PERFORMANCE ANALYSIS IN A NUTSHELL

HANDS ON WITH SCORE-P, SCALASCA, AND VAMPIR

OCTOBER 13, 2022 I MICHAEL KNOBLOCH

TUTORIAL EXERCISE OBJECTIVES

- Familiarise with usage of VI-HPS tools
  - complementary tools' capabilities & interoperability
- Prepare to apply tools productively to *your* applications(s)
- Exercise is based on a small portable benchmark code
  - unlikely to have significant optimisation opportunities

- Optional (recommended) exercise extensions
  - analyse performance of alternative configurations
  - investigate effectiveness of system-specific compiler/MPI optimisations and/or placement/binding/affinity capabilities
  - investigate scalability and analyse scalability limiters
  - compare performance on different HPC platforms
  - …

BT-MZ @ LEVANTE

REFERENCE RUN

JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

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

COMPILER AND MPI MODULES (LEVANTE)

- Select modules for the Intel + OpenMPI tool chain

```
% module load intel-oneapi-compilers/2022.0.1-gcc-11.2.0
% module load openmpi/4.1.2-intel-2021.5.0
```

Should already been done on login

- Copy tutorial sources to your HOME directory

```
% cd $HOME
% tar zxvf /home/k/k203166/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

NPB-MZ-MPI SUITE

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from:

    http://www.nas.nasa.gov/Software/NPB

  - 3 benchmarks in Fortran77
  - Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% ls
bin/    common/  jobscript/  Makefile  README.install   SP-MZ/
BT-MZ/  config/  LU-MZ/      README    README.tutorial  sys/
```

- Subdirectories contain source code for each benchmark
  - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it is ready to “make” one or more of the benchmarks
  - but config/make.def may first need to be adjusted to specify appropriate compiler flags

NPB-MZ-MPI / BT: CONFIG/MAKE.DEF

```
#                SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Configured for generic MPI with GCC compiler
#---------------------------------------------------------------------------
#OPENMP = -fopenmp          # GCC compiler
OPENMP = -fopenmp          # Intel compiler

...
#---------------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------------
MPIF77 = mpiifort

# Alternative variants to perform instrumentation
...

#MPIF77 = scorep --user  mpiifort
...
```

Uncomment COMPILER flags according to current environment

Default (no instrumentation)

Hint: uncomment a compiler wrapper to do instrumentation

BUILDING AN NPB-MZ-MPI BENCHMARK

```
% make
  ============================================
  =      NAS PARALLEL BENCHMARKS 3.3         =
  =      MPI+OpenMP Multi-Zone Versions      =
  =      F77                                 =
  ============================================

  To make a NAS multi-zone benchmark type

         make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

  where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
        <class>          is "S", "W", "A" through "F"
        <nprocs>         is number of processes

  [...]

 ************************************************************
 * Custom build configuration is specified in config/make.def  *
 * Suggested tutorial exercise configuration for Levante:      *
 *       make bt-mz CLASS=C NPROCS=28                          *
 ************************************************************
```

- Type “make” for instructions

BUILDING AN NPB-MZ-MPI BENCHMARK

```
% make bt-mz CLASS=C NPROCS=28
make[1]: Entering directory `BT-MZ'
make[2]: Entering directory `sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory `sys'
../sys/setparams bt-mz 28 C
make[2]: Entering directory `../BT-MZ'
mpif77 -g -c  -O3 -fopenmp        bt.f
[…]
mpif77 -g -c  -O3 -fopenmp        mpi_setup.f
cd ../common; mpif77 -g -c  -O3 -fopenmp    print_results.f
cd ../common; mpif77 -g -c  -O3 -fopenmp    timers.f
mpif77 -g -O3 -fopenmp     -o ../bin/bt-mz_B.8 bt.o
 initialize.o exact_solution.o exact_rhs.o set_constants.o adi.o
 rhs.o zone_setup.o x_solve.o y_solve.o  exch_qbc.o solve_subs.o
 z_solve.o add.o error.o verify.o mpi_setup.o ../common/print_results.o
 ../common/timers.o
make[2]: Leaving directory `BT-MZ'
Built executable ../bin/bt-mz_C.28
make[1]: Leaving directory `BT-MZ'
```

- Specify the benchmark configuration
  - benchmark name: **bt-mz**, lu-mz, sp-mz
  - the benchmark class (S, W, A, B, C, D, E): CLASS=**C**
  - the number of MPI processes: NPROCS=**28**

Shortcut: `% make suite`

NPB-MZ-MPI / BT (BLOCK TRIDIAGONAL SOLVER)

- What does it do?
  - Solves a discretized version of the unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules

- Uses MPI & OpenMP in combination
  - 28 processes each with 4 threads should be reasonable for 2 compute nodes of Levante
  - bt-mz_C.28 should run in about 10 seconds with the Intel toolchain

NPB-MZ-MPI / BT REFERENCE EXECUTION

```
% cd bin
% cp ../jobscript/levante/reference.sbatch .
% less reference.sbatch
% sbatch --account=kg0166 reference.sbatch

% cat bt-mz.<job_id>.out
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:  8 x  8
 Iterations:  200   dt:   0.000300
 Number of active processes:    28
 Use the default load factors with threads
 Total number of threads:    112   (  4.0 threads/process)

 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 17.33
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application

Hint: save the benchmark output (or note the run time) to be able to refer to it later

BT-MZ @ LEVANTE

INITIAL SCORE-P MEASUREMENT

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

LOCAL INSTALLATION (LEVANTE)

- Latest/recent versions of Score-P and tools available via modules
  - Score-P installation is toolchain specific

```
% ml intel-oneapi-compilers/2022.0.1-gcc-11.2.0 openmpi/4.1.2-intel-2021.5.0
% ml scorep/7.0-intel-2021.5.0 cube/4.6-gcc-11.2.0 scalasca/2.6-gcc-11.2.0
% spack load vampir@10.0.2
% source /home/k/k203166/scorep.env
```

  - Check `module avail scorep` for alternate Score-P modules available

- Copy tutorial sources to your $HOME directory (should be done already)

```
% cd $HOME
% tar zxvf /home/k/k203166/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

NPB-MZ-MPI / BT INSTRUMENTATION

```
#---------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------
#MPIF77 = mpif77

# Alternative variants to perform instrumentation
...
MPIF77 = scorep --user mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK   = $(MPIF77)
...
```

- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: MPIF77

Uncomment the Score-P compiler wrapper specification

NPB-MZ-MPI / BT INSTRUMENTED BUILD

```
% make clean

% make bt-mz CLASS=C NPROCS=28
cd BT-MZ; make CLASS=B NPROCS=28 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 28 C
scorep --user mpif77 -g -c  -O3 -qopenmp bt.f
 [...]
cd ../common;  scorep --user mpif77 -g -c  -O3 -qopenmp timers.f
 [...]
scorep --user mpif77 -g -O3 -qopenmp -o ../bin.scorep/bt-mz_B.28 \
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \
solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \
../common/print_results.o ../common/timers.o
Built executable ../bin.scorep/bt-mz_C.28
make: Leaving directory 'BT-MZ'
```

- Return to root directory and clean-up
- Re-build executable using Score-P compiler wrapper

MEASUREMENT CONFIGURATION: SCOREP-INFO

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
 [...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
 [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
 [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
 [...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
 [...]
SCOREP_METRIC_PAPI
  Description: PAPI metric names to measure
 [...]
SCOREP_METRIC_RUSAGE
  Description: Resource usage metric names to measure
 [... More configuration variables ...]
```

- Score-P measurements are configured via environmental variables

SUMMARY MEASUREMENT COLLECTION

```
% cd bin.scorep
% cp ../jobscript/levante/scorep.sbatch .
% cat scorep.sbatch
...
# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum
#export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,…
#export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L2_TCM
#export SCOREP_METRIC_RUSAGE=ru_stime
#export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss
#export SCOREP_TIMER=gettimeofday

# Run the application
mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS

% sbatch -account=kg0166 scorep.sbatch
```

- Change to the directory containing the new executable before running it with the desired configuration
- Check settings

Leave these lines commented out for the moment

- Submit job

SUMMARY MEASUREMENT COLLECTION

```
% less npb_btmz.o<job_id>

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

 Number of zones:  8 x  8
 Iterations: 200    dt:   0.000300
 Number of active processes:     28
 Use the default load factors with threads
 Total number of threads:    112   (  4.0 threads/process)

 Calculated speedup = 71.69

 Time step    1

 [... More application output ...]
```

- Check the output of the application run

BT-MZ SUMMARY ANALYSIS REPORT EXAMINATION

```
% ls
bt-mz_C.28 bt-mz.<job_id>.out scorep_bt-mz_sum/
% ls scorep_bt-mz_sum
MANIFEST.md profile.cubex  scorep.cfg
```

```
% cube scorep_bt-mz_sum/profile.cubex

      [CUBE GUI showing summary analysis report]
```

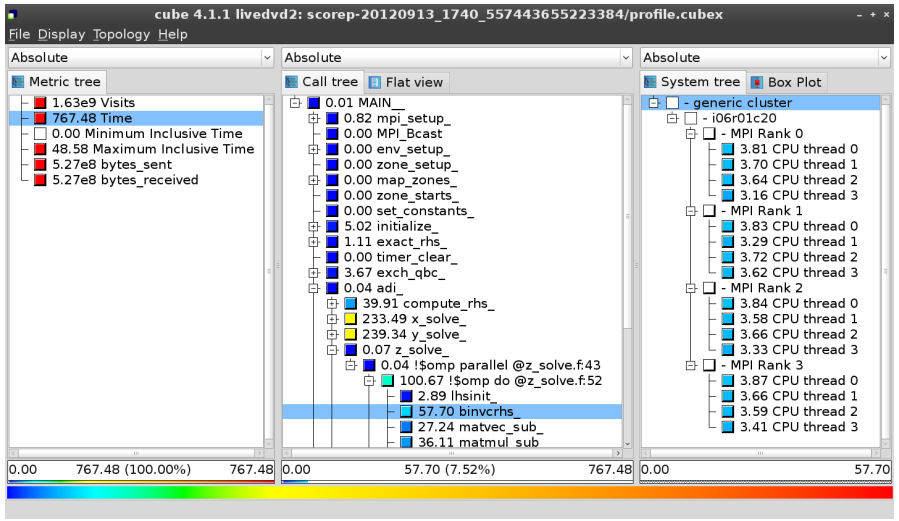
- Creates experiment directory including
  - A brief content overview (MANIFEST.md)
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)
- Interactive exploration with Cube

Hint:
Copy ‘profile.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI

Reference results available:
`/home/k/k203166/reference_results`

CUBE

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - Requires Qt4 ≥4.6 or Qt 5
- Originally developed as part of the Scalasca toolset
- Now available as a separate component
  - Can be installed independently of Score-P, e.g., on laptop or desktop
  - Latest release: Cube v4.6 (April 2021)



Note:
Binary packages provided for Windows & MacOS, from **www.scalasca.org** website in software/Cube-4x

CUBE GUI (LEVANTE)

mailto: **scalasca@fz-juelich.de**

- Run ***remote*** *(often convenient)*
  - start X server (e.g., Xming) locally
  - connect to Levante with X forwarding enabled

```
desk$ ssh -X levante
Welcome to levante…
Levante$ cd $PATH_TO_BT
levante$ module load cube
levante$ cube ./scorep_sum/profile.cubex
```

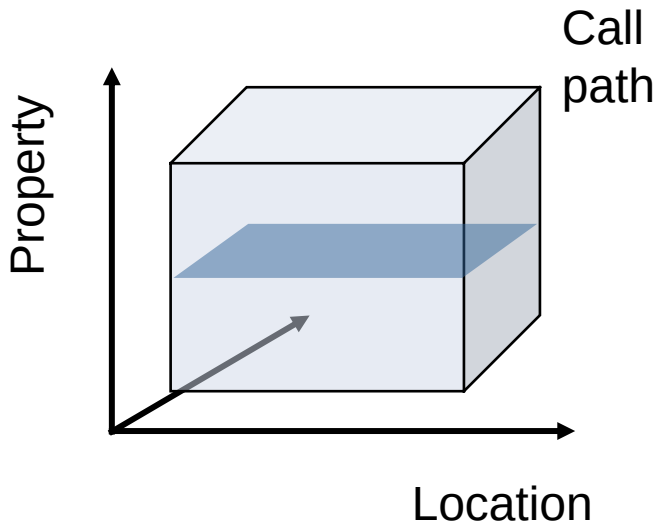
- Install & run ***local*** *(recommended)*
  - install Cube GUI locally on desktop
    - binary packages available for MacOS & Windows and externally provided by OpenHPC and various Linux distributions
    - source package available for Linux, requires Qt
      - configure/build/install manually or use your favourite framework (e.g. Spack or EasyBuild)
  - copy .cubex file (or entire scorep directory) to desktop from remote system
    OR locally mount remote filesystem
  - start cube locally

