Analysis Report Examination with Cube

Wadud Miah. Numerical Algorithms Group

Thanks to Markus Geimer of JSC

EU H2020 Centre of Excellence (CoE)

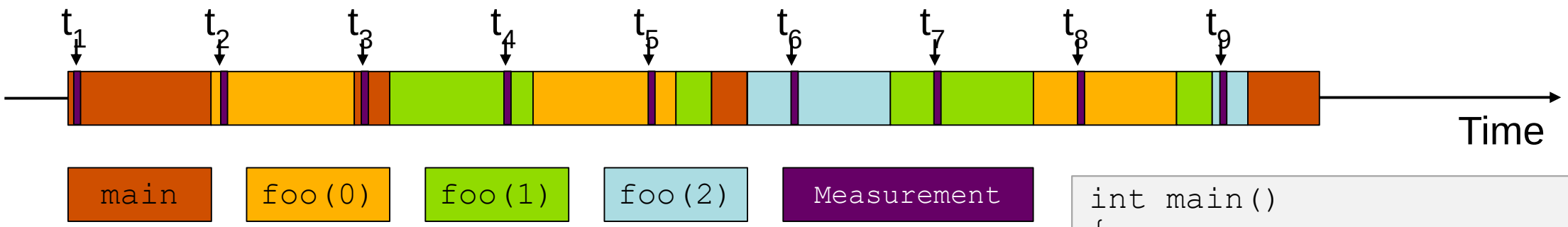
1 October 2015 – 31 March 2018

Grant Agreement No 676553

Scalasca

- Portable
- Open source
  - Developed by a consortium of German universities
- Measurement and analysis toolset
  - Very good scalability – targeting large scale codes, e.g. 28,672 MPI processes with 64 threads each (1,835,008 threads in total)
  - Automatic search for patterns of inefficient behaviour
  - Hardware counter information via PAPI
- Supports
  - Fortran, C, C++ (partial)
  - MPI, OpenMP, and hybrid OpenMP/MPI

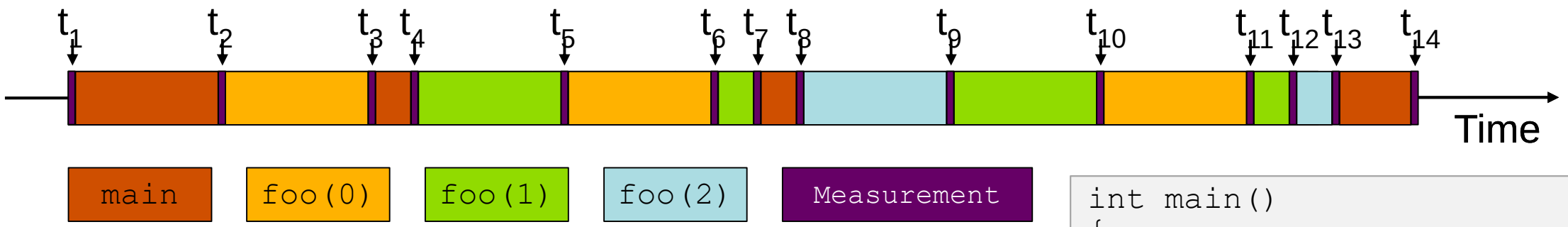
Sampling




```
int main()
{
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i - 1);

}
```

- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Service routine examines return-address stack
  - Addresses are mapped to routines using symbol table information
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

Instrumentation




```
int main()
{
  int i;
  Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
  Leave("main");
  return 0;
}

void foo(int i)
{
  Enter("foo");
  if (i > 0)
    foo(i - 1);
  Leave("foo");
}
```

- Measurement code is inserted such that every event of interest is captured directly
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

Scalasca - Overview

- Program sources get instrumentation added
- Instrumentation can be
  - **Automatic**, manual region, semi-automatic, selective
- Uses the community measurement system Score-P to generate profiles and tracing results.
  - **Instrument** the source code
  - **Analyse** the execution
    - Standard analysis is a runtime summarization
    - Optionally trace all calls and communication – large amounts of data produced
  - **Examine** the measurements

Scalasca – Quick Start I

- Compile/link stage. Prepend `scorep` and any instrumentation flags to your compile/link commands and add `-g` flag so your code can be viewed in Scalasca:

Original command	Instrumentation command
`mpicc -g -c foo.c`	`scorep mpicc -g -c foo.c`
`mpif90 -g -openmp -o bar bar.f90`	`scorep mpif90 -g -openmp -o bar bar.f90`

- Run the application:
  - **`scalasca -analyze`** `mpirun -n 32 foo.exe`
    - Produces runtime summarization in directory: `scorep_<executable>_<ncores>_sum`
  - **`scalasca -analyze -t`** `mpirun -n 32 foo.exe`
    - Produces trace directory: `scorep_<executable>_<ncores>_trace`

Scalasca – Quick Start II

- Certain functions can be filtered during the instrumentation, e.g. BLAS subroutines, timing subroutines, etc;

```
scalasca -analyze -f scorep.filt mpirun -n 4 ./bt.A.4
```

- The filter file scorep.filt can contain the following:

```
SCOREP_REGION_NAMES_BEGIN EXCLUDE
timer*
rhs*
dgemm*
```

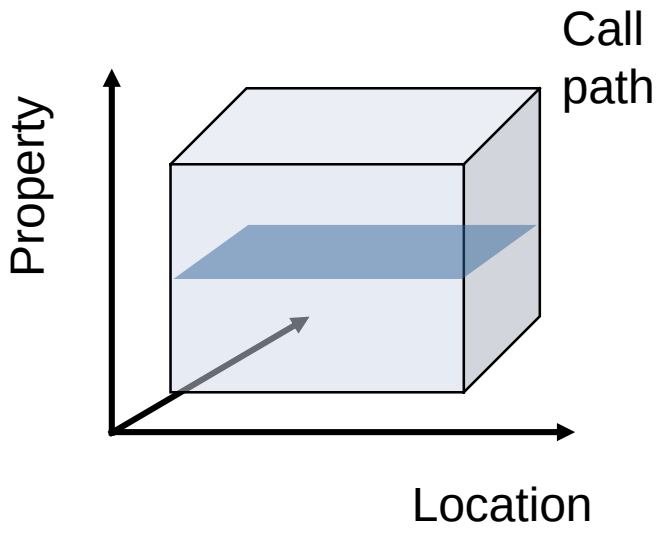
- This will reduce the trace file size. If using highly optimised third-party libraries, you may not want to profile this.

