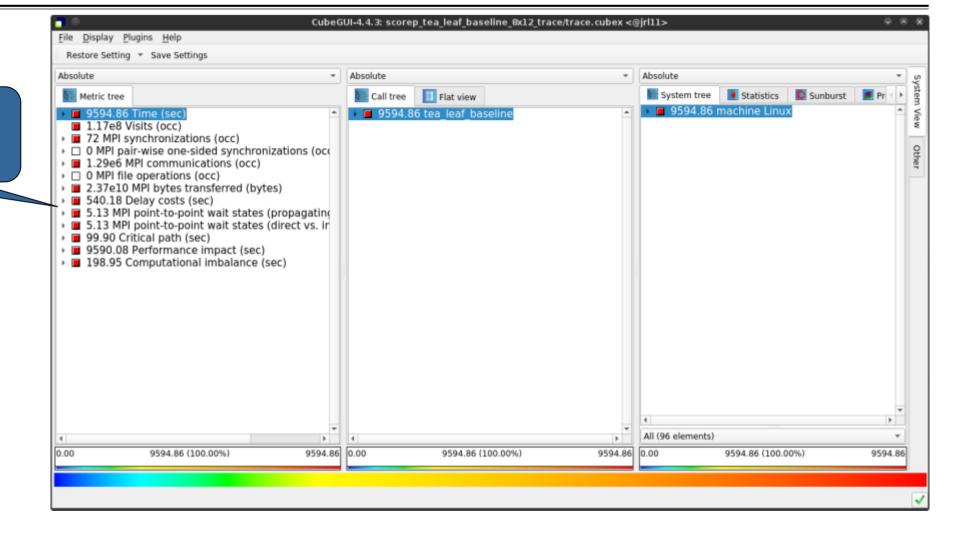
- Part of the Mantevo 3.0 suite
- Available on GitHub: https://uk-mac.github.io/TeaLeaf/
- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, Score-P 5.0, and Scalasca 2.5
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each
    - Distributed across 4 compute nodes (2 ranks per node)
    - Test problem "5": 4000 × 4000 cells, CG solver