```
desk$ mkdir $HOME/mnt
desk$ sshfs [user@]remote.sys:[dir] $HOME/mnt
desk$ cd $HOME/mnt
desk$ cube ./scorep_sum/profile.cubex
```

https://www.**scalasca.org**/scalasca/software/cube-4.x/download.html

ANALYSIS PRESENTATION AND EXPLORATION

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)

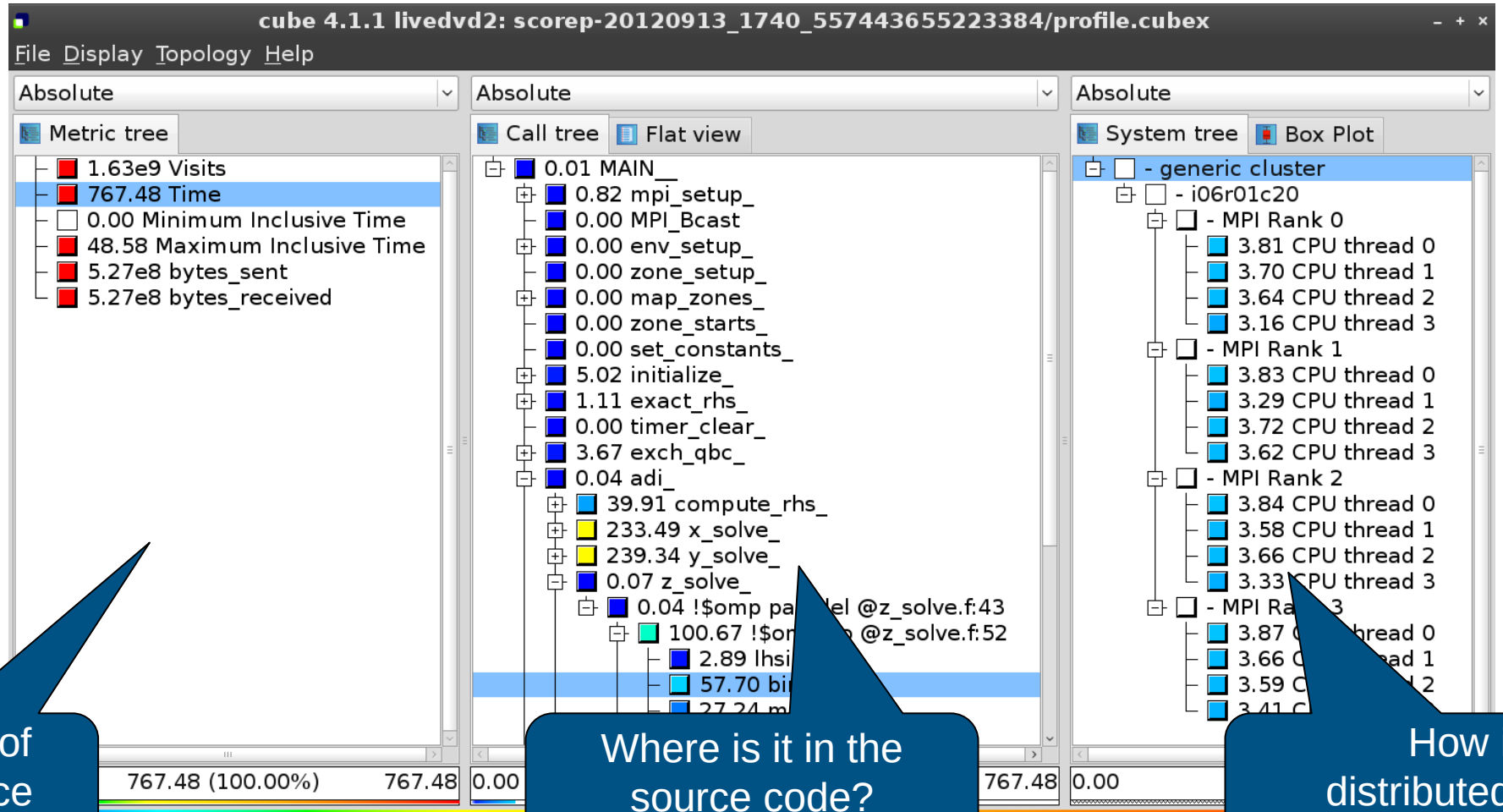


- Three coupled tree browsers

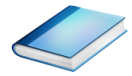
- Cube displays severities
  - As *value*: for precise comparison
  - As *colour*: for easy identification of hotspots
  - *Inclusive* value when closed & *exclusive* value when expanded
  - Customizable via display modes

ANALYSIS PRESENTATION

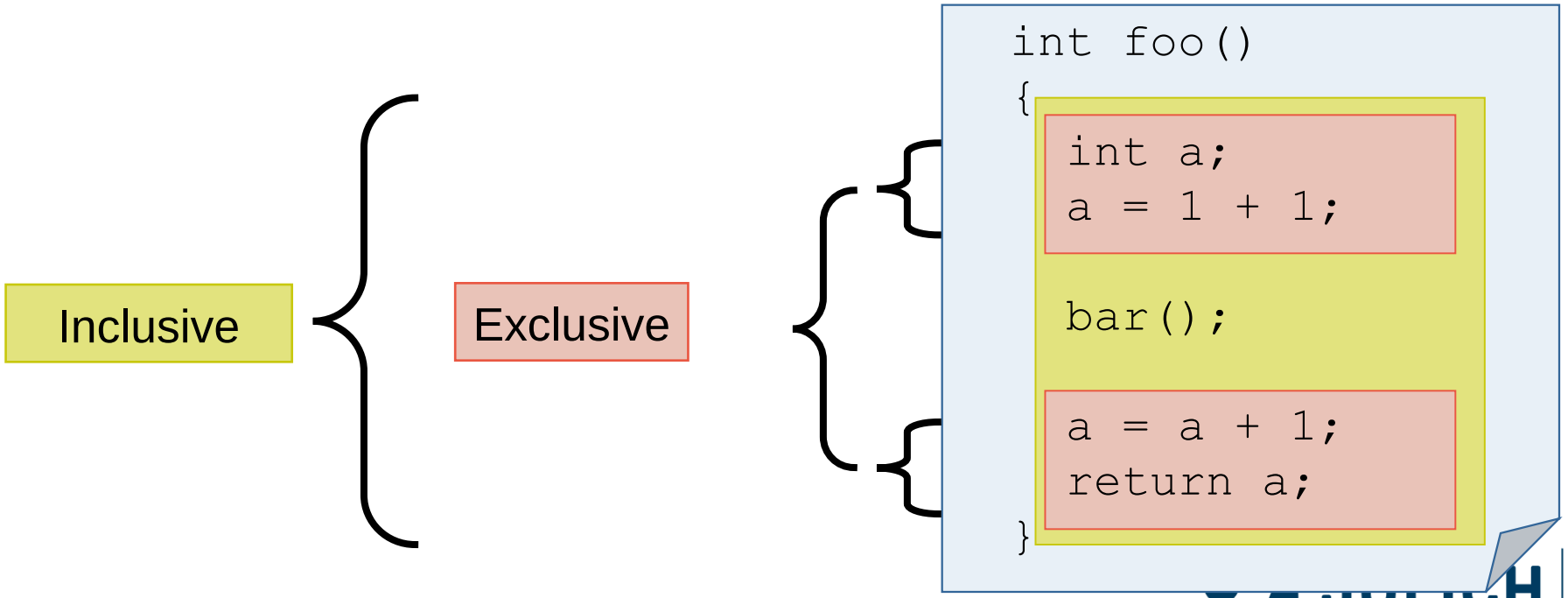
JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



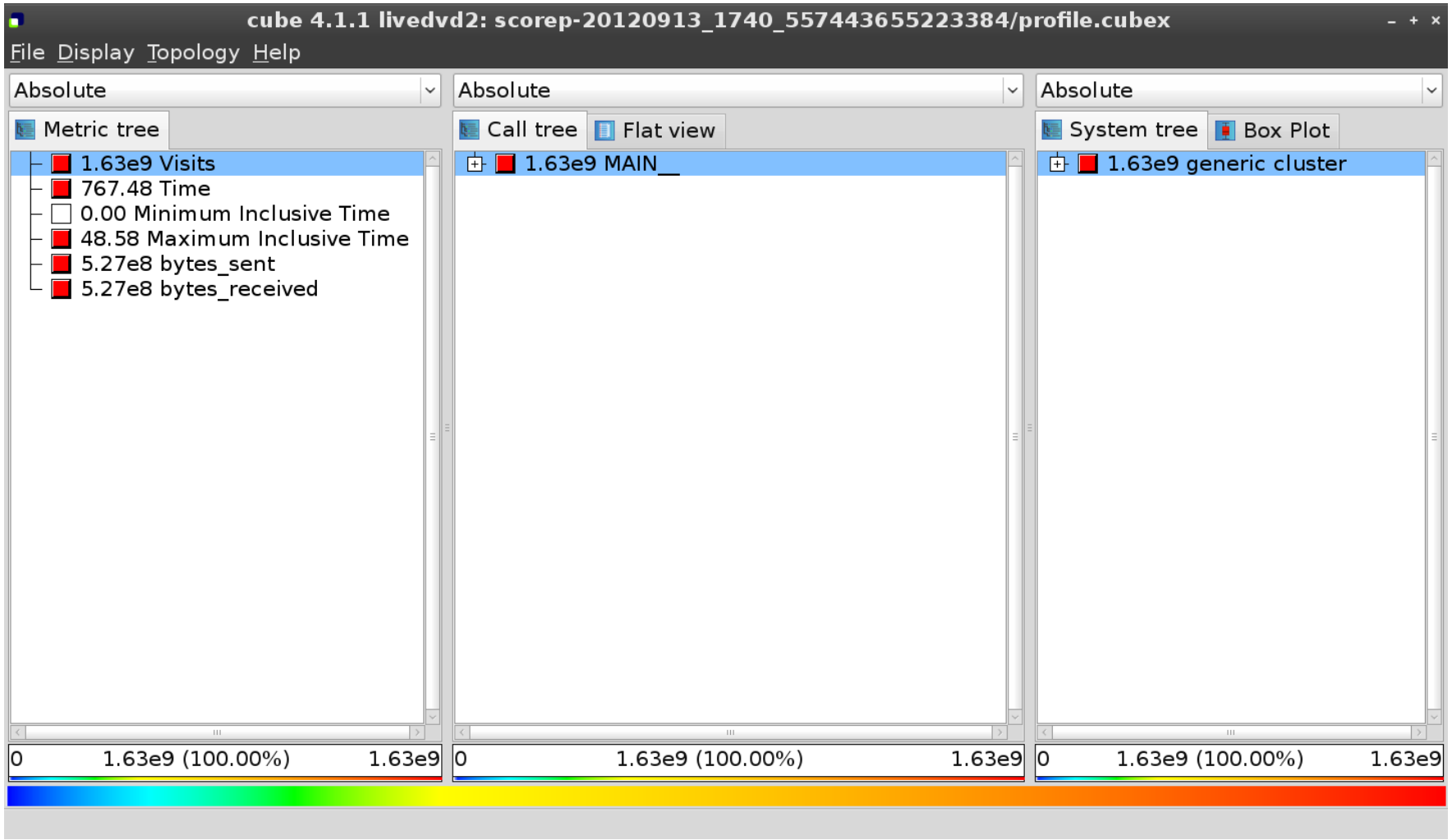
INCLUSIVE VS. EXCLUSIVE VALUES



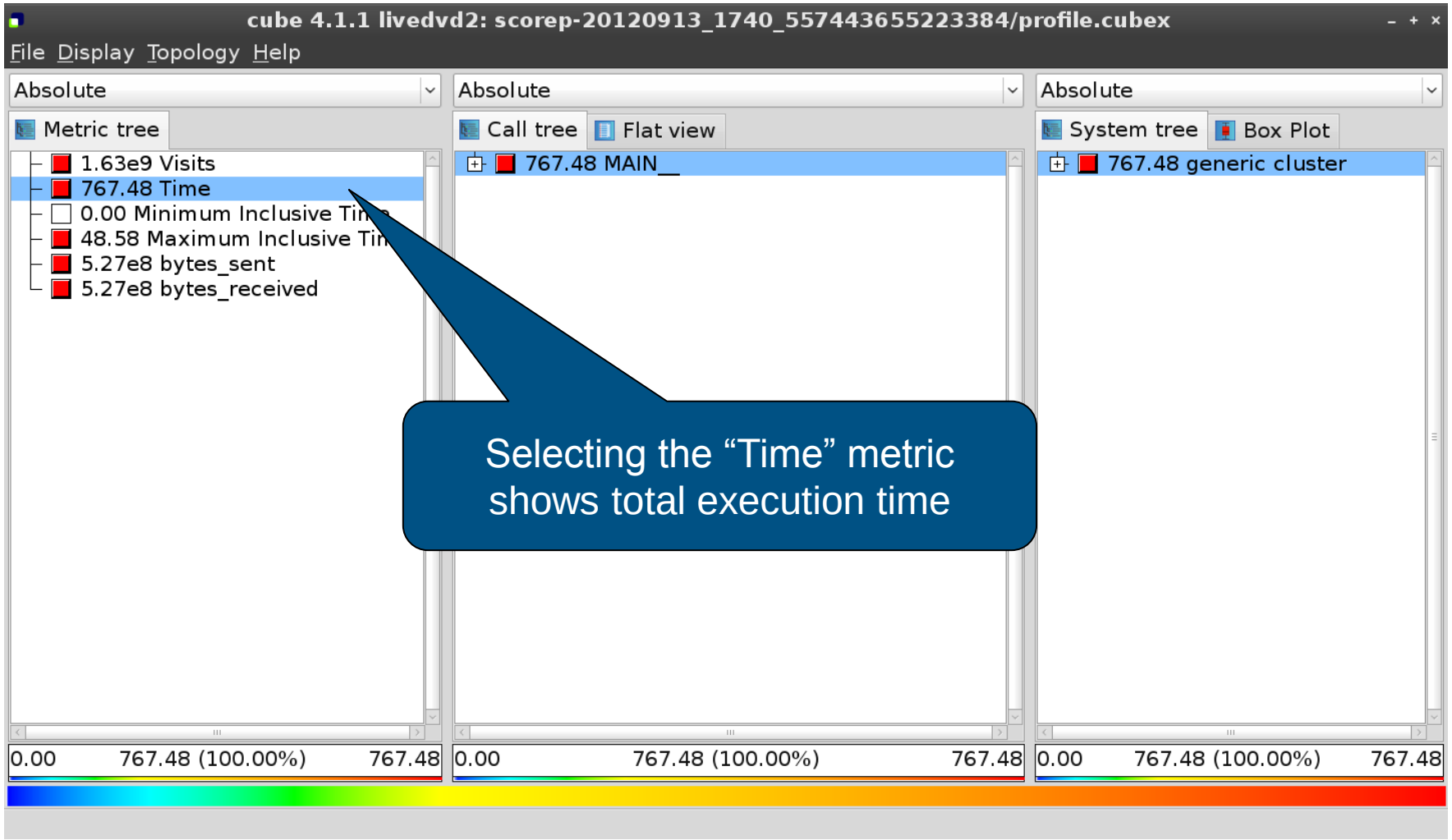
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



SCORE-P ANALYSIS REPORT EXPLORATION (OPENING VIEW)

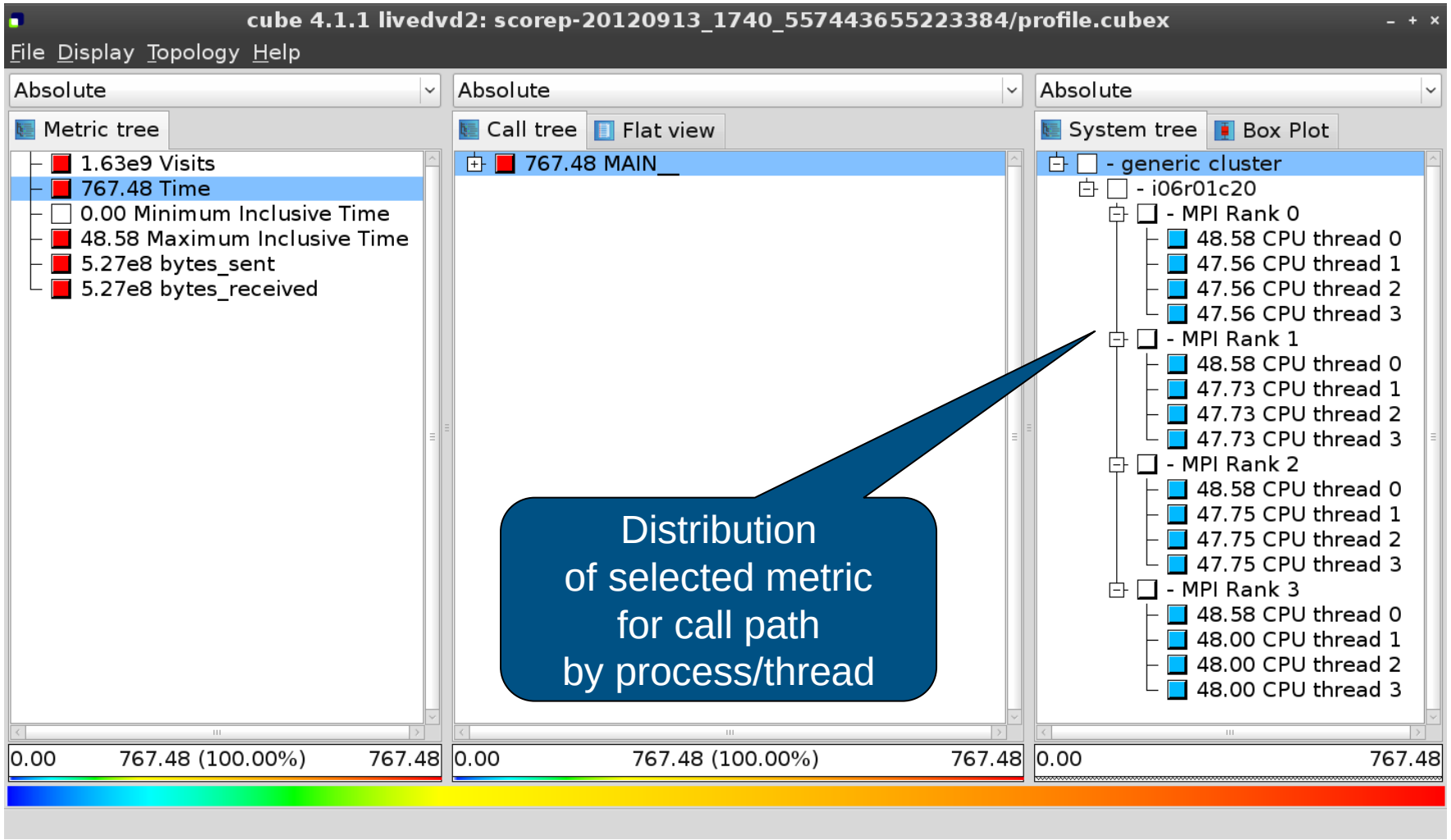


METRIC SELECTION

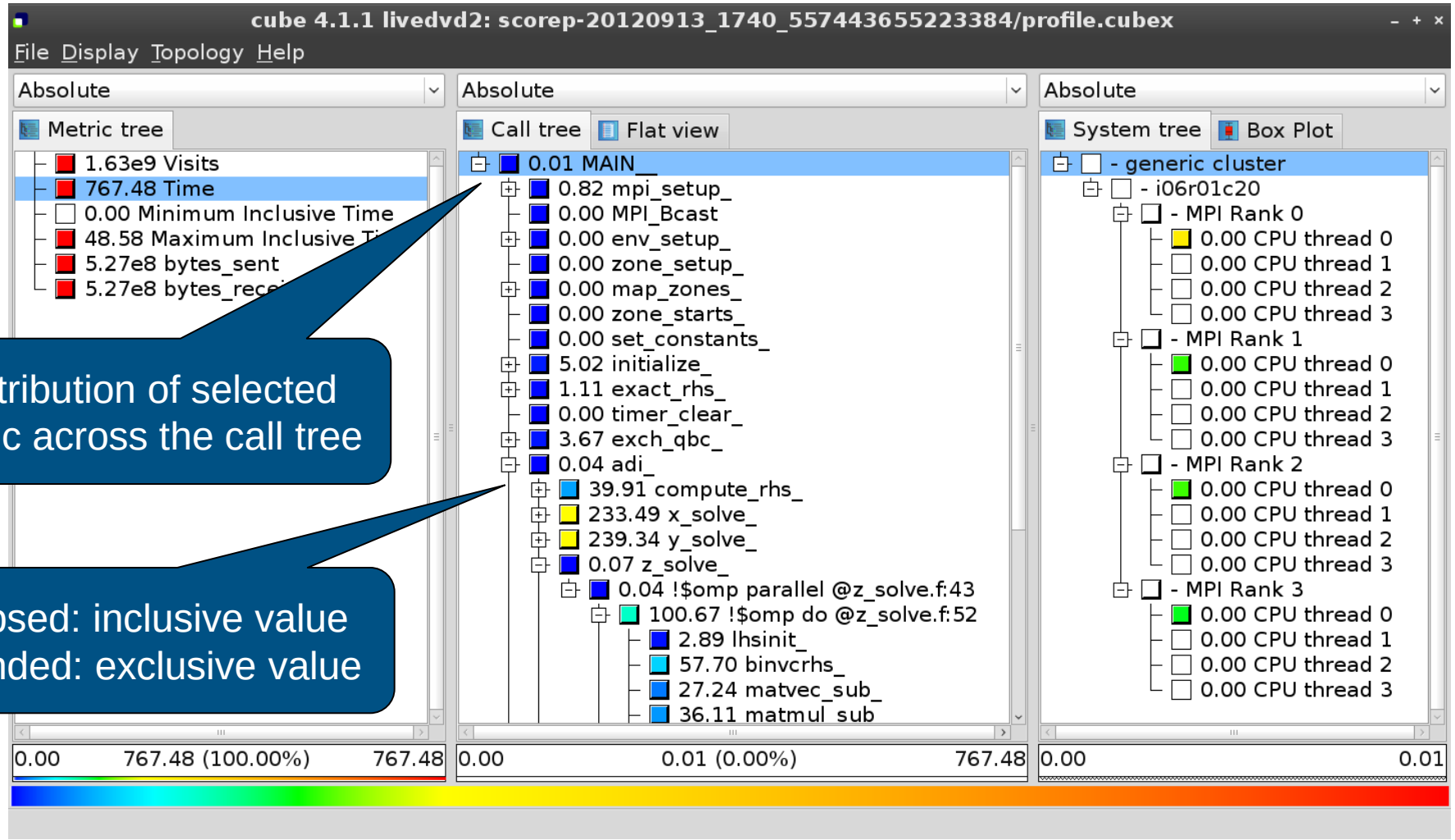


EXPANDING THE SYSTEM TREE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



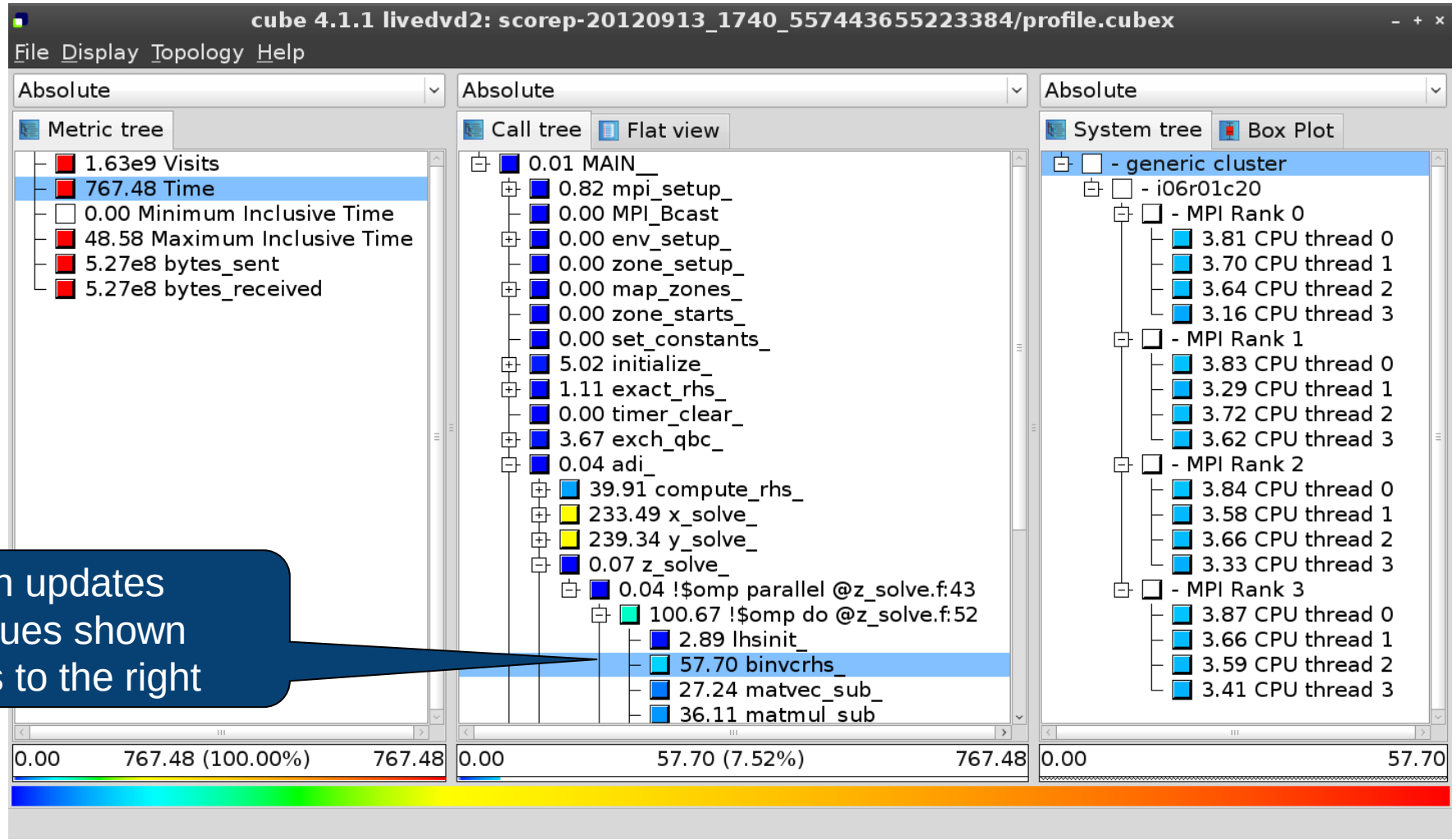
EXPANDING THE CALL TREE



Distribution of selected metric across the call tree

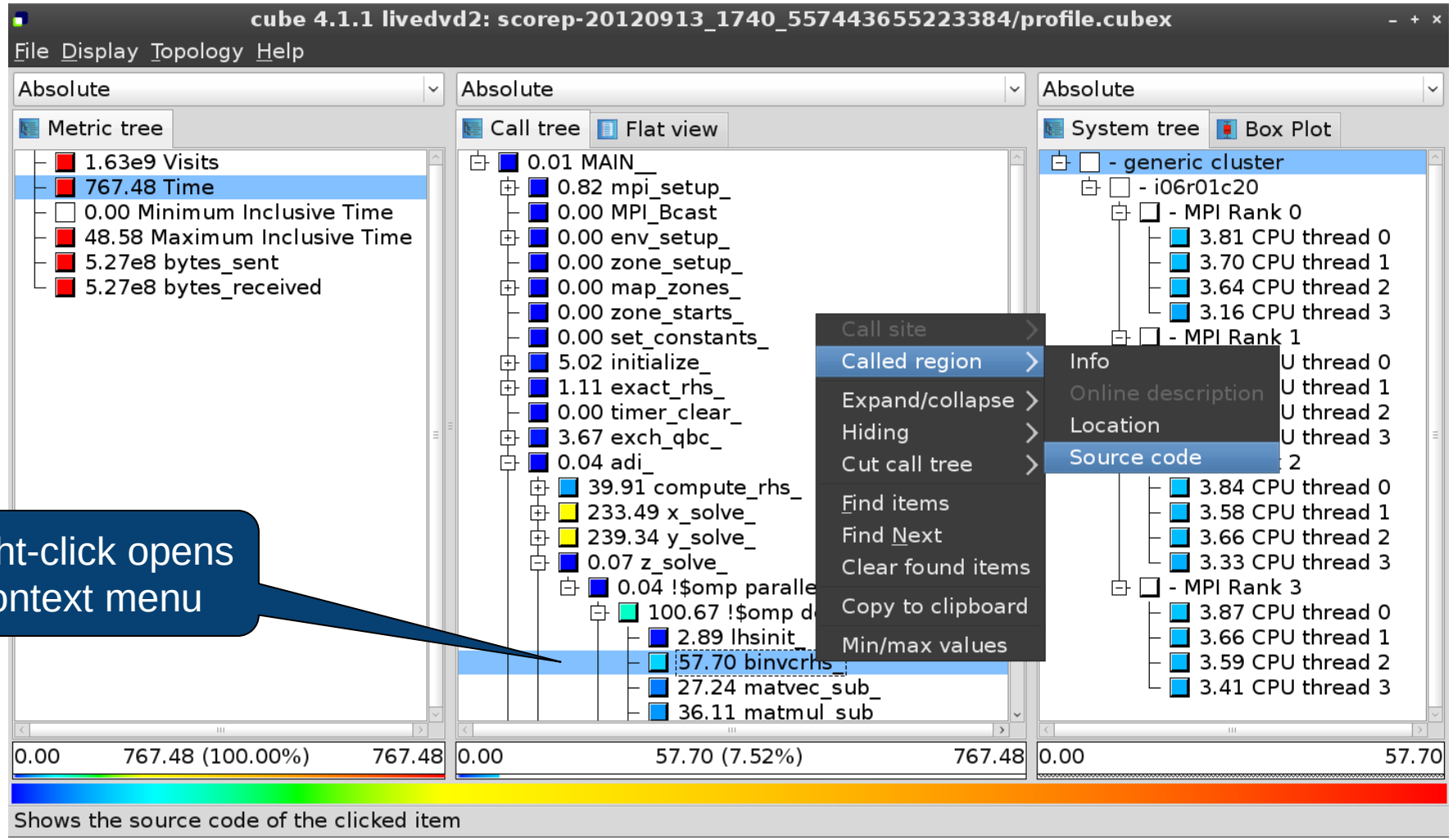
Collapsed: inclusive value
Expanded: exclusive value

SELECTING A CALL PATH



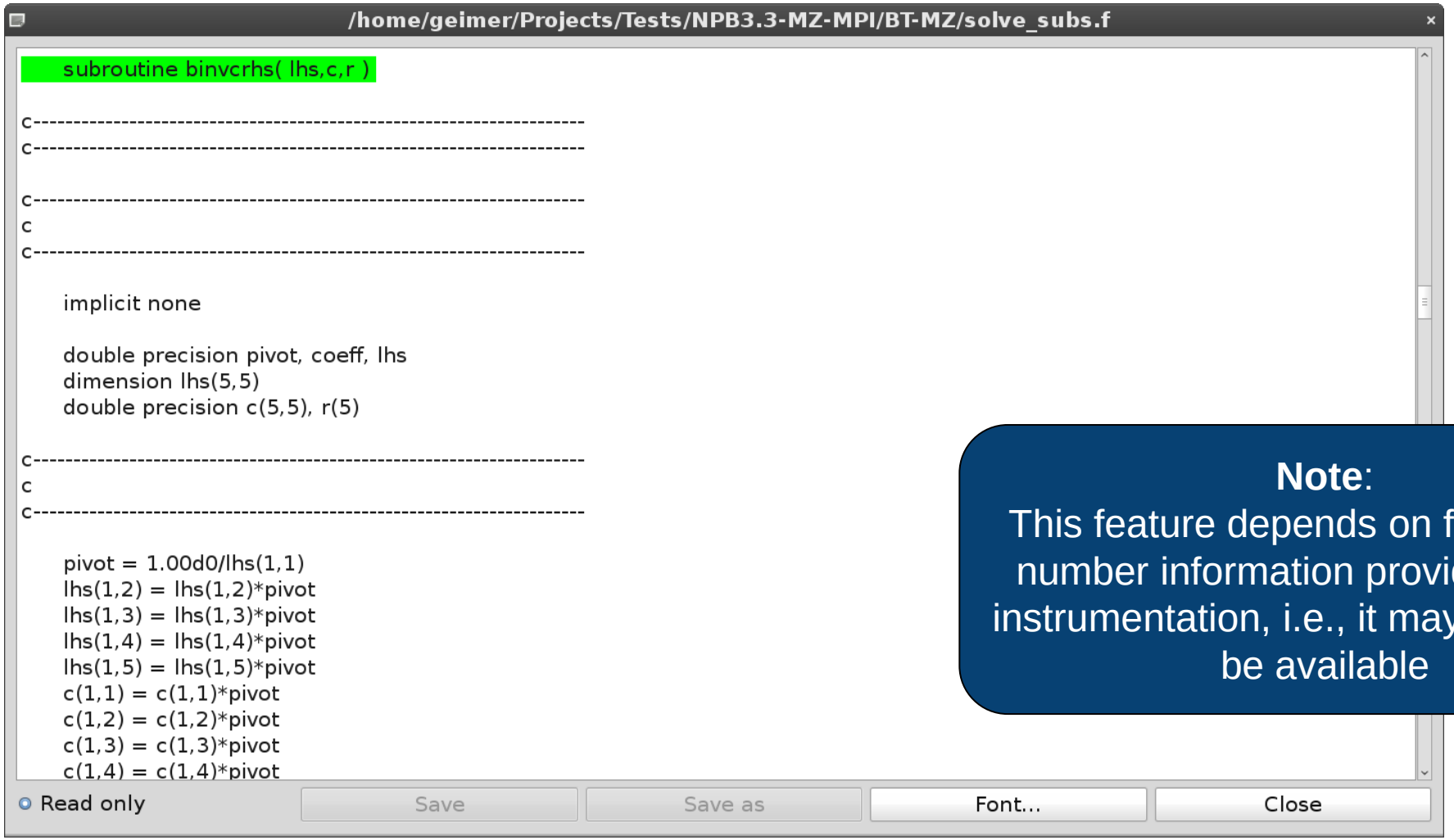
Selection updates metric values shown in columns to the right

SOURCE-CODE VIEW VIA CONTEXT MENU



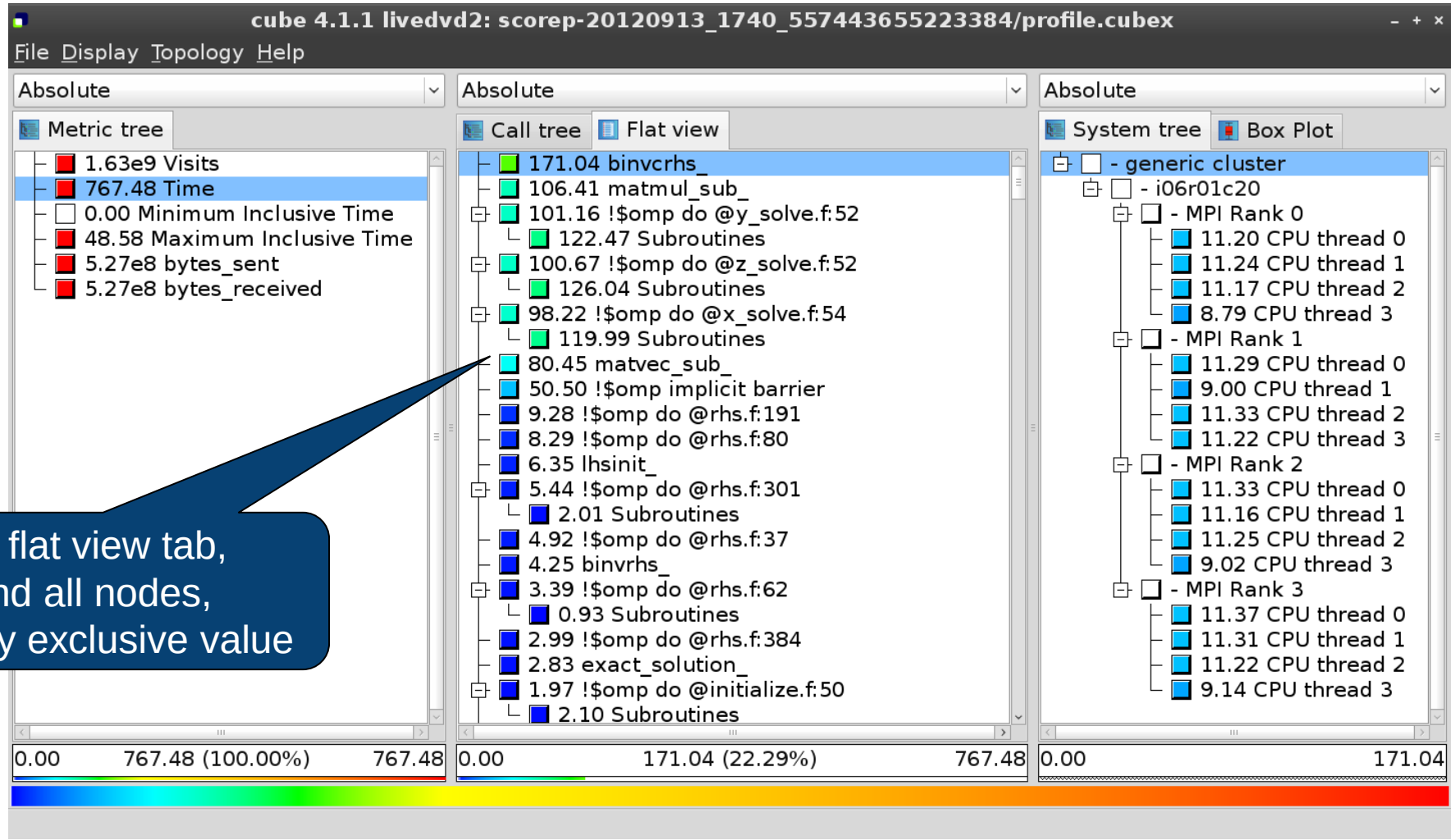
SOURCE-CODE VIEW

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



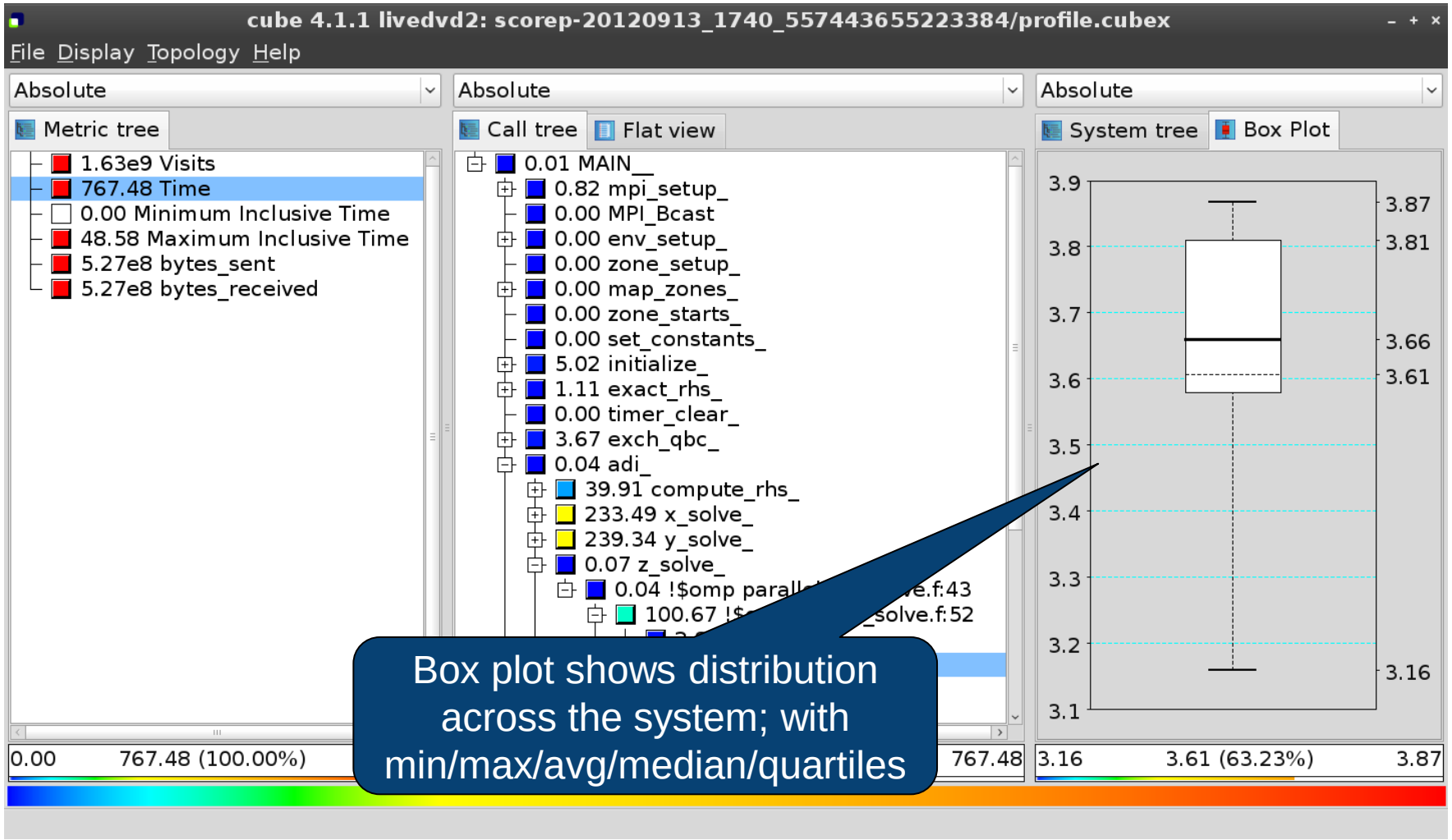
Note:
This feature depends on file and line number information provided by the instrumentation, i.e., it may not always be available