Scalasca – Quick Start III

- Summary analysis/report examination
  - `scalasca -examine -s <dir>`
    - Textual output `<dir>/scorep.score` - similar to gprof output
  - `scalasca -examine <dir>`
    - interactive exploration with Cube visualiser

Analysis Presentation and Exploration

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

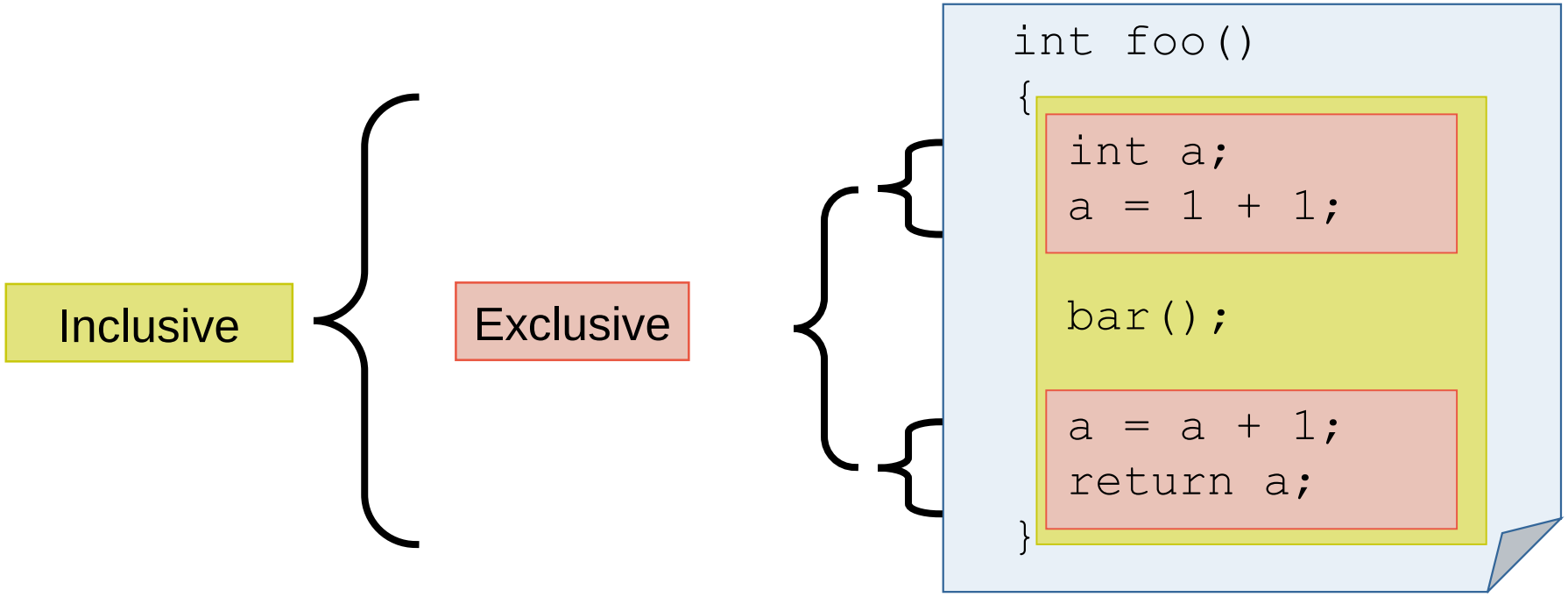


- Three coupled tree browsers

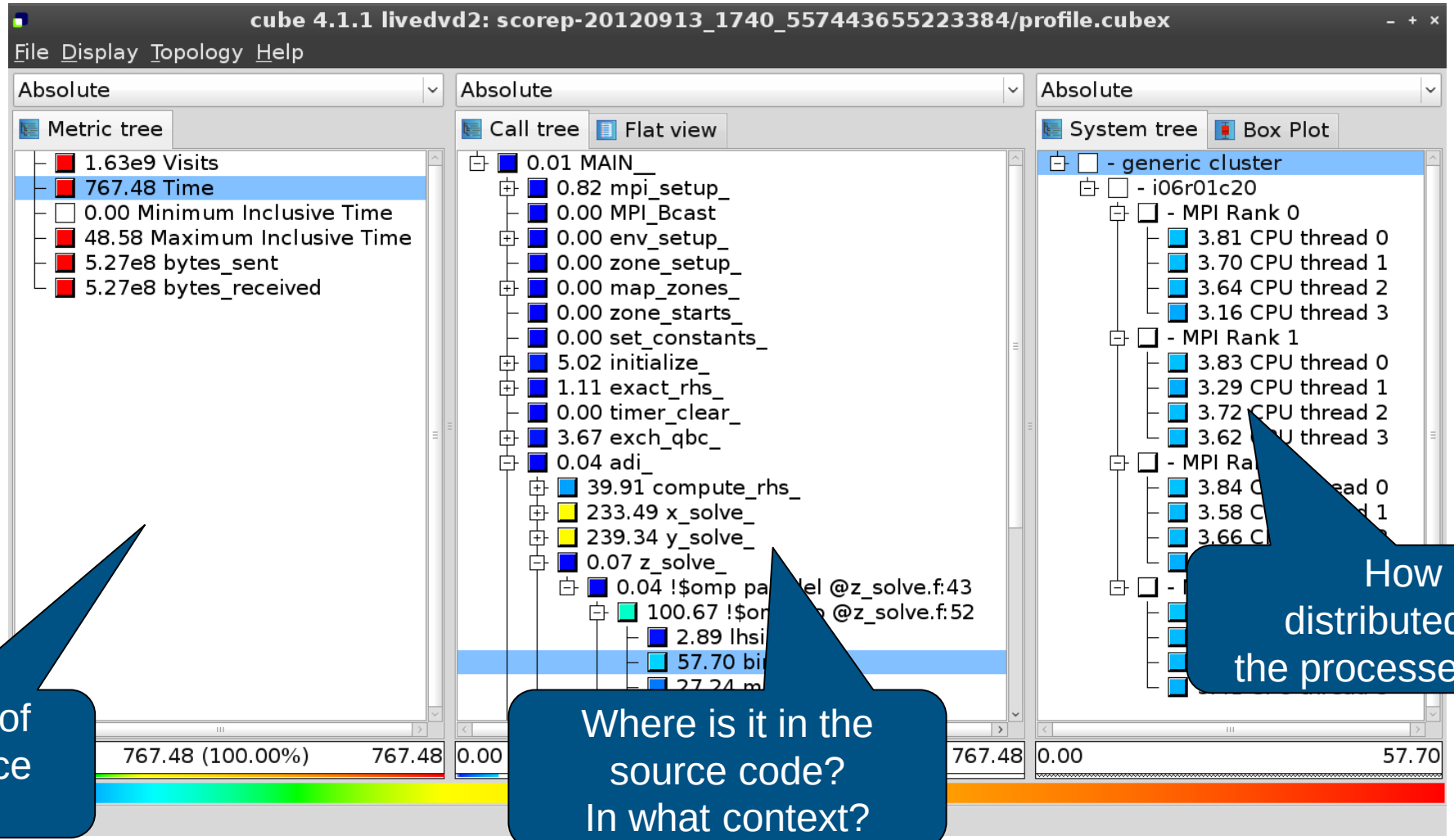
- Cube displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes

Inclusive vs. Exclusive Values

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



Analysis presentation

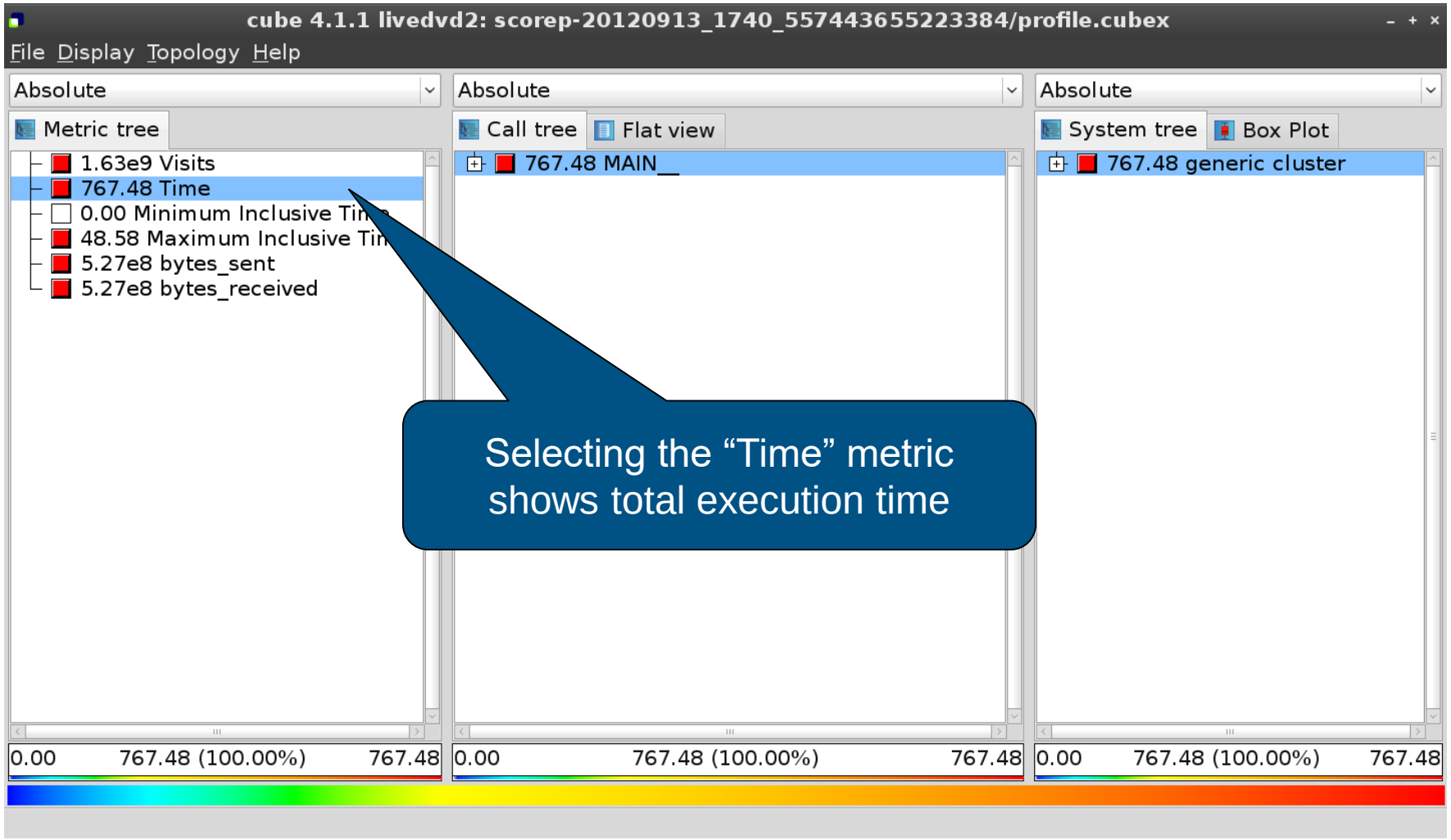


What kind of performance metric?

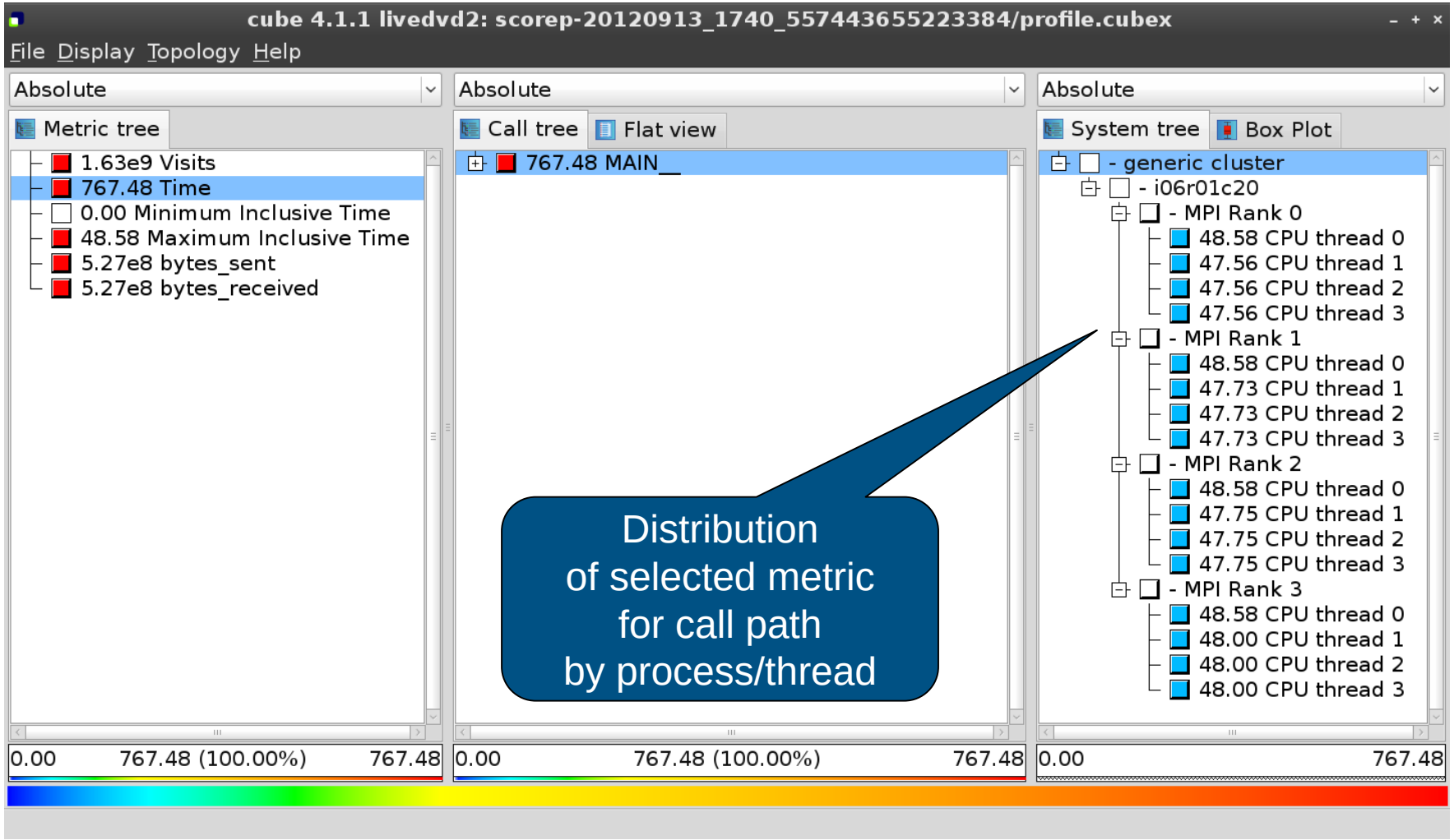
Where is it in the source code? In what context?