## VI-HPS

## Scalasca analysis report exploration (opening view)



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

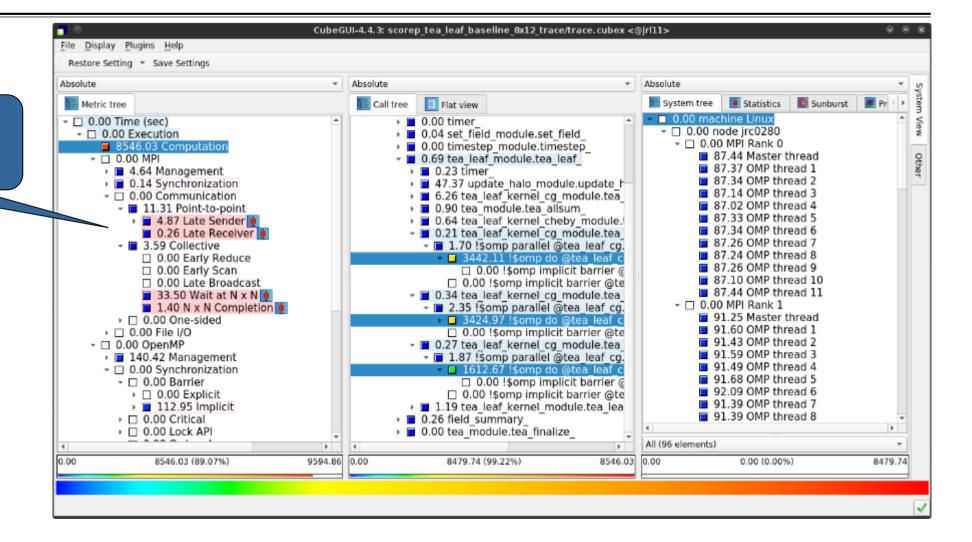




#### Scalasca wait-state metrics



...plus additional waitstate metrics as part of the "Time" hierarchy

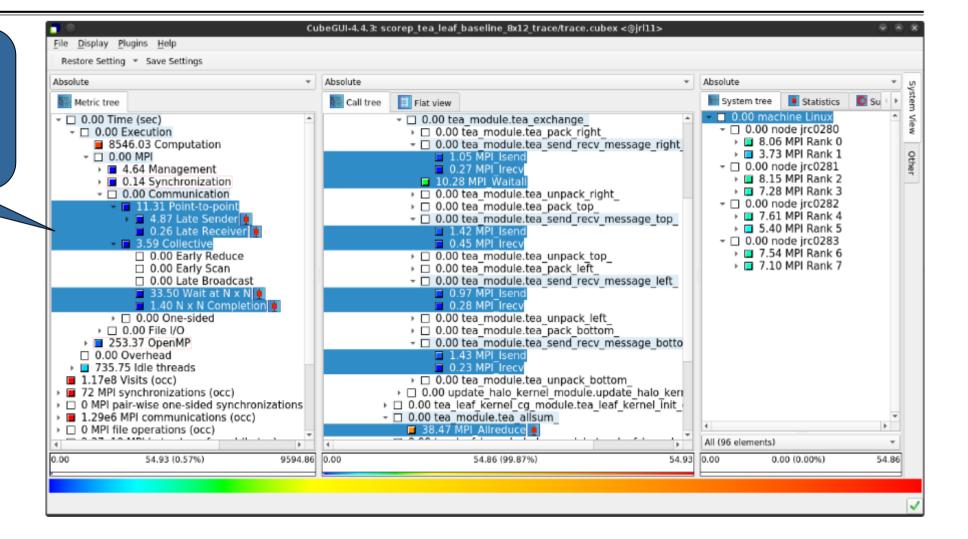




#### TeaLeaf Scalasca report analysis (I)



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