FLAT PROFILE VIEW

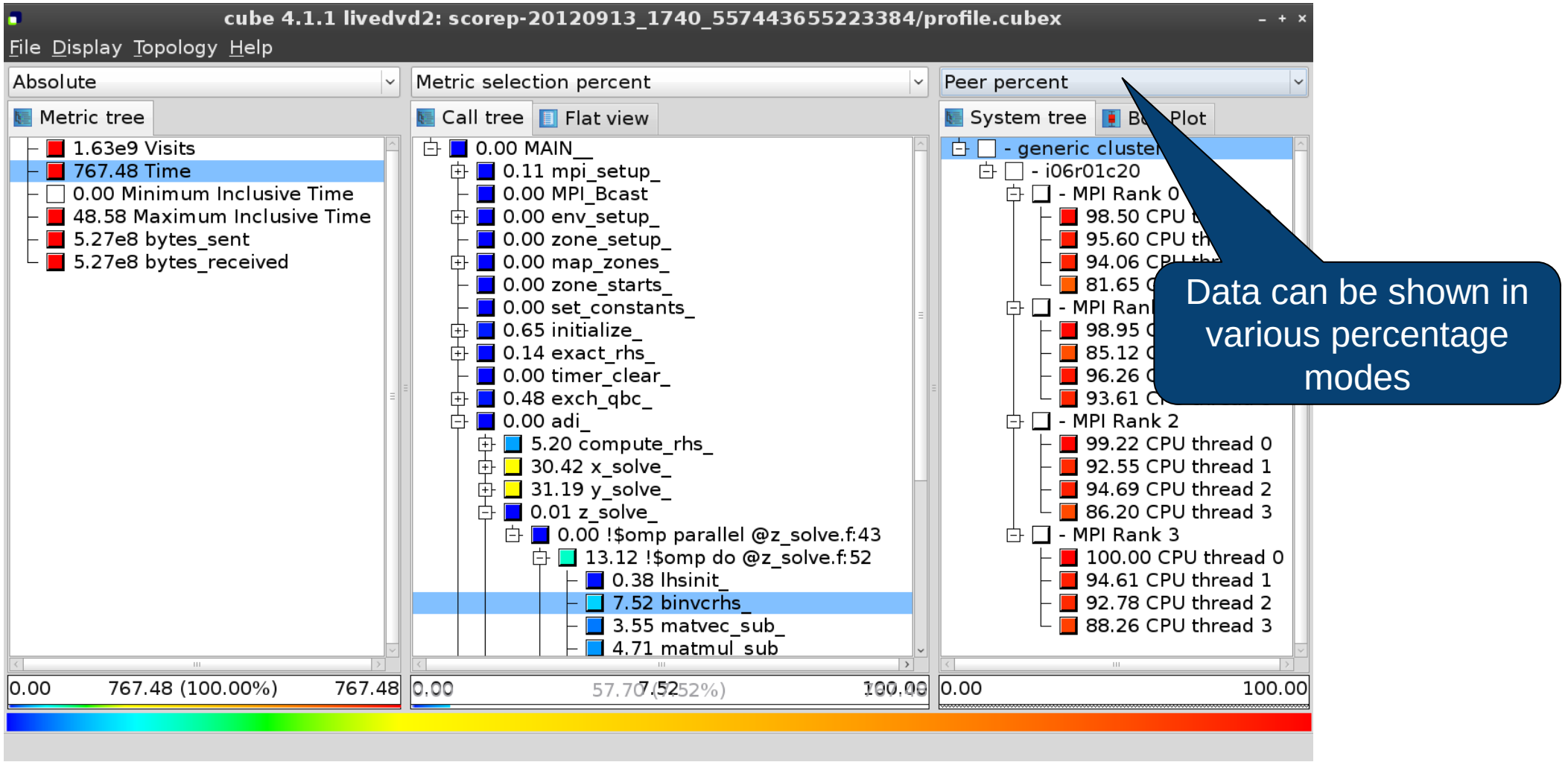


Select flat view tab,
expand all nodes,
and sort by exclusive value

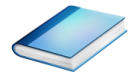
BOX PLOT VIEW



ALTERNATIVE DISPLAY MODES



IMPORTANT DISPLAY MODES

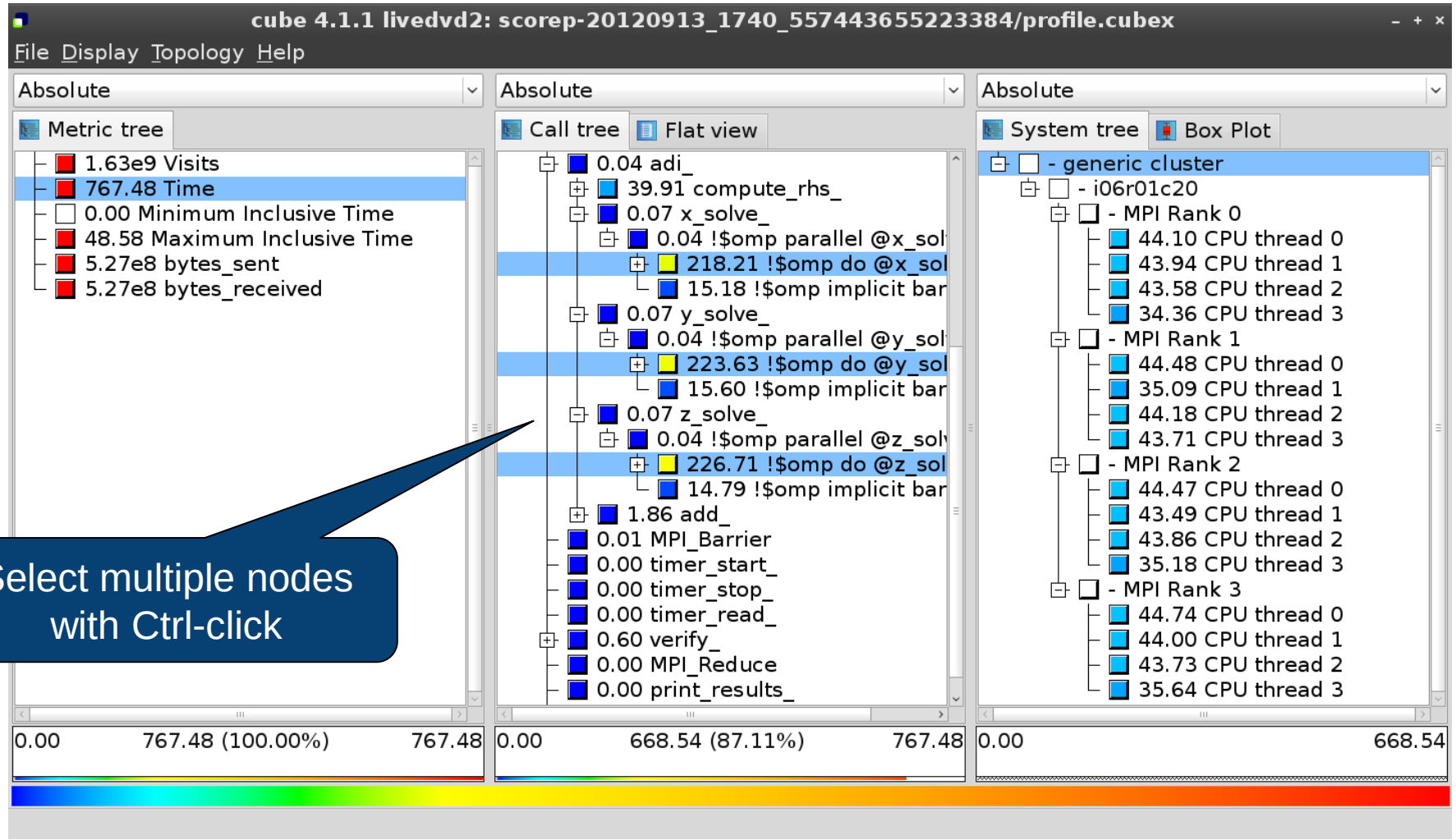


- Absolute
  - Absolute value shown in seconds/bytes/counts

- Selection percent
  - Value shown as percentage w.r.t. the selected node “on the left” (metric/call path)

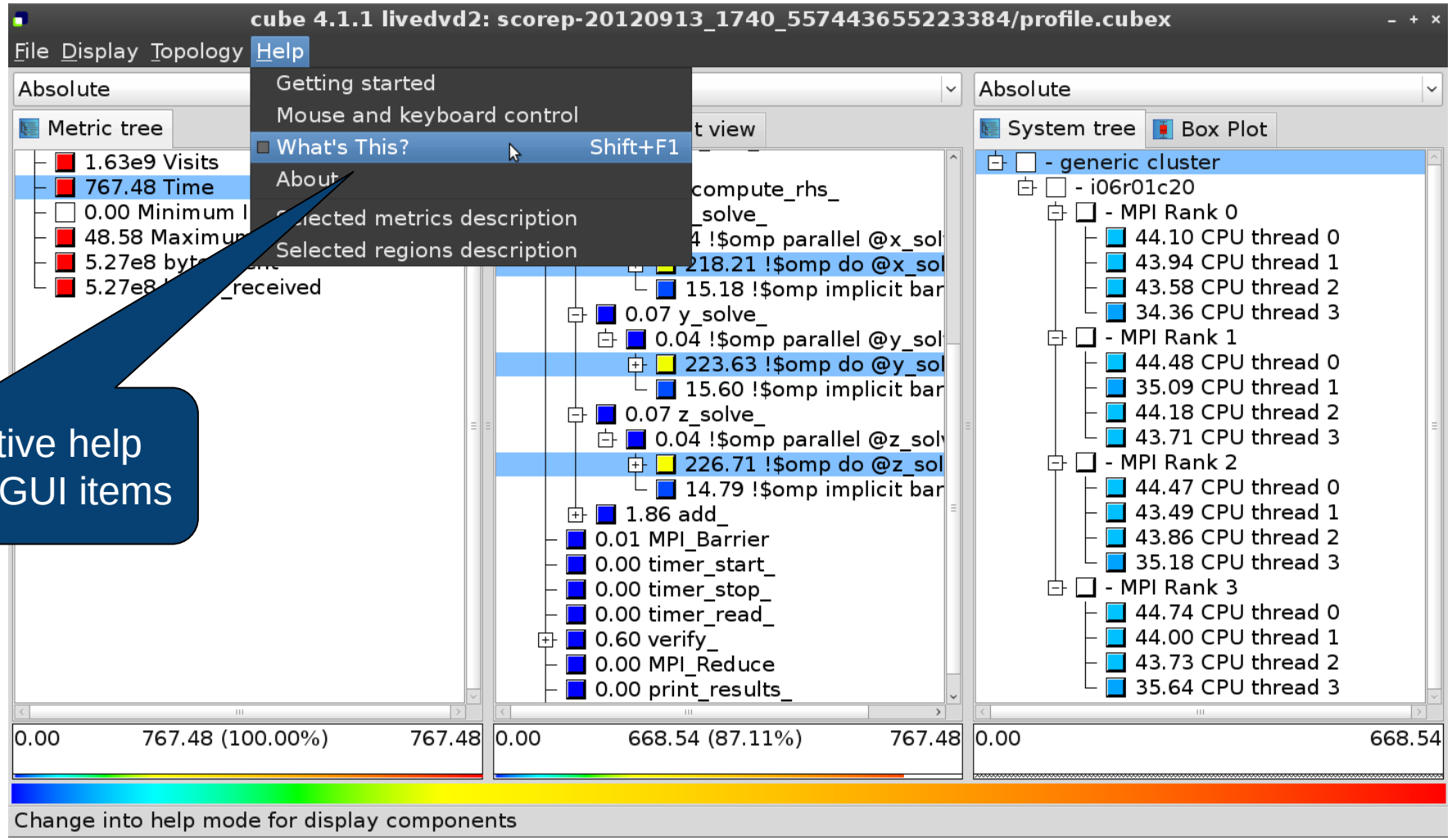
- Peer percent (system tree only)
  - Value shown as percentage relative to the maximum peer value

MULTIPLE SELECTION



CONTEXT-SENSITIVE HELP

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



Context-sensitive help available for all GUI items

DERIVED METRICS

- Derived metrics are defined using CubePL expressions, e.g.:

  metric::time(i)/metric::visits(e)

- Values of derived metrics are not stored, but calculated on-the-fly
- Types of derived metrics:
  - Prederived: evaluation of the CubePL expression is performed before aggregation
  - Postderived: evaluation of the CubePL expression is performed after aggregation

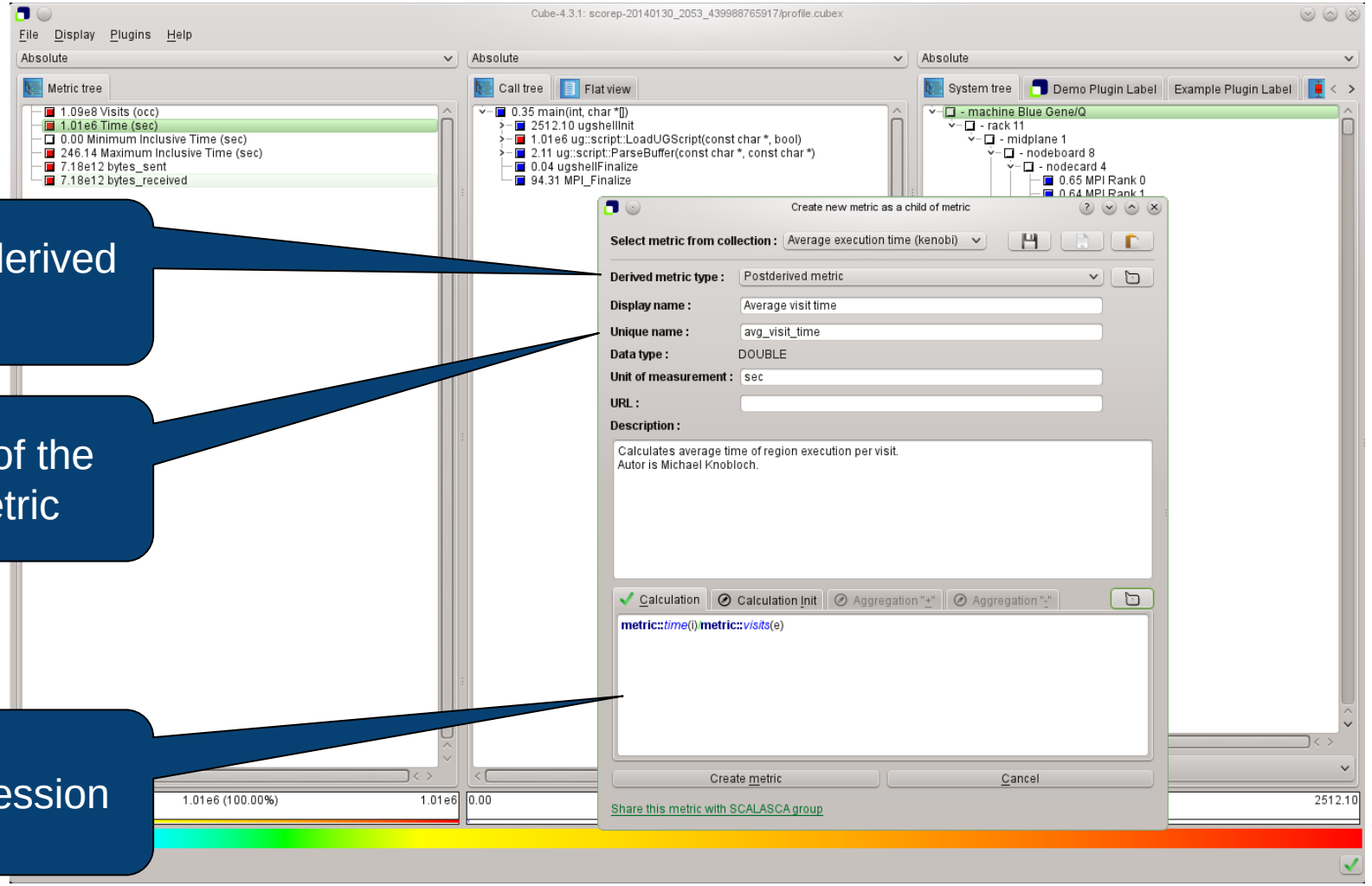
- Examples:
  - “Average execution time”: Postderived metric with expression

    metric::time(i)/metric::visits(e)

  - “Number of FLOP per second”: Postderived metric with expression

    metric::FLOP()/metric::time()

DERIVED METRICS IN CUBE GUI

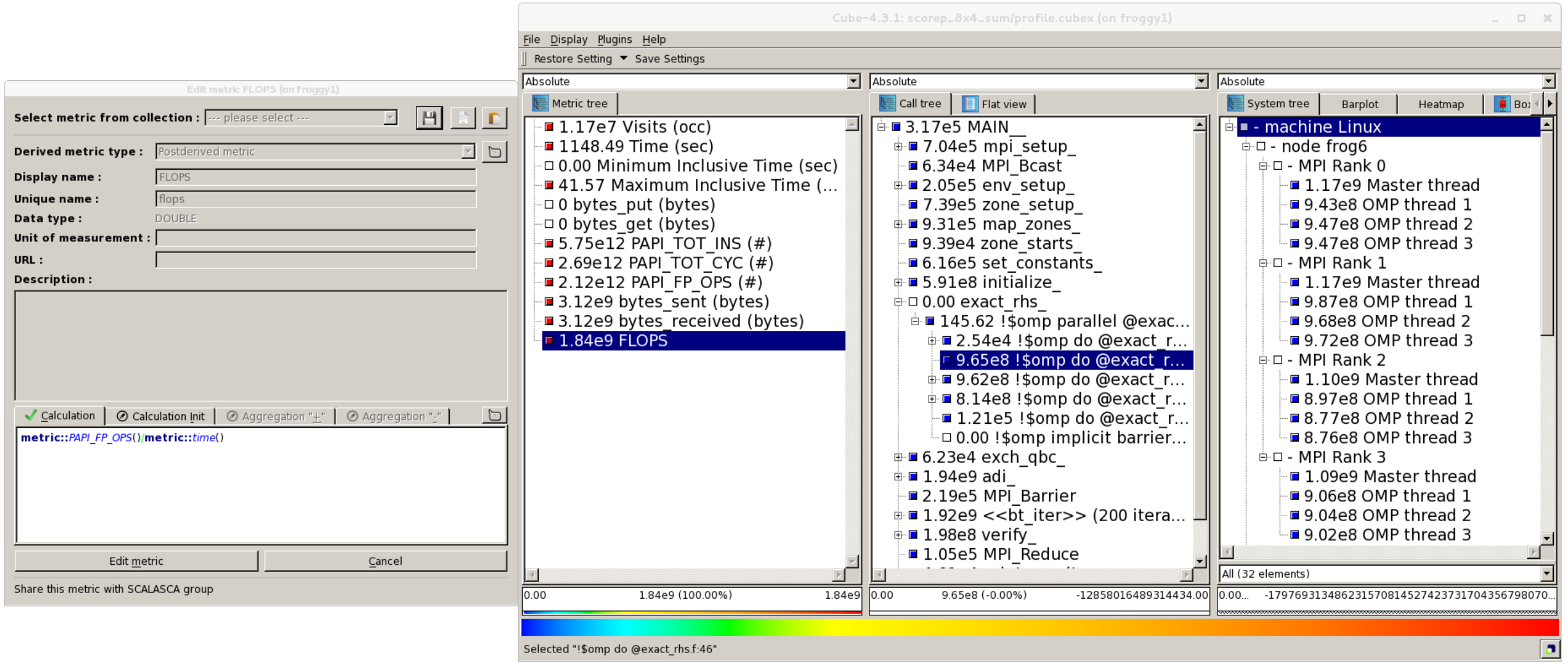


Collection of derived metrics

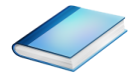
Parameters of the derived metric

CubePL expression

EXAMPLE: FLOPS BASED ON PAPI_FP_OPS AND TIME



ITERATION PROFILING



- Show time dependent behavior by “unrolling” iterations

- Preparations:
  - Mark loop body by using Score-P instrumentation API in your source code