How is it distributed across the processes/threads?

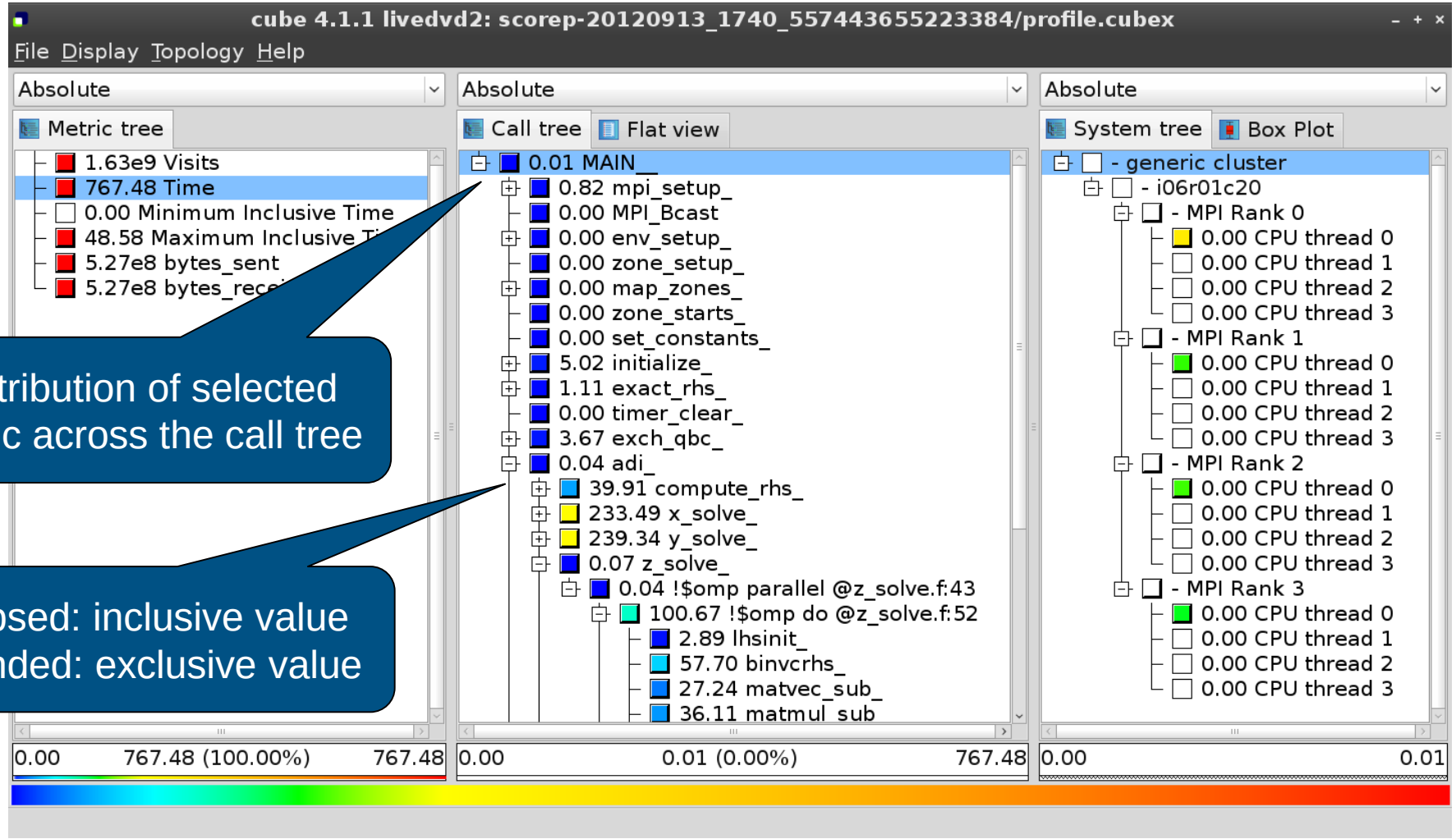
Metric selection



Expanding the System Tree



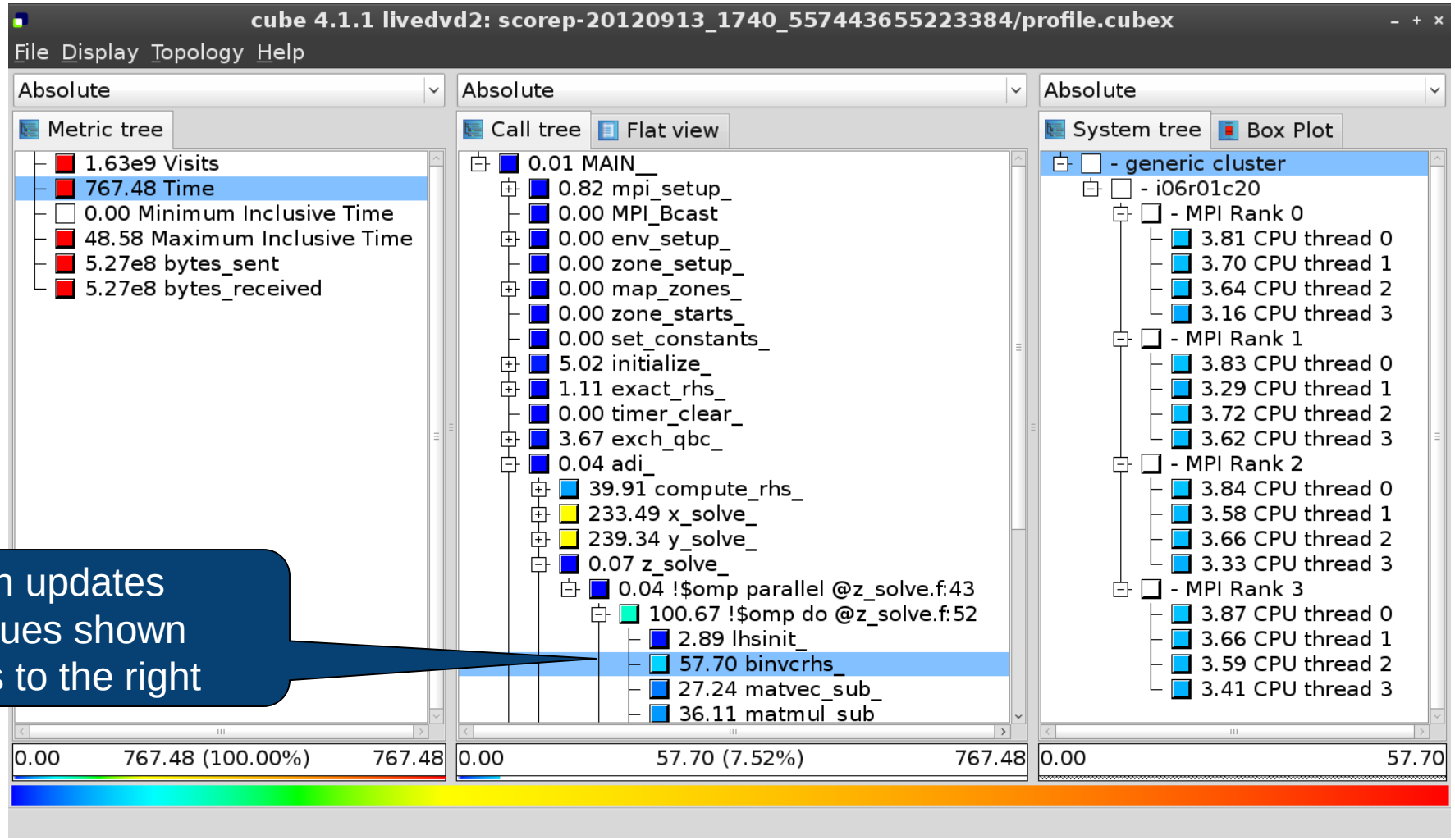