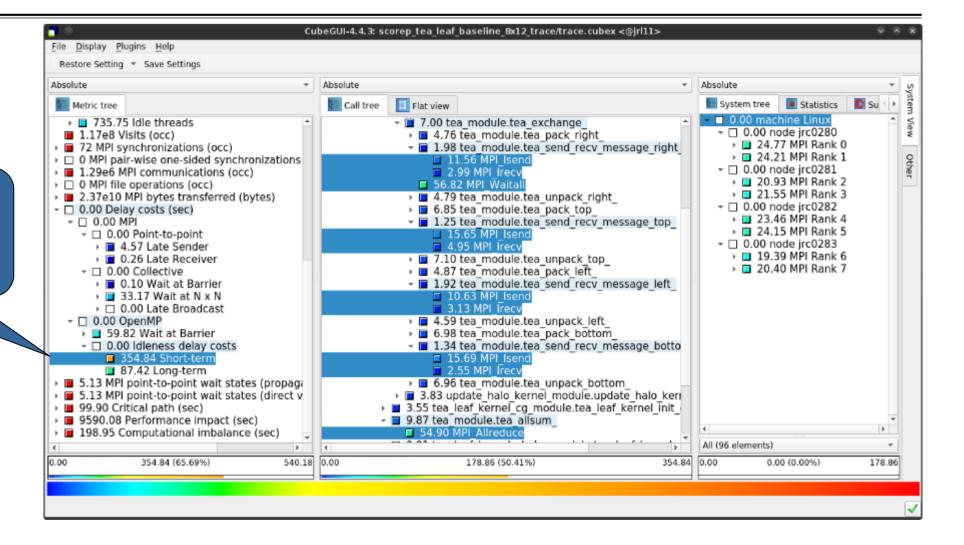


# VI-HPS

## TeaLeaf Scalasca report analysis (II)



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

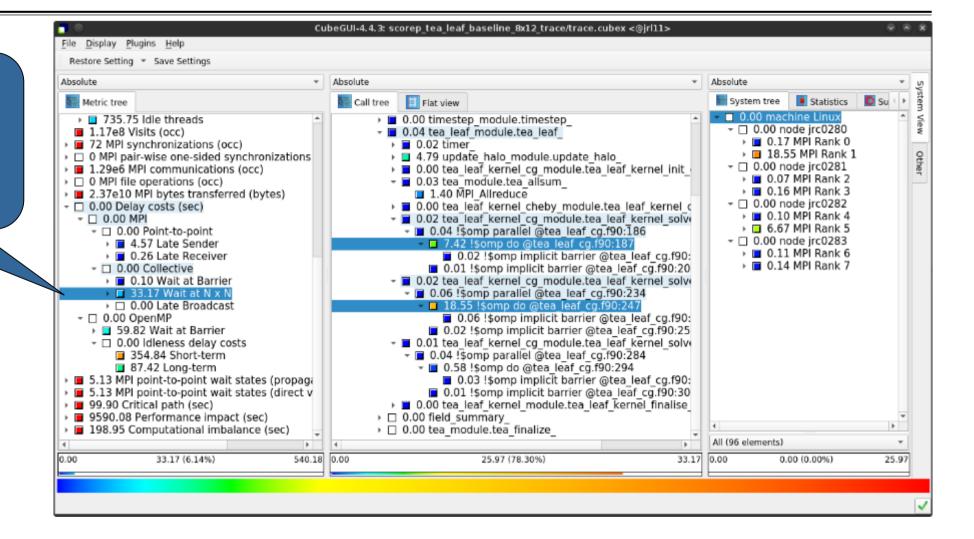


## VI-HPS

#### TeaLeaf Scalasca report analysis (III)



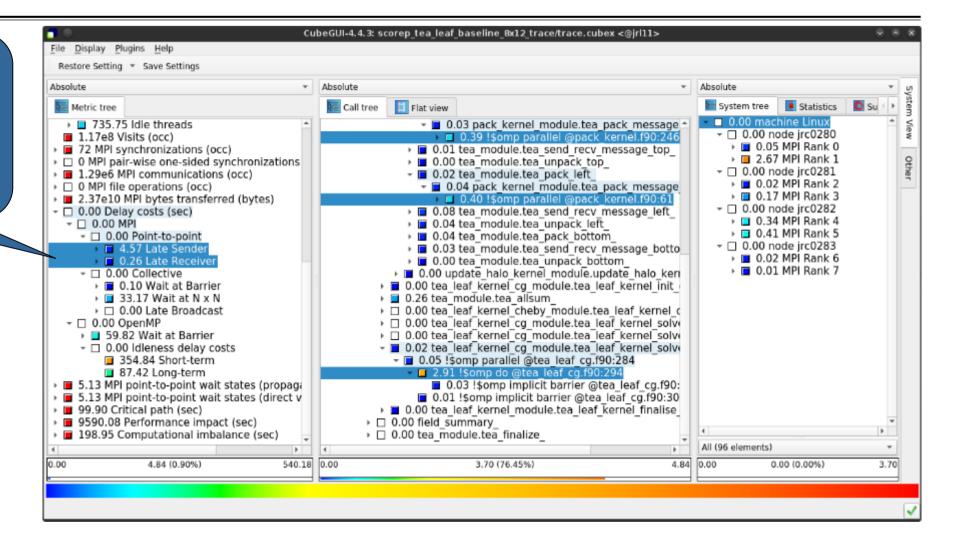
The "Wait at NxN" collective wait states are mostly caused by the first 2 OpenMP do loops of the solver (on ranks 5 & 1, resp.)...