```
SCOREP_USER_REGION_DEFINE( scorep_bt_loop )
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )
SCOREP_USER_REGION_END( scorep_bt_loop )
```

- Result in the Cube profile:
  - Iterations shown as separate call trees

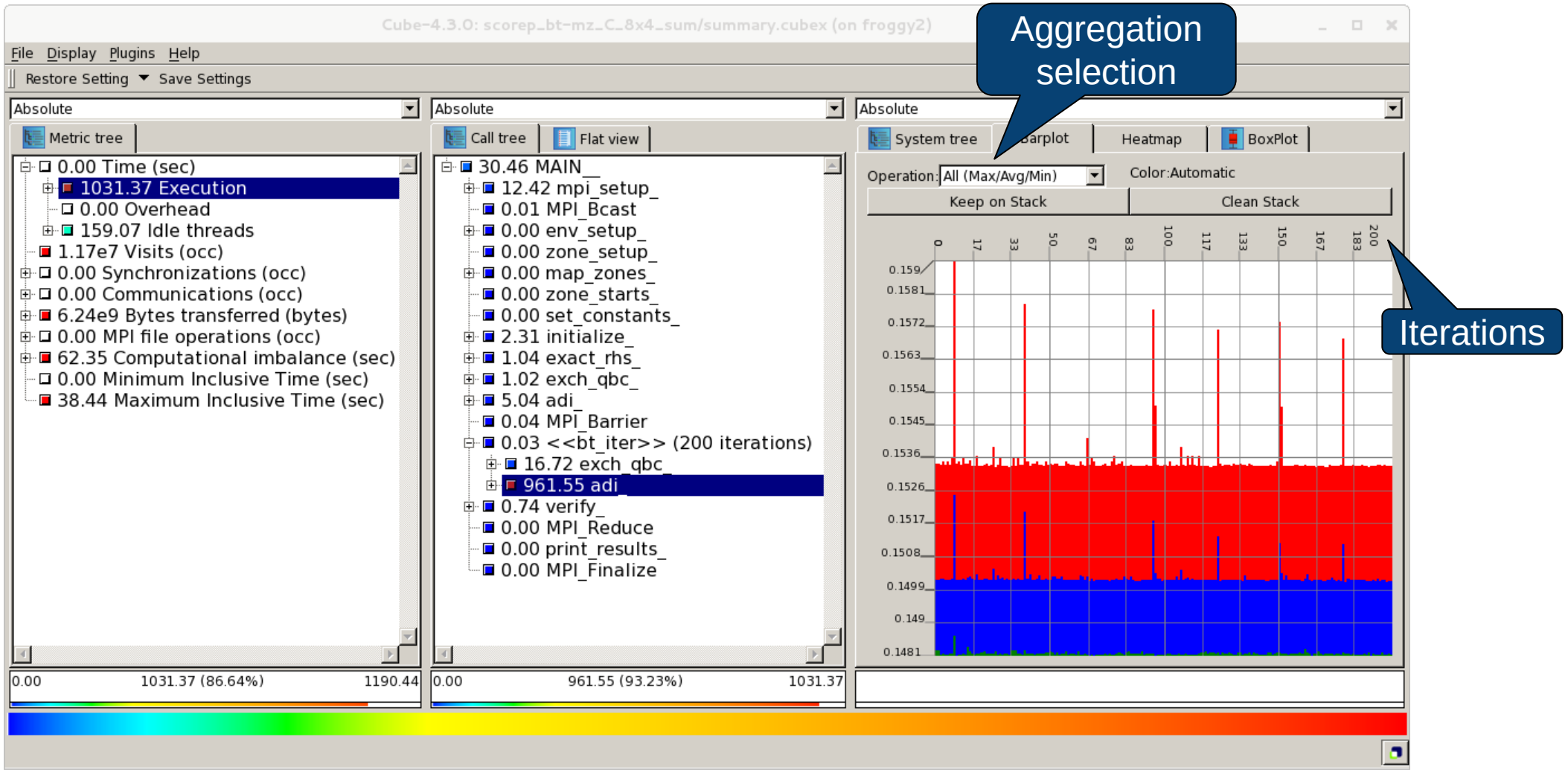
  ➢ Useful for checking results for specific iterations

  or

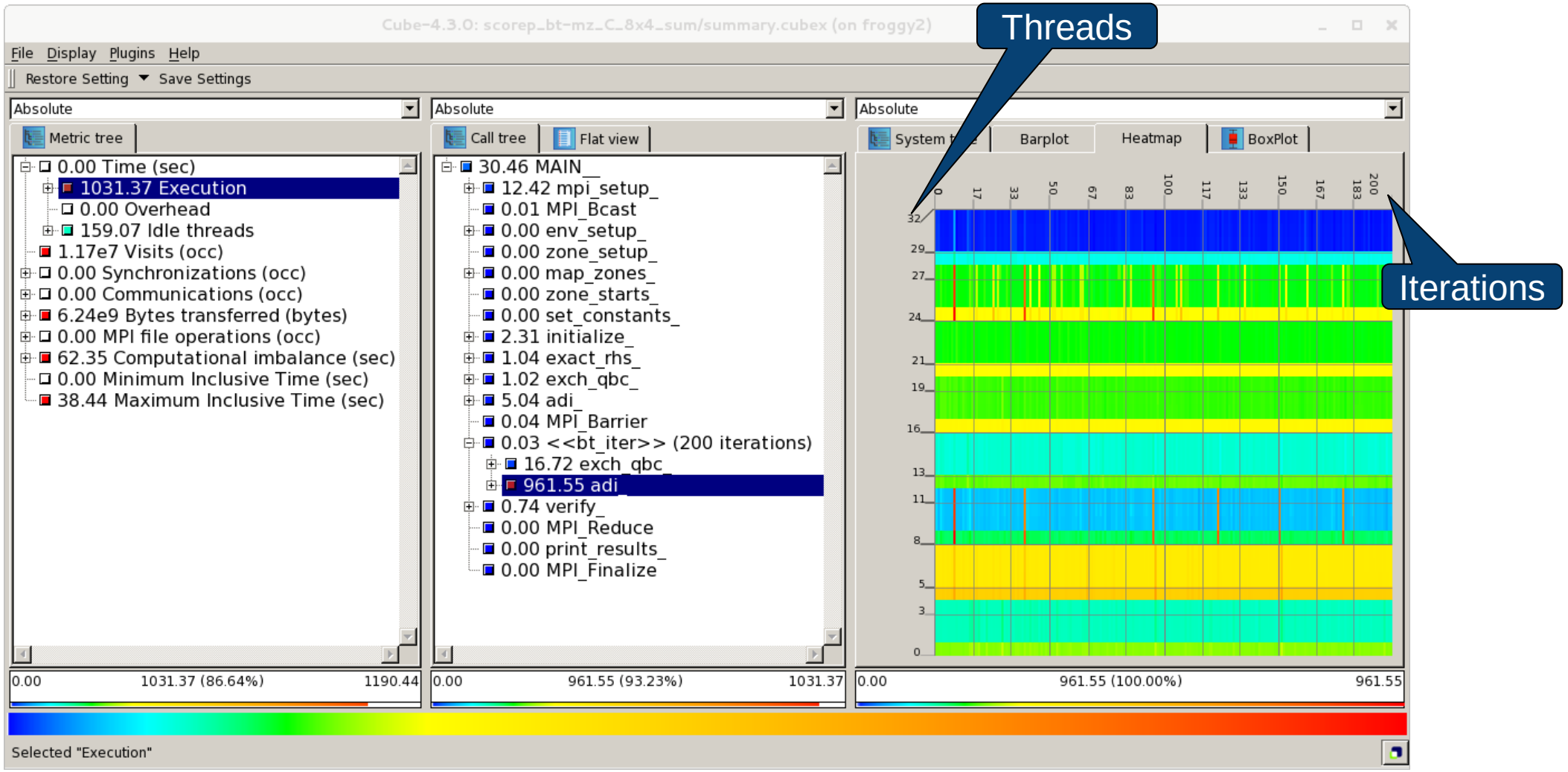
  - Select your user-instrumented region and mark it as loop
  - Choose “Hide iterations”

  ➢ View the Barplot statistics or the (thread x iterations) Heatmap

ITERATION PROFILING: BARPLOT



ITERATION PROFILING: HEATMAP



CUBE ALGEBRA UTILITIES

- Extracting solver sub-tree from analysis report

```
% cube_cut  -r '<<ITERATION>>'  scorep_bt-mz_C_16x8_sum/profile.cubex
Writing cut.cubex... done.
```

- Calculating difference of two reports

```
% cube_diff  scorep_bt-mz_C_16x8_sum/profile.cubex  cut.cubex
Writing diff.cubex... done.
```

- Additional utilities for merging, calculating mean, etc.
- Default output of cube_*utility* is a new report *utility*.cubex
- Further utilities for report scoring & statistics
- Run utility with `-h' (or no arguments) for brief usage info

SQUARE SNEAK PREVIEW

- Scalasca provides ***square*** to facilitate analysis report exploration
  - square = scalasca –examine [OPTIONS] (./scorep_expt_sum | ./profile.cubex)
- Processes intermediate .cubex files produced by Score-P and Scout
  - profile.cubex -> summary.cubex
  - scout.cubex -> trace.cubex
- and (optionally) starts CUBE GUI with the post-processed file
  - containing additional derived metrics and metric hierarchies

BT-MZ @ LEVANTE
SCORING & FILTERING

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

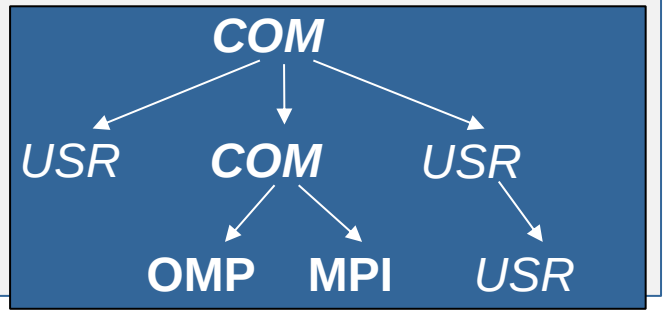
- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

BT-MZ SUMMARY ANALYSIS RESULT SCORING

```
% scorep-score scorep_bt-mz_sum/profile.cubex

Estimated aggregate size of event trace:                   160GB
Estimated requirements for largest trace buffer (max_buf): 11GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       11GB
(warning: The memory requirements cannot be satisfied by Score-P to avoid
 intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the
 maximum supported memory or reduce requirements using USR regions filters.)

flt     type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
         ALL 10,791,335,059 6,589,342,123 2360.04   100.0           0.36  ALL
         USR 10,754,591,276 6,574,805,745  926.34    39.3           0.14  USR
         OMP     34,990,128    13,667,328 1241.50    52.6          90.84  OMP
         COM      1,178,450       725,200    1.97     0.1           2.71  COM
         MPI        616,168       143,834  190.23     8.1        1322.55  MPI
      SCOREP             41            16    0.01     0.0         372.15  SCOREP
```



- Report scoring as textual output

160 GB total memory
11 GB per rank!

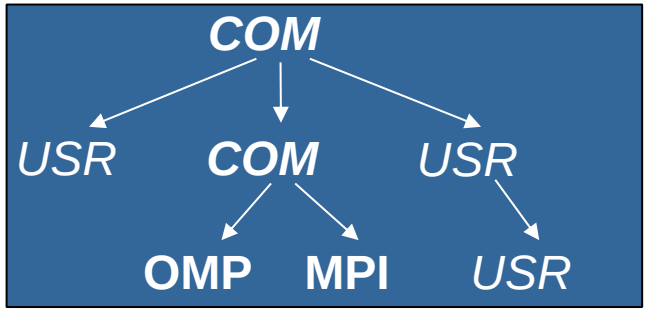
- Region/callpath classification
  - **MPI** pure MPI functions
  - **OMP** pure OpenMP regions
  - **USR** user-level computation
  - **COM** "combined" USR+OpenMP/MPI
  - **ALL** aggregate of all region types

BT-MZ SUMMARY ANALYSIS REPORT BREAKDOWN

- Score report breakdown by region

```
% scorep-score -r scorep_bt-mz_sum/profile.cubex
  [...]
  [...]
flt      type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
         ALL 10,791,335,059 6,589,342,123 2360.04   100.0           0.36  ALL
         USR 10,754,591,276 6,574,805,745  926.34    39.3           0.14  USR
         OMP     34,990,128    13,667,328 1241.50    52.6          90.84  OMP
         COM      1,178,450       725,200    1.97     0.1           2.71  COM
         MPI        616,168       143,834  190.23     8.1        1322.55  MPI
      SCOREP             41            16    0.01     0.0         372.15  SCOREP

         USR  3,454,903,374 2,110,313,472  373.15    15.8           0.18  binvcrhs_
         USR  3,454,903,374 2,110,313,472  218.75     9.3           0.10  matvec_sub_
         USR  3,454,903,374 2,110,313,472  303.12    12.8           0.14  matmul_sub_
         USR    149,170,944    87,475,200   14.95     0.6           0.17  lhsinit_
         USR    149,170,944    87,475,200    9.69     0.4           0.11  binvrhs_
         USR    112,148,088    68,892,672    6.69     0.3           0.10  exact_solution
```



More than
10 GB just for these
6 regions

BT-MZ SUMMARY ANALYSIS SCORE

- Summary measurement analysis score reveals
  - Total size of event trace would be ~160 GB
  - Maximum trace buffer size would be ~11 GB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.5% of the trace requirements are for USR regions
    - purely computational routines never found on COM call-paths common to communication routines or OpenMP parallel regions
  - These USR regions contribute around 39% of total time
    - however, much of that is very likely to be measurement overhead for frequently-executed small routines
- Advisable to tune measurement configuration
  - Specify an adequate trace buffer size
  - Specify a filter file listing (USR) regions not to be measured

BT-MZ SUMMARY ANALYSIS REPORT FILTERING

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs*
    matmul_sub*
    matvec_sub*
    exact_solution*
    binvrhs*
    lhs*init*
    timer_*
SCOREP_REGION_NAMES_END

% scorep-score -f ../config/scorep.filt -c 2 \
      scorep_bt-mz_sum/profile.cubex

Estimated aggregate size of event trace:                   1381MB
Estimated requirements for largest trace buffer (max_buf): 87MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       97MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=97MB to avoid
       intermediate flushes or reduce requirements using
       USR regions filters.)
```

- Report scoring with prospective filter listing 7 USR regions

1.4 GB of memory in total, 87 MB per rank!

(Including 2 metric values)

BT-MZ SUMMARY ANALYSIS REPORT FILTERING

```
% scorep-score -r -f ../config/scorep.filt \
      scorep_bt-mz_sum/profile.cubex
flt     type     max_buf[B]        visits time[s] time[%] time/     region
                                                 visit[us]
 -       ALL 10,791,335,059 6,589,342,123 2360.04   100.0     0.36  ALL
 -       USR 10,754,591,276 6,574,805,745  926.34    39.3     0.14  USR
 -       OMP     34,990,128    13,667,328 1241.50    52.6    90.84  OMP
 -       COM      1,178,450       725,200    1.97     0.1     2.71  COM
 -       MPI        616,168       143,834  190.23     8.1  1322.55  MPI
 -    SCOREP             41            16    0.01     0.0   372.15  SCOREP

 *       ALL     36,820,329    14,558,235 1433.71    60.7    98.48  ALL-FLT
 +       FLT 10,754,555,760 6,574,783,888  926.33    39.3     0.14  FLT
 -       OMP     34,990,128    13,667,328 1241.50    52.6    90.84  OMP-FLT
 *       COM      1,178,450       725,200    1.97     0.1     2.71  COM-FLT
 -       MPI        616,168       143,834  190.23     8.1  1322.55  MPI-FLT
 *       USR         35,542        21,857    0.01     0.0     0.28  USR-FLT
 -    SCOREP             41            16    0.01     0.0   372.15  SCOREP-FLT

 +       USR  3,454,903,374 2,110,313,472  373.15    15.8     0.18  binvcrhs_
 +       USR  3,454,903,374 2,110,313,472  218.75     9.3     0.10  matvec_sub_
 +       USR  3,454,903,374 2,110,313,472  303.12    12.8     0.14  matmul_sub_
 +       USR    149,170,944    87,475,200   14.95     0.6     0.17  lhsinit_
 +       USR    149,170,944    87,475,200    9.69     0.4     0.11  binvrhs_
 +       USR    112,148,088    68,892,672    6.69     0.3     0.10  exact_solution
```

- Score report breakdown by region (w/o additional metrics)

Filtered routines marked with '+'

BT-MZ FILTERED SUMMARY MEASUREMENT

```
% cd bin.scorep
% cp ../jobscript/levante/scorep.sbatch .
% vi scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_METRIC_RUSAGE=ru_stime
#export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss

# Run the application
mpirun -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS

% sbatch --account=kg0166 scorep.sbatch
```

- Set new experiment directory and re-run measurement with new filter configuration
- Submit job

SCORE-P FILTERING

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs*
    matmul_sub*
    matvec_sub*
    exact_solution*
    binvrhs*
    lhs*init*
    timer_*
SCOREP_REGION_NAMES_END

% export SCOREP_FILTERING_FILE=\
../config/scorep.filt
```

Region name filter block using wildcards

Apply filter

- Filtering by source file name
  - All regions in files that are excluded by the filter are ignored
- Filtering by region name
  - All regions that are excluded by the filter are ignored
  - Overruled by source file filter for excluded files
- Apply filter by
  - exporting

SOURCE FILE NAME FILTER BLOCK

- Keywords
  - Case-sensitive
  - `SCOREP_FILE_NAMES_BEGIN`, `SCOREP_FILE_NAMES_END`
    - Define the source file name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated source file names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
    - Unlike bash, `*` may match a string that contains slashes
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions in source files that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_FILE_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE */foo/bar*
  INCLUDE */filter_test.c
SCOREP_FILE_NAMES_END
```

REGION NAME FILTER BLOCK

- Keywords
  - Case-sensitive
  - `SCOREP_REGION_NAMES_BEGIN`, `SCOREP_REGION_NAMES_END`
    - Define the region name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated region names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_REGION_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE *
  INCLUDE bar foo
          baz
          main
SCOREP_REGION_NAMES_END
```

REGION NAME FILTER BLOCK, MANGLING

- Name mangling
  - Filtering based on names seen by the measurement system
    - Dependent on compiler
    - Actual name may be mangled
- `scorep-score` names as starting point

  (e.g. `matvec_sub_`)
  - Use `*` for Fortran trailing underscore(s) for portability
  - Use `?` and `*` as needed for full signatures or overloading
  - Use `\` to escape special characters

```
void bar(int* a) {
    *a++;
}
int main() {
    int i = 42;
    bar(&i);
    return 0;
}
```

```
# filter bar:
# for gcc-plugin, scorep-score
# displays 'void bar(int*)',
# other compilers may differ

SCOREP_REGION_NAMES_BEGIN
  EXCLUDE void?bar(int?)
SCOREP_REGION_NAMES_END
```

BT-MZ @ LEVANTE

SCALASCA TRACE ANALYSIS

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 2.6
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
       --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
```

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:
  -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

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
   -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)

BT-MZ SUMMARY MEASUREMENT COLLECTION...

```
% cd bin.scorep
% cp ../jobscript/levante/scalasca.sbatch .
% cat scalasca.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=32M


# run the application
scalasca -analyze mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

```
% sbatch --account=kg0166 scalasca.sbatch
```

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

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

BT-MZ SUMMARY MEASUREMENT

```
S=C=A=N: Scalasca 2.6 runtime summarization
S=C=A=N: ./scorep_bt-mz_C_28x4_sum experiment archive
S=C=A=N: Thu Jun  10 11:48:50 2021: Collect start
mpirun./bt-mz_C.28

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

 Number of zones:  8 x  8
 Iterations: 200    dt:   0.000300
 Number of active processes:    28

 [... More application output ...]

S=C=A=N: Thu Jun  10 11:49:02 2021: Collect done (status=0) 12s
S=C=A=N: ./scorep_bt-mz_C_28x4_sum complete.
```

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command

- Creates experiment directory:
  scorep_bt-mz_C_28x4_sum

BT-MZ SUMMARY ANALYSIS REPORT EXAMINATION

- Score summary analysis report

```
% square -s  scorep_bt-mz_B_28x4_sum
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Score report written to ./scorep_bt-mz_B_28x4_sum/scorep.score
```

- Post-processing and interactive exploration with Cube

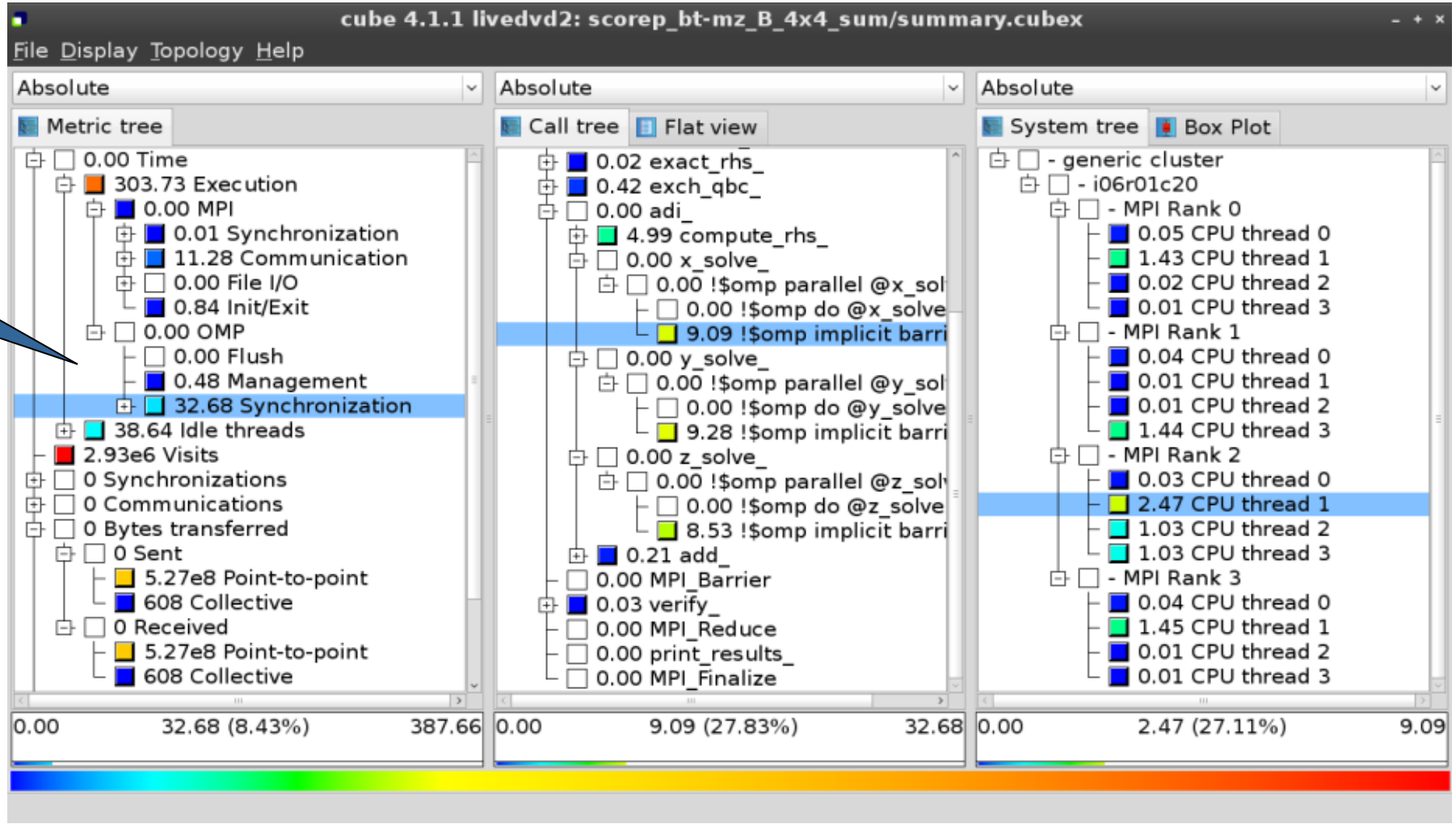
```
% square  scorep_bt-mz_B_28x4_sum
INFO: Displaying ./scorep_bt-mz_B_28x4_sum/summary.cubex...
                    [GUI showing summary analysis report]
```

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

POST-PROCESSED SUMMARY ANALYSIS

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/levante/scalasca.sbatch .
% vim scalasca.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=46M

# run the application
scalasca -analyze -t mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