Expanding the Call Tree



Distribution of selected metric across the call tree

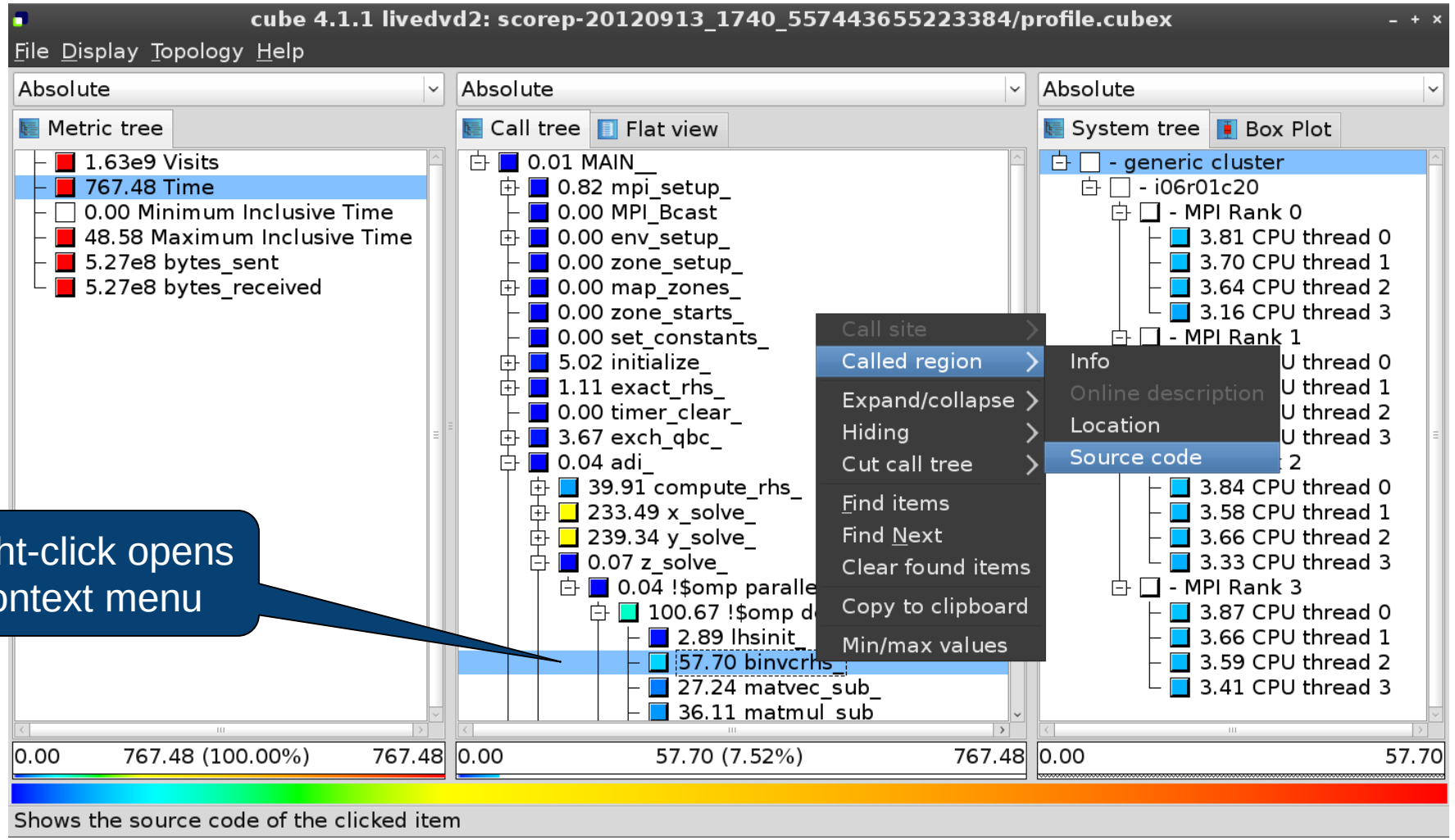
Collapsed: inclusive value
Expanded: exclusive value

Selecting a Call Path



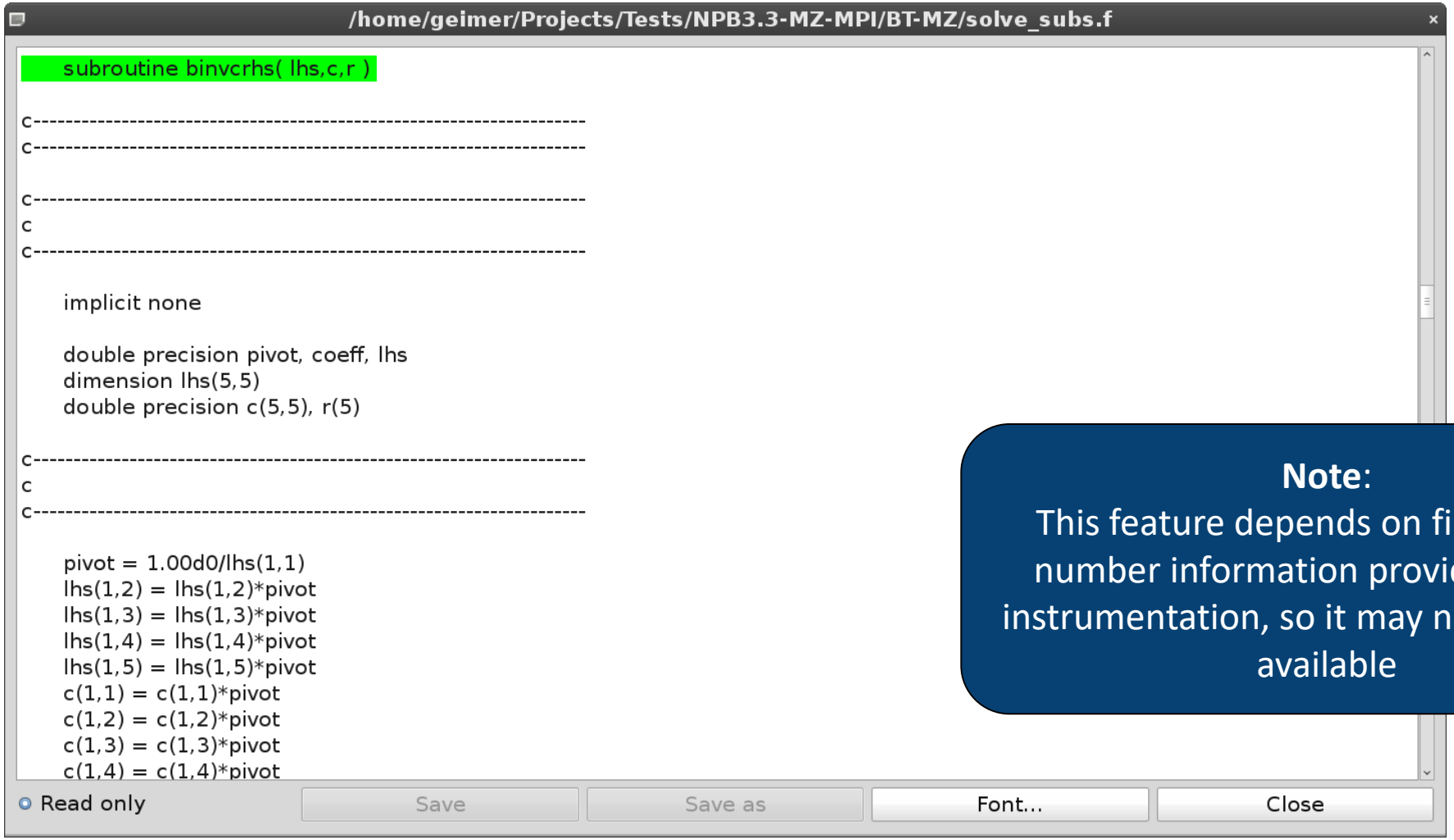
Selection updates metric values shown in columns to the right

Source Code View via Context Menu



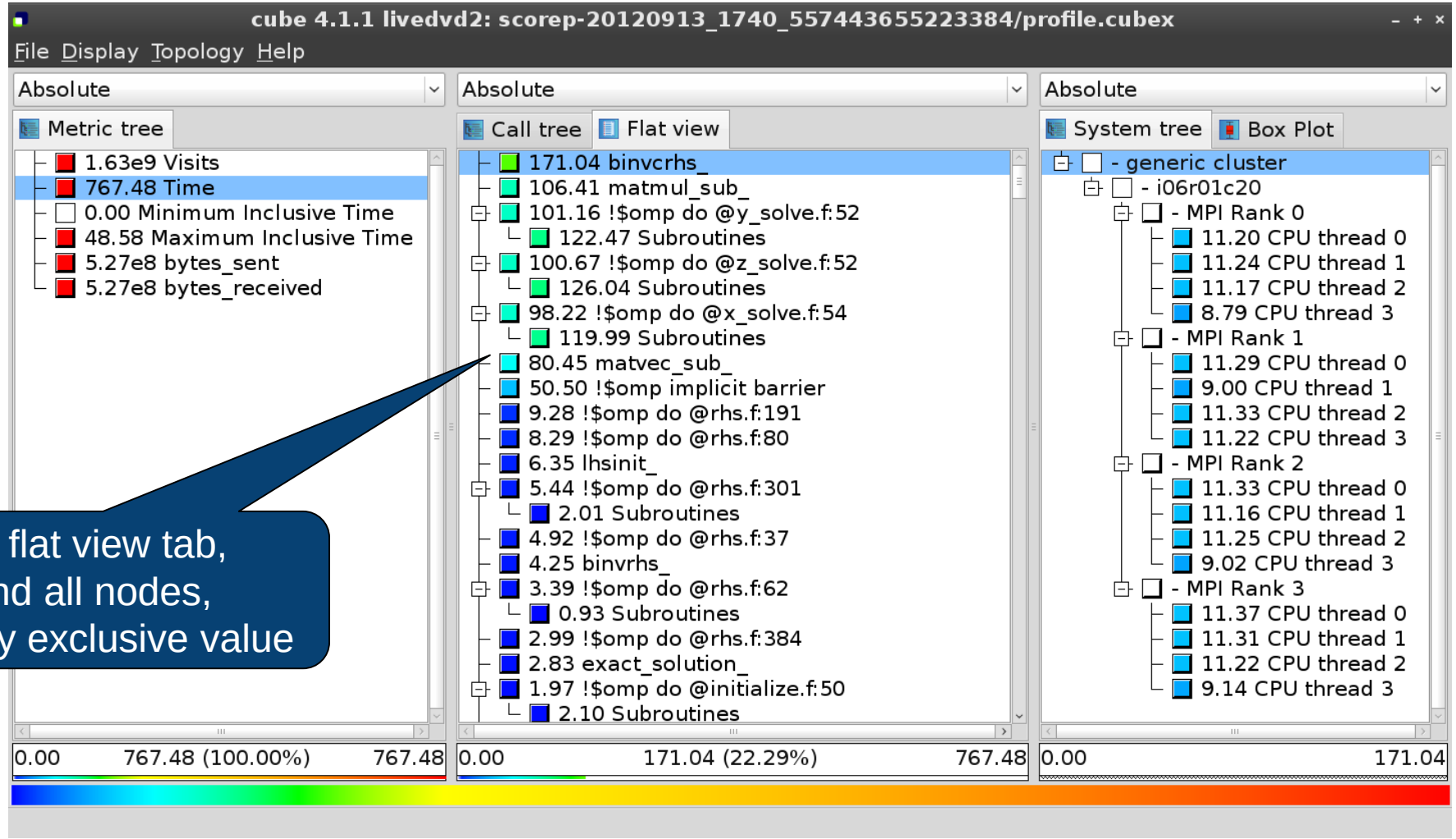
Right-click opens context menu

Source Code View



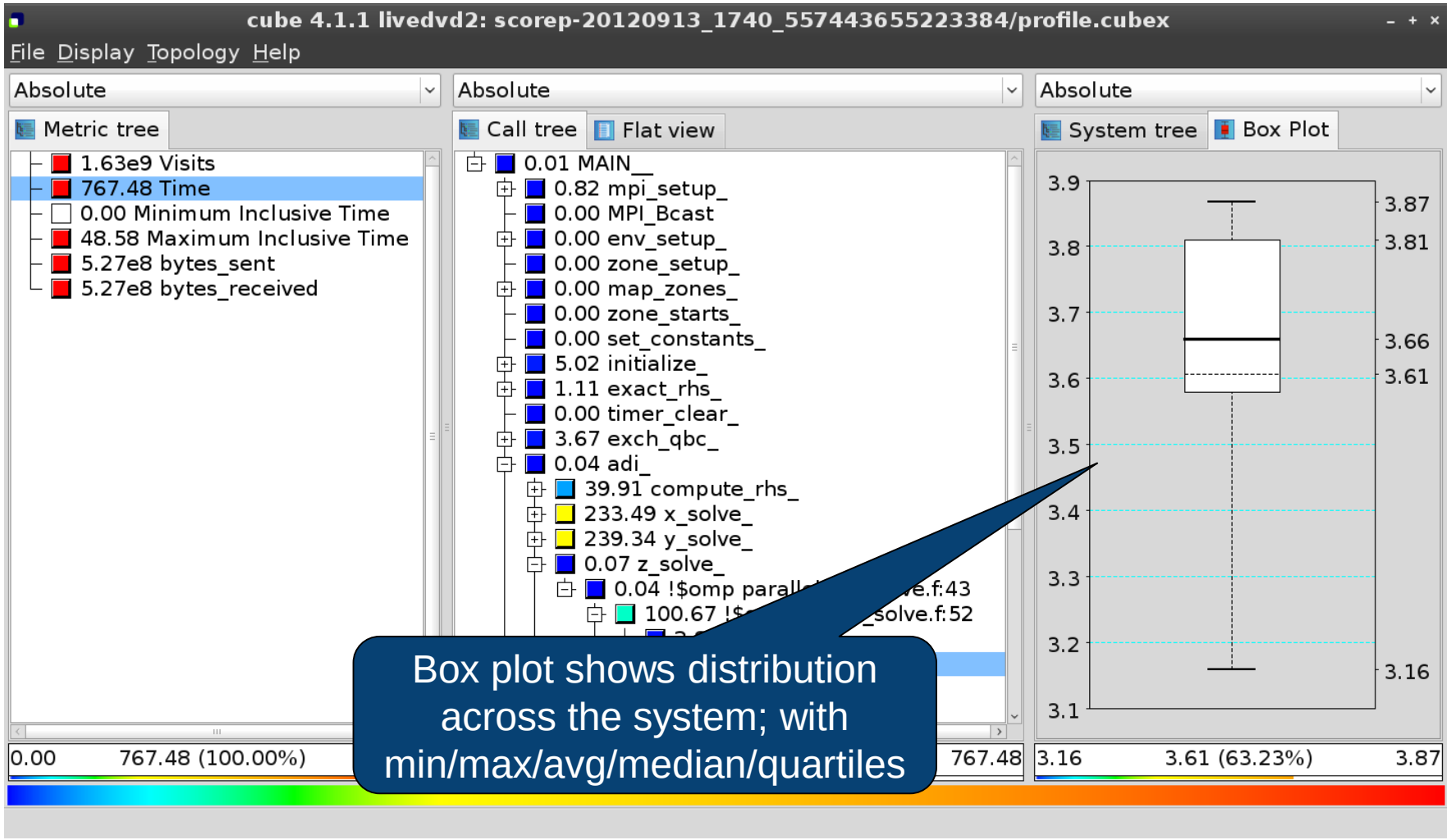
Note:
This feature depends on file and line number information provided by the instrumentation, so it may not always be available

Flat Profile View

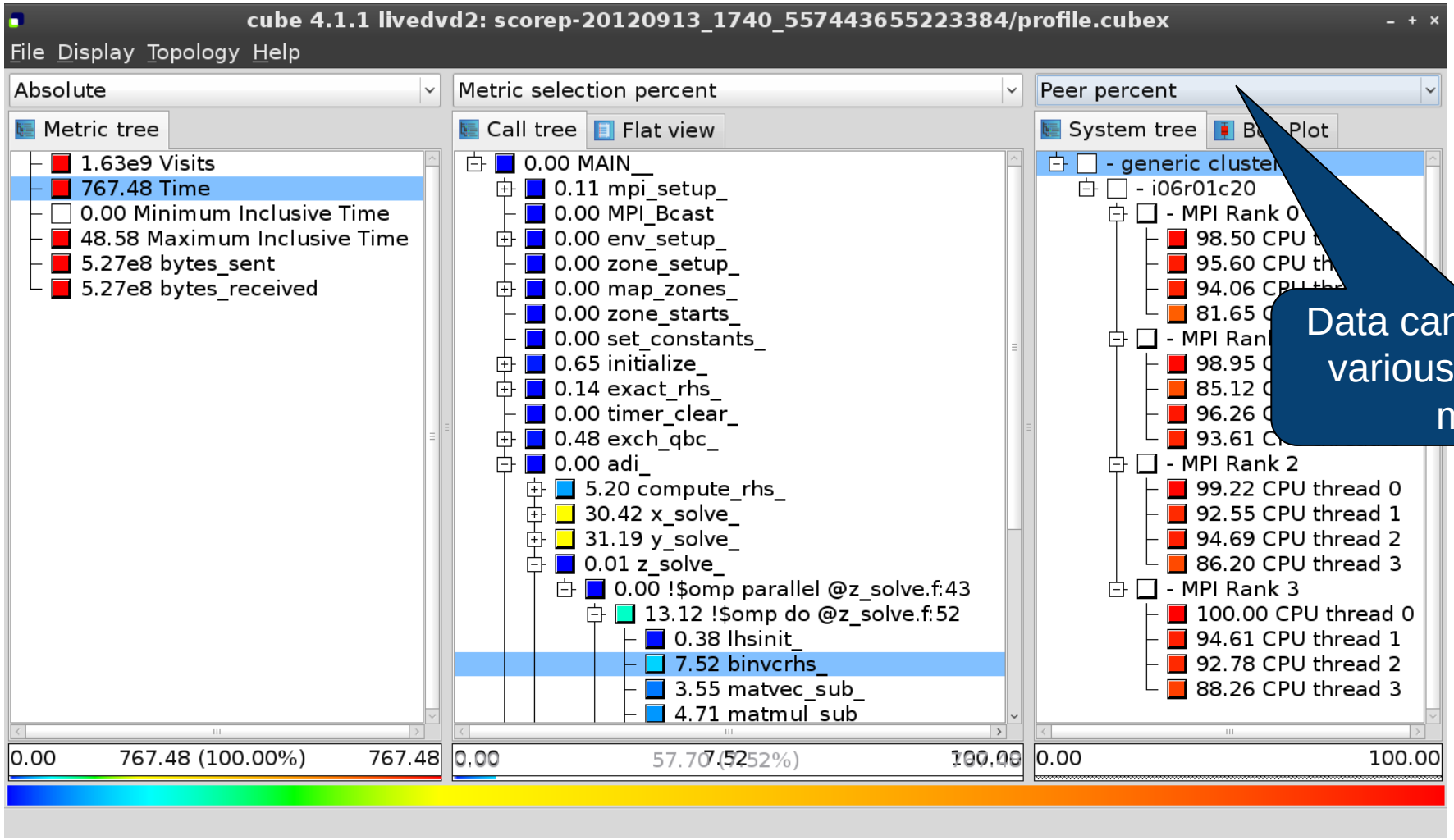


Select flat view tab,
expand all nodes,
and sort by exclusive value

Box Plot View



Alternative Display Modes

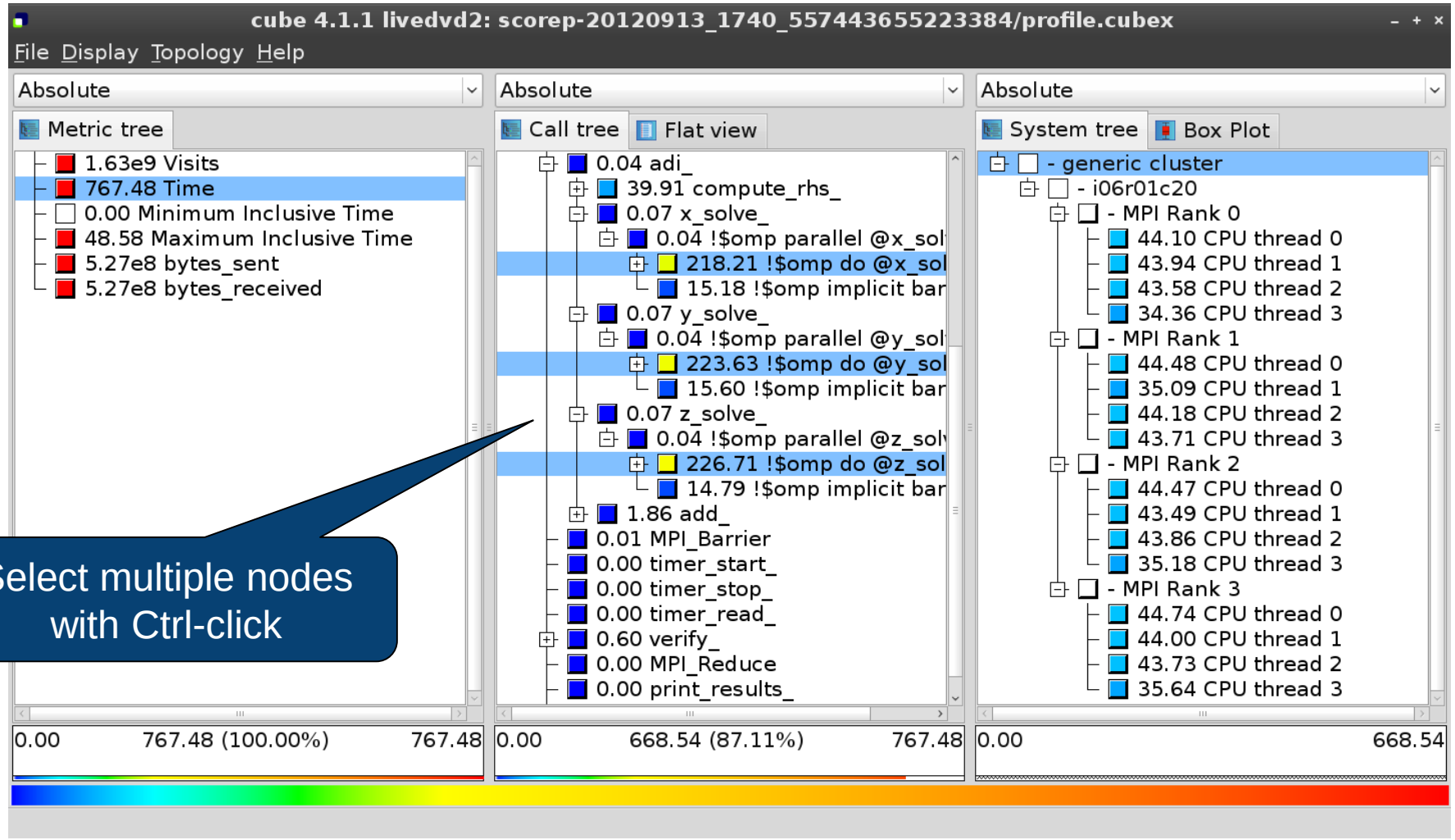


Data can be shown in various percentage modes

Important Display Modes