# TeaLeaf Scalasca report analysis (IV)



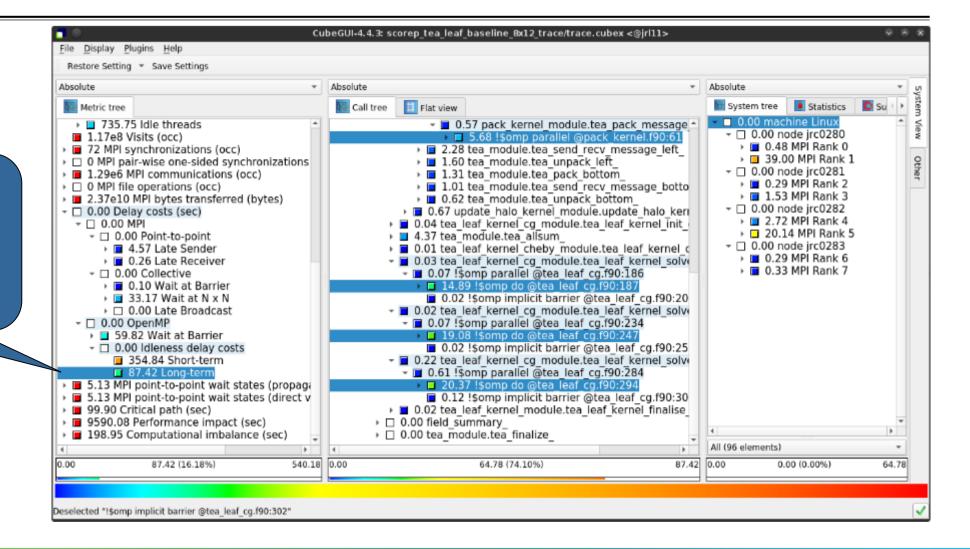
...while the MPI pointto-point wait states are caused by the 3<sup>rd</sup> solver do loop (on rank 1) and two loops in the halo exchange



# TeaLeaf Scalasca report analysis (V)



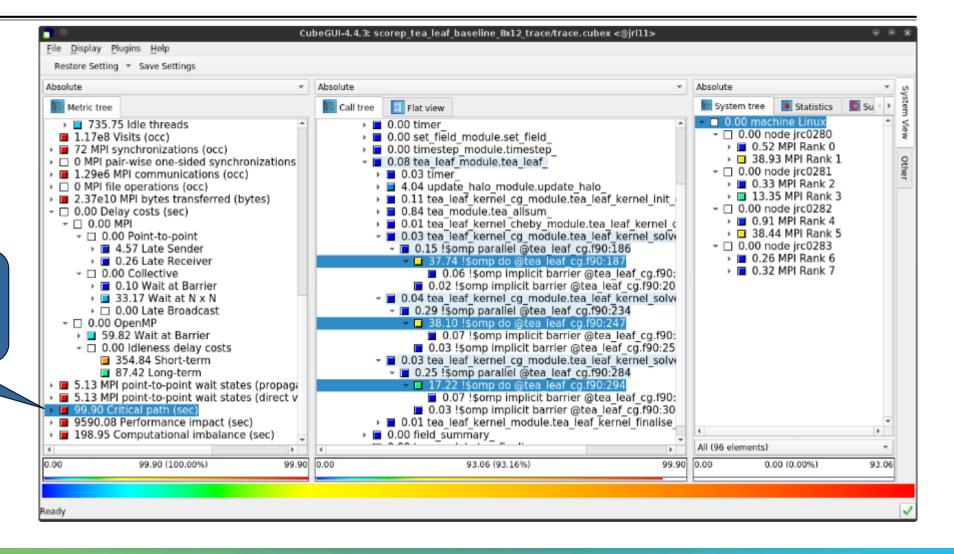
Various OpenMP do loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation



# TeaLeaf Scalasca report analysis (VI)



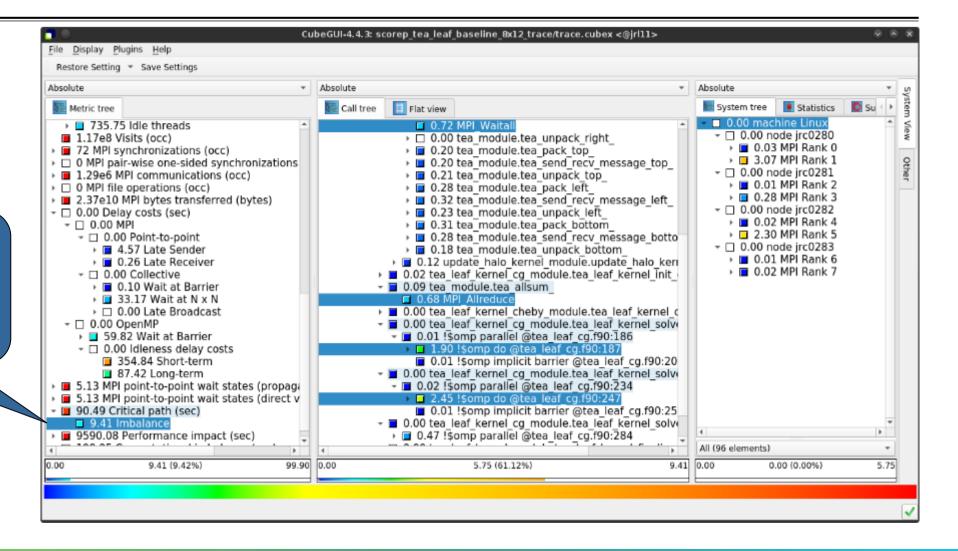
The Critical Path also highlights the three solver loops...



# TeaLeaf Scalasca report analysis (VII)



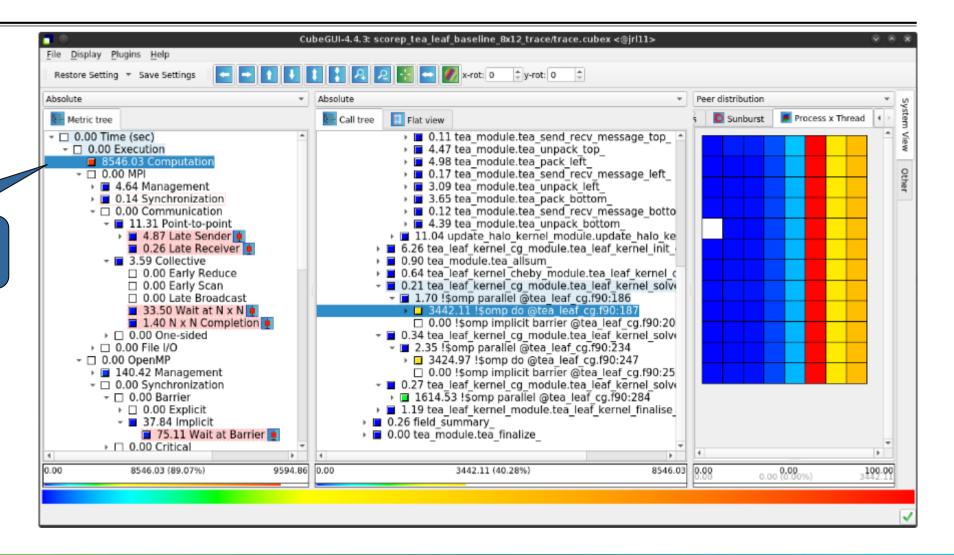
...with imbalance (time on critical path above average) mostly in the first two loops and MPI communication



# TeaLeaf Scalasca report analysis (VIII)



Computation time of 1st...