```
% sbatch --account=kg0166 scalasca.sbatch
```

- Change to directory with the Score-P instrumented executable 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 trace collection and analysis
S=C=A=N: Thu Jun  10 12:05:30 2021: Collect start
mpirun./bt-mz_C.28

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

 Number of zones:  8 x  8
 Iterations: 200    dt:   0.000300
 Number of active processes:    28

 [... More application output ...]

S=C=A=N: Thu Jun  10 12:05:44 2021: Collect done (status=0) 14s
```

- Starts measurement with collection of trace files …

BT-MZ TRACE MEASUREMENT ... ANALYSIS

```
…
S=C=A=N: Thu Jun  10 12:05:44 2021: Analyze start
 mpirun scout.hyb --time-correct \
>    ./scorep_bt-mz_C_28x4_trace/traces.otf2

SCOUT   (Scalasca 2.6)

Analyzing experiment archive ./scorep_bt-mz_C_28x4_trace/traces.otf2

Opening experiment archive ... done (0.002s).
Reading definition data    ... done (0.004s).
Reading event trace data   ... done (0.113s).
Preprocessing              ... done (0.179s).
Timestamp correction       ... done (0.431s).
Analyzing trace data       ... done (5.174s).
Writing analysis report    ... done (0.175s).

Max. memory usage         : 422.312MB

        # passes         : 1
        # violated       : 0

Total processing time     : 6.140s
S=C=A=N: Thu Jun  10 12:05:51 2021: Analyze done (status=0) 7s
```

- 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

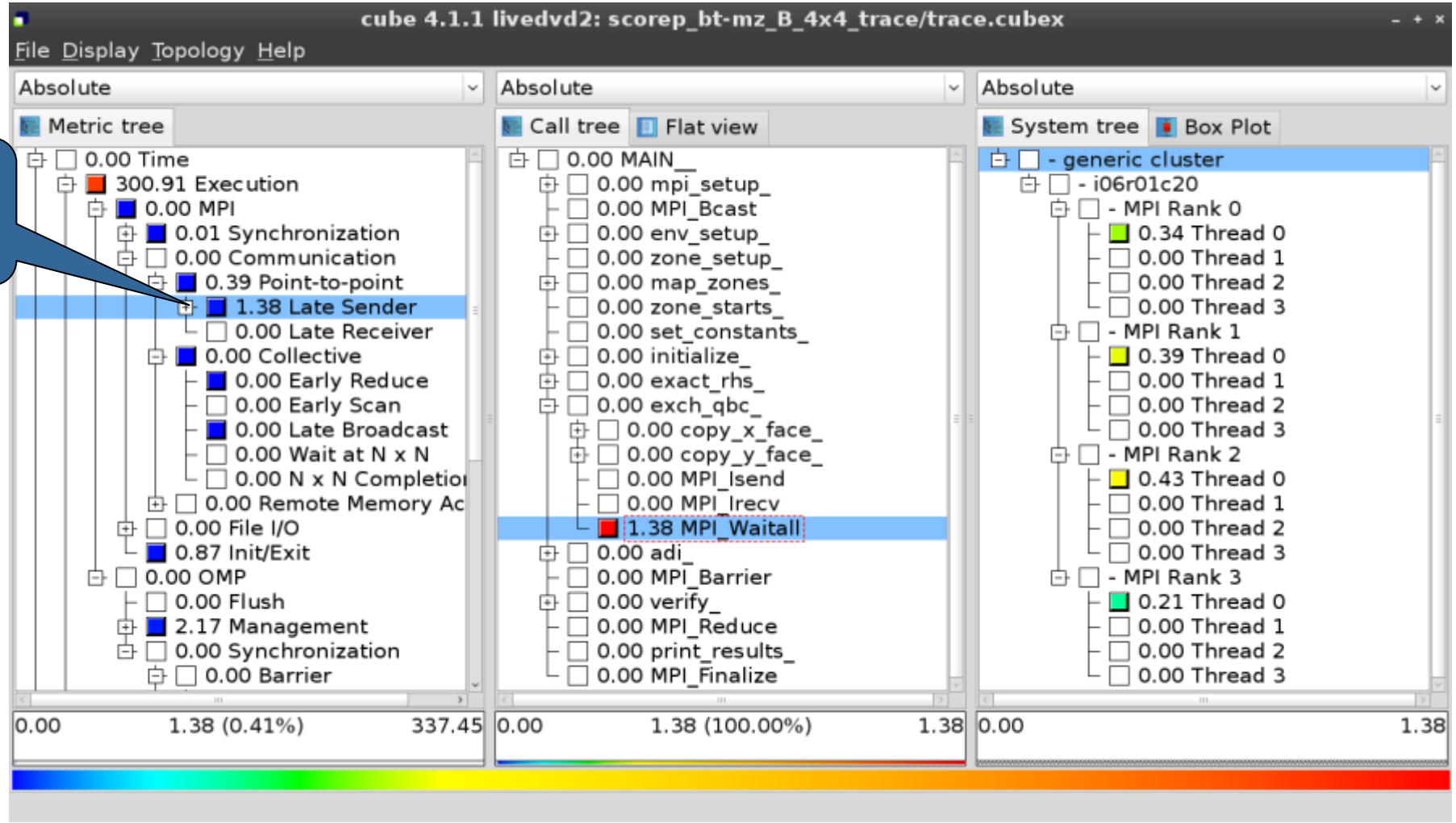
```
% square  scorep_bt-mz_C_28x4_trace
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_bt-mz_C_28x4_trace/trace.cubex...

                [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

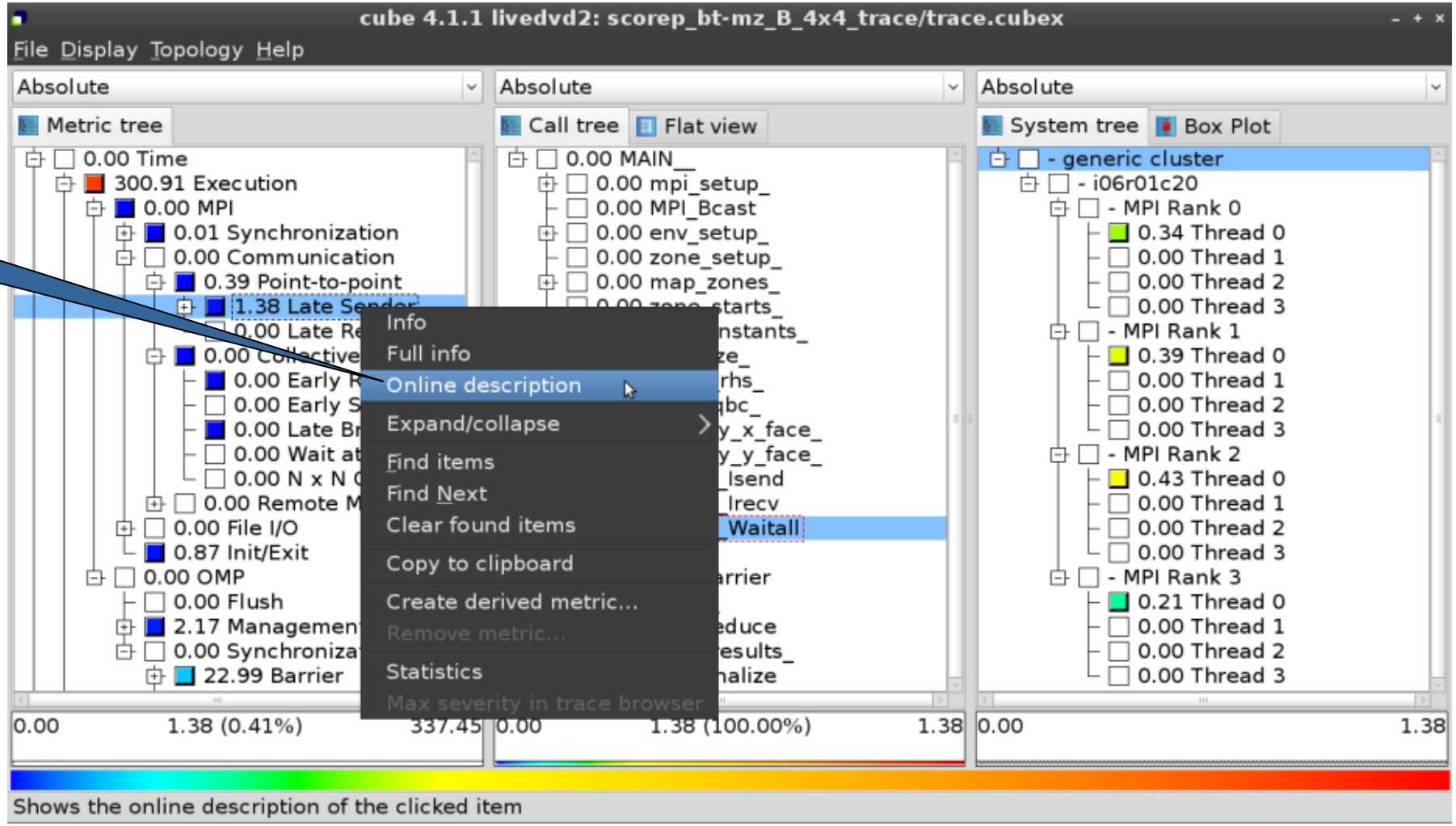
POST-PROCESSED TRACE ANALYSIS REPORT

Additional trace-based metrics in metric hierarchy



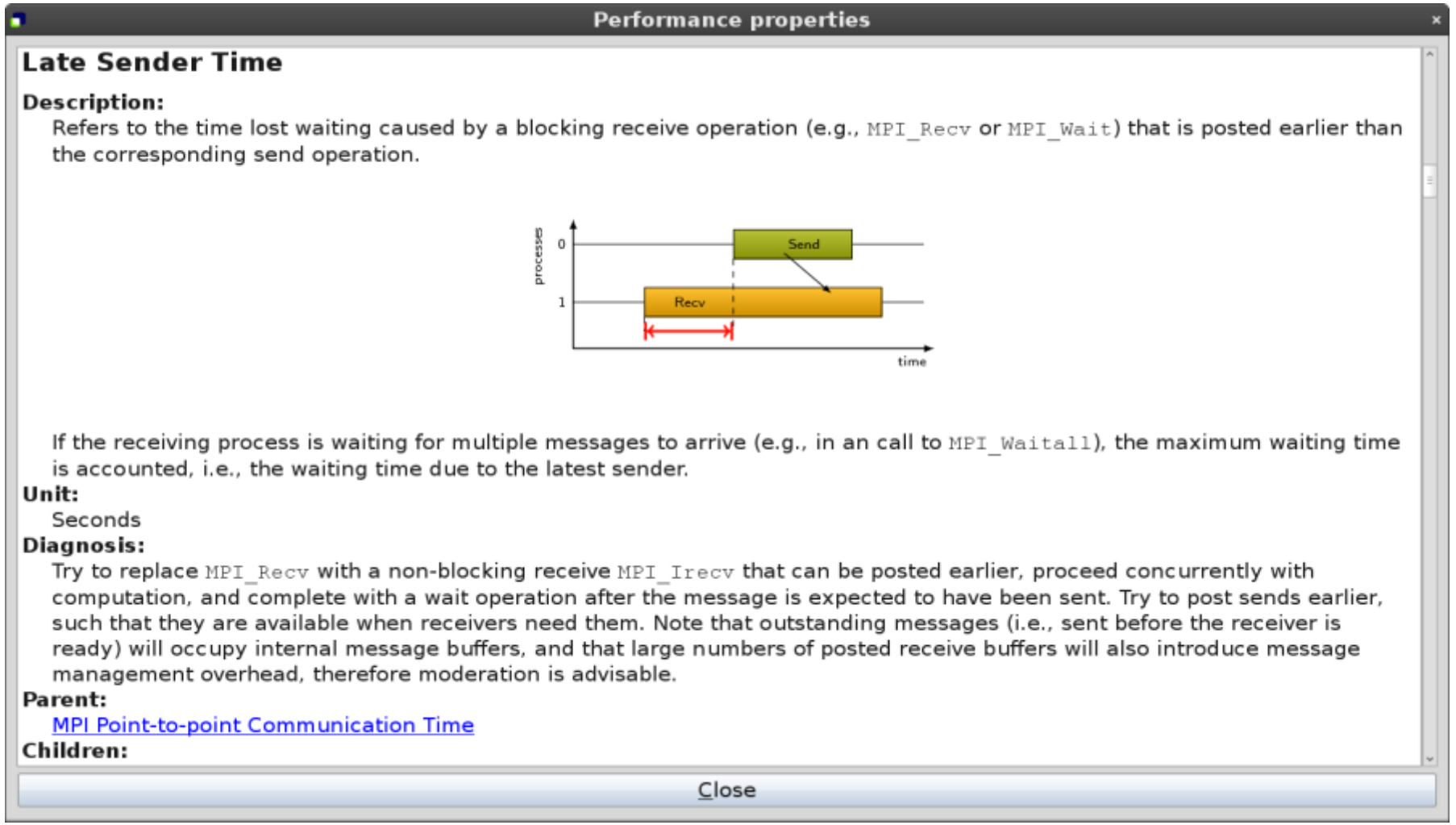
ONLINE METRIC DESCRIPTION

Access online metric description via context menu



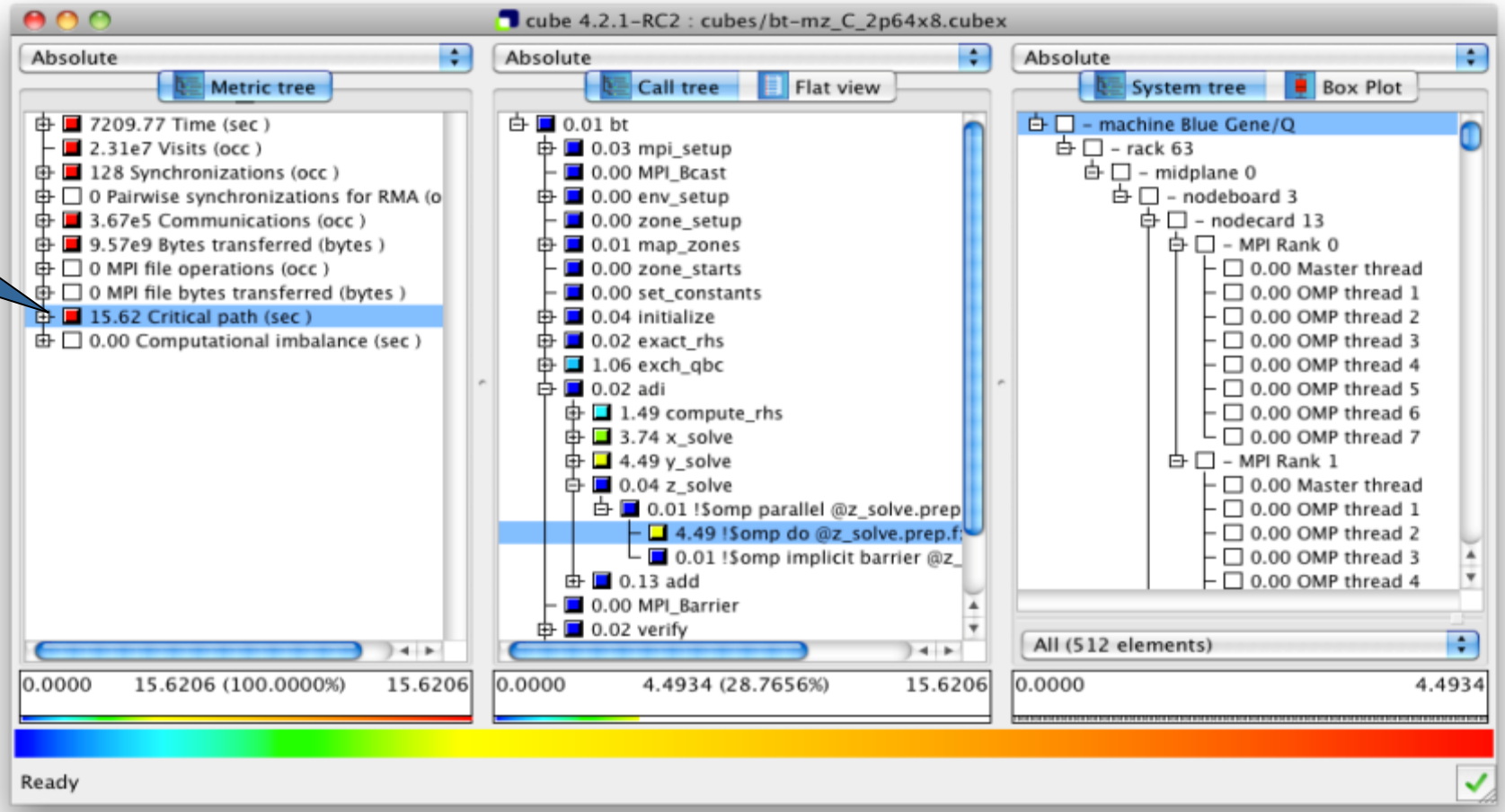
ONLINE METRIC DESCRIPTION

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



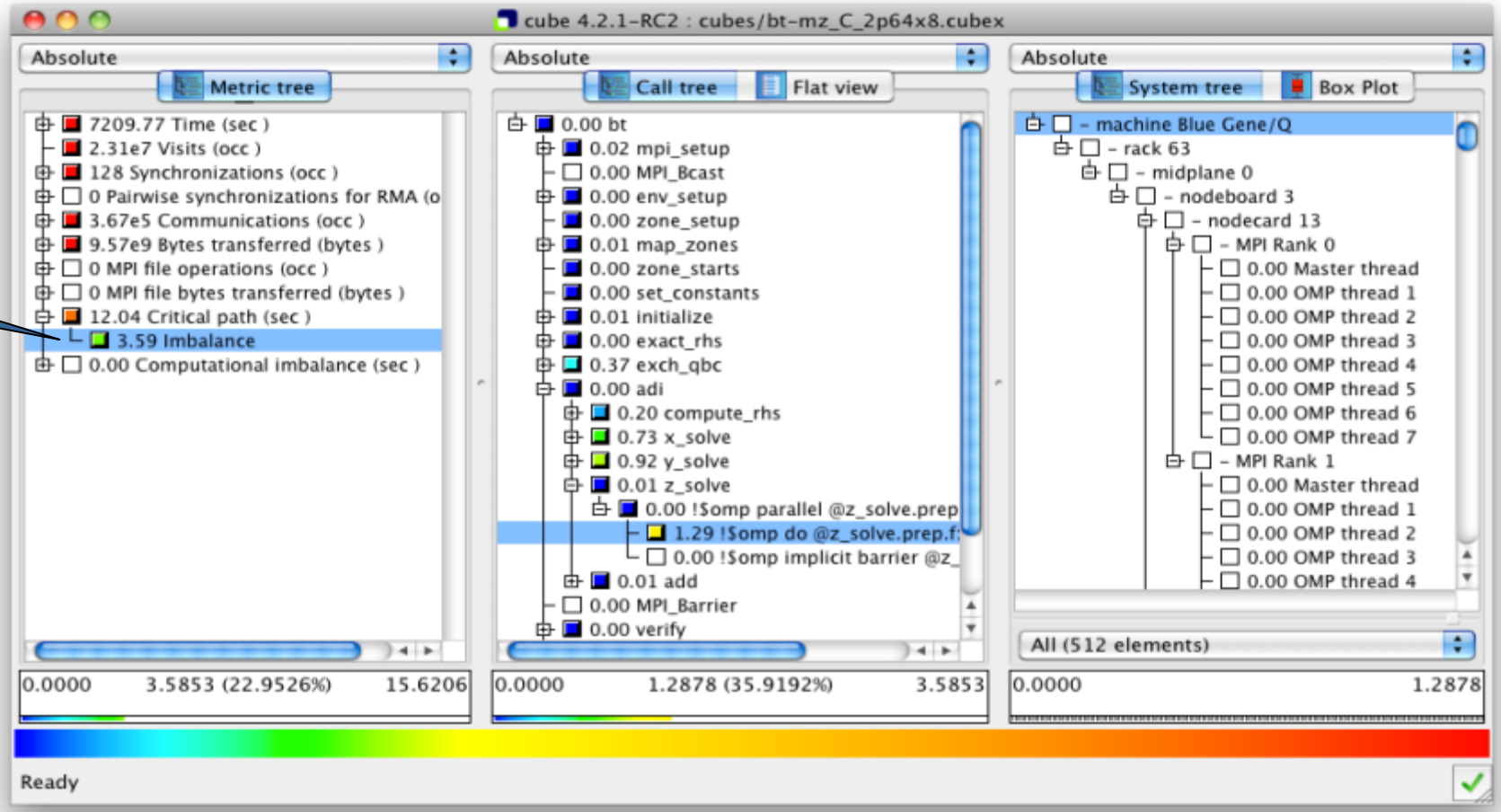
CRITICAL-PATH ANALYSIS

Critical-path profile shows wall-clock time impact



CRITICAL-PATH ANALYSIS

Critical-path imbalance highlights inefficient parallelism

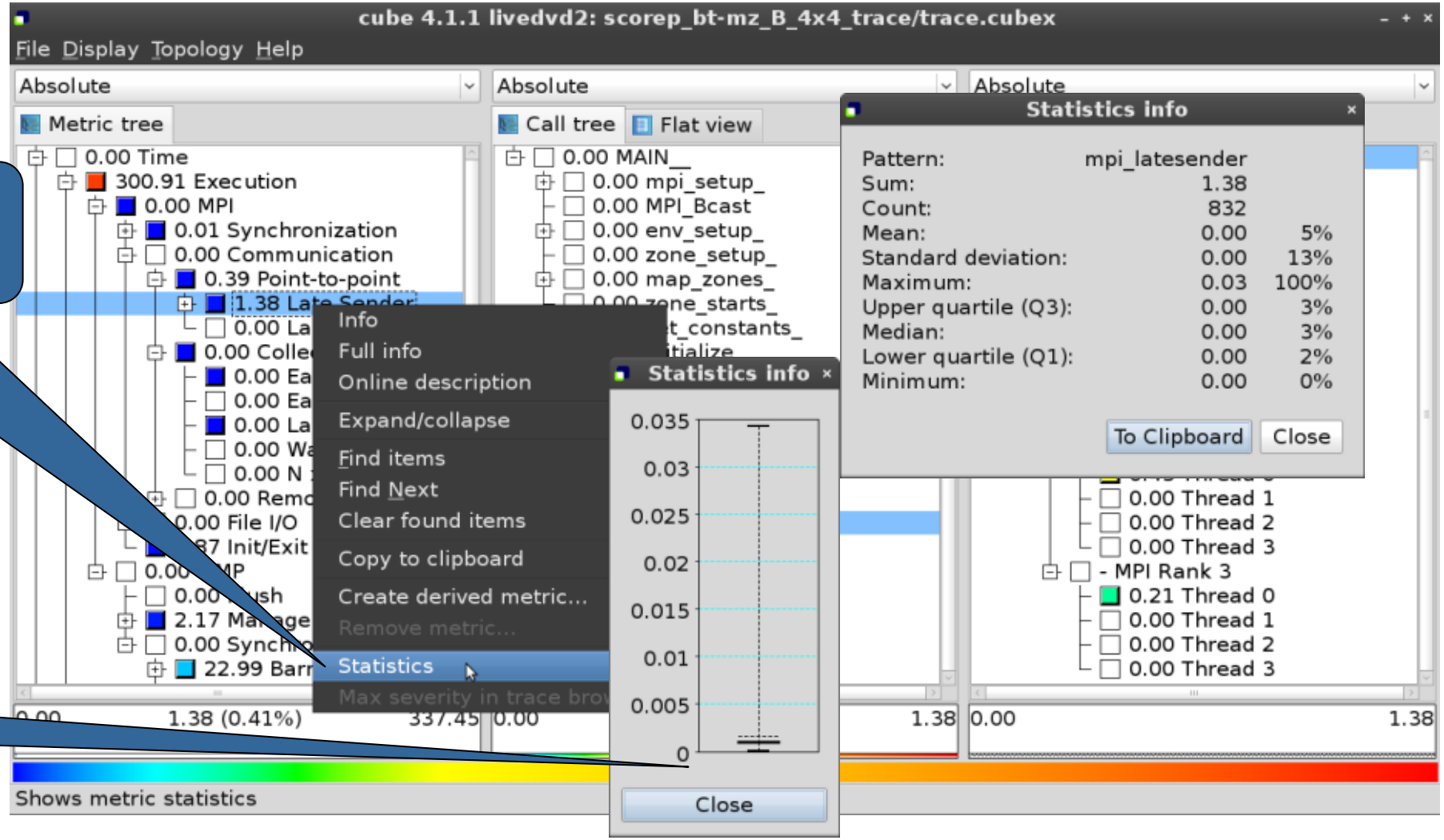


PATTERN INSTANCE STATISTICS

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

Access pattern instance statistics via context menu

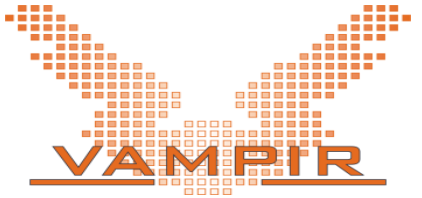
Click to get statistics details



BT-MZ @ LEVANTE

VAMPIR

VAMPIR EVENT TRACE VISUALIZER



- Offline trace visualization for Score-Ps OTF2 trace files
- Visualization of MPI, OpenMP and application events:
  - All diagrams highly customizable (through context menus)
  - Large variety of displays for ANY part of the trace
- http://www.vampir.eu

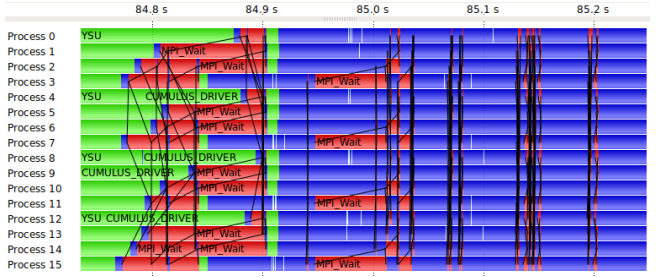
- Advantage:
  - Detailed view of dynamic application behavior
- Disadvantage:
  - Completely manual analysis
  - Too many details can hide the relevant parts

EVENT TRACE VISUALIZATION WITH VAMPIR

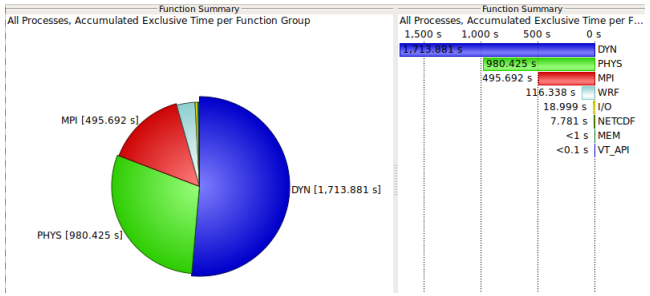
- Visualization of dynamic runtime behaviour at any level of detail along with statistics and performance metrics
- Alternative and supplement to automatic analysis

- **Typical questions that Vampir helps to answer**
  - What happens in my application execution during a given time in a given process or thread?
  - How do the communication patterns of my application execute on a real system?
  - Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

- **Timeline charts**
  - Application activities and communication along a time axis



- **Summary charts**
  - Quantitative results for the currently selected time interval



VAMPIR PERFORMANCE CHARTS

Timeline Charts


Master Timeline ➡ *all threads' activities*


Process Timeline ➡ *single thread's activities*


Summary Timeline ➡ *all threads' function call statistics*


Performance Radar ➡ *all threads' performance metrics*


Counter Data Timeline ➡ *single threads' performance metrics*


I/O Timeline ➡ *all threads' I/O activities*

Summary Charts


Function Summary


Message Summary

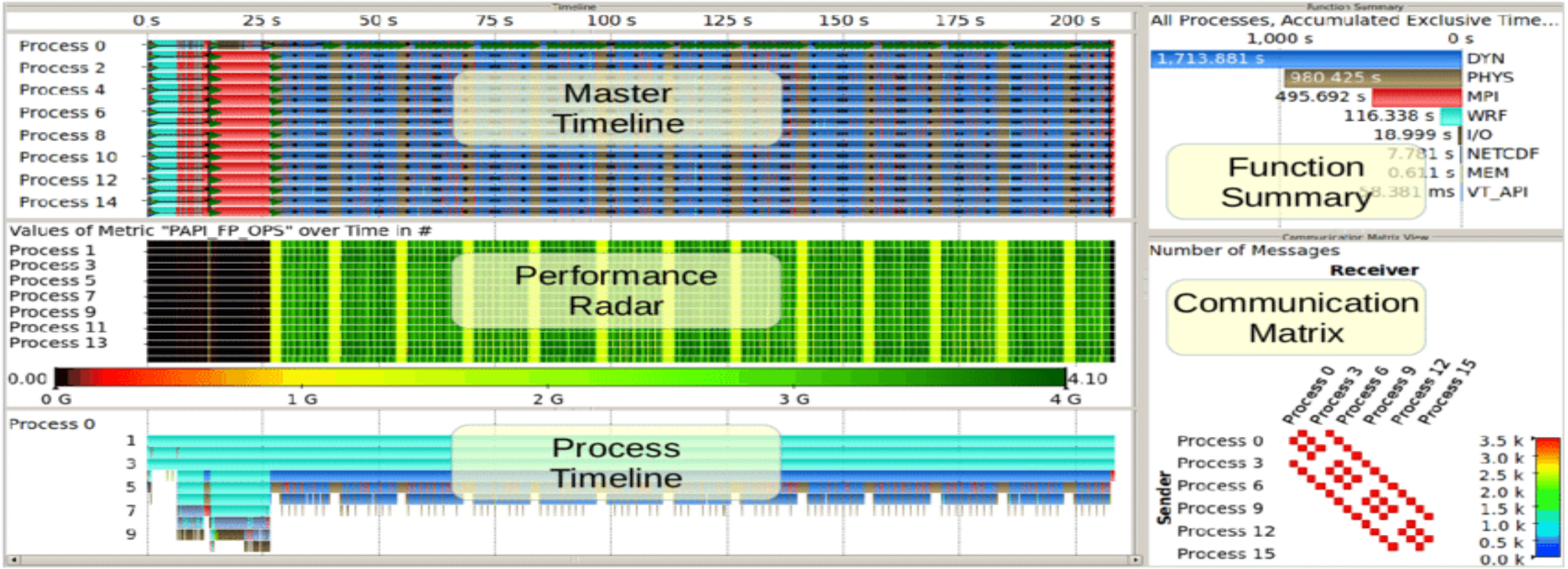

I/O Summary


Process Summary


Communication Matrix View


Call Tree

VAMPIR DISPLAYS



BT-MZ TRACE VISUALIZATION

- Visualize the generated trace files with Vampir

```
% vampir  scorep_bt-mz_C_28x4_trace/traces.otf2
                        [GUI showing trace timeline]
```

VAMPIR START VIEW

VAMPIR ZOOM


File Edit Chart Filter Window Help
Timeline
Function Summary
Accumulated Exclusive Time per Function
Function Legend
Application
Monitor
MPI
MPI Collective Communicat
MPI Management
MPI One-sided Communica
MPI Point-to-point Commun
MPI Request Handling
MPI Synchronization
MPI-I/O
OMP
OMP_LOOP
OMP_PARALLEL
OMP_SYNC
Communication Matrix View
Aggregated Message Volume
Receiver
Sender
Master thread:0
94.224 ms