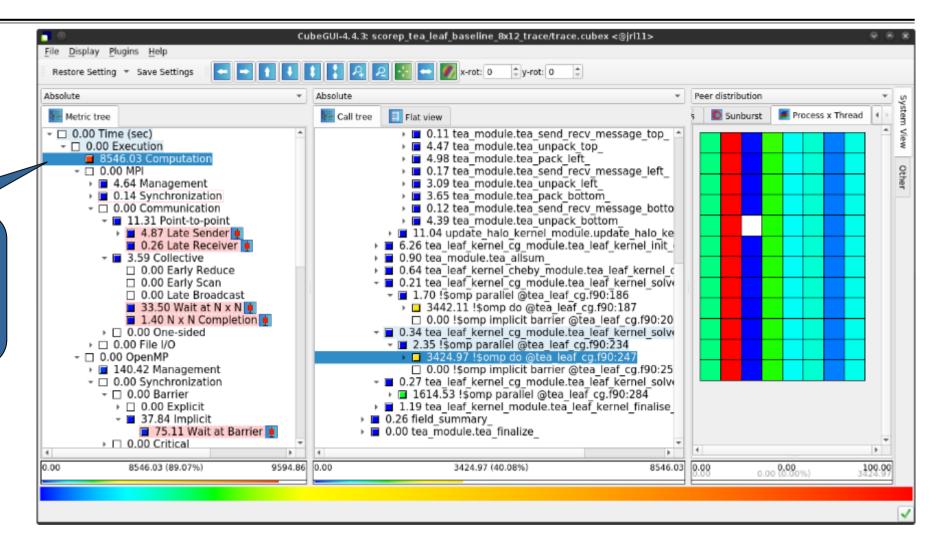




# TeaLeaf Scalasca report analysis (IX)



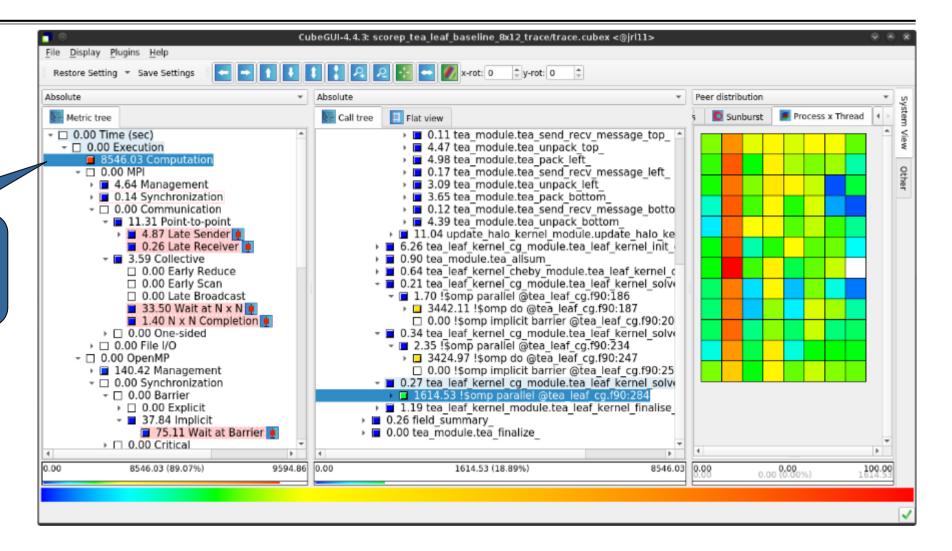
...and 2<sup>nd</sup> do loop mostly balanced within each rank, but vary considerably across ranks...



# 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 (quided) 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

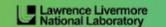


# **Exercises** (if you don't have your own code)



































#### Warm-up

- Build the BT-MZ example code for class (i.e., problem size) "D"
  - Perform a baseline measurement w/o instrumentation (should run in ~190s)
  - Re-build the executable with Score-P instrumentation
- Repeat the hands-on exercise with the new executable
  - Perform a summary measurement
  - Score the summary measurement result
  - Adjust the measurement configuration appropriately
  - Perform a trace measurement and analysis



# **Trace analysis report examination**

- What is the poportion of computation time vs. parallelization overheads?
- Which code regions are mostly responsible for the overall execution time?
- Are there any load balancing issues?
- If so, in which routines?
- What are the most significant wait states/parallelization overheads?
- What are their root causes?



# **Optimization**

- What are possible optimizations?
  - Hint: Take a look at the TeaLeaf case study
- Modify the source code to apply those and re-do the measurement
  - Don't worry it's straightforward even if you don't know the code ;-)
  - Remember: One step at a time!
- How did the performance change?
  - The cube\_diff tool or the "Cube Diff" plugin of the GUI (see the File → Context-free plugin menu) may come in handy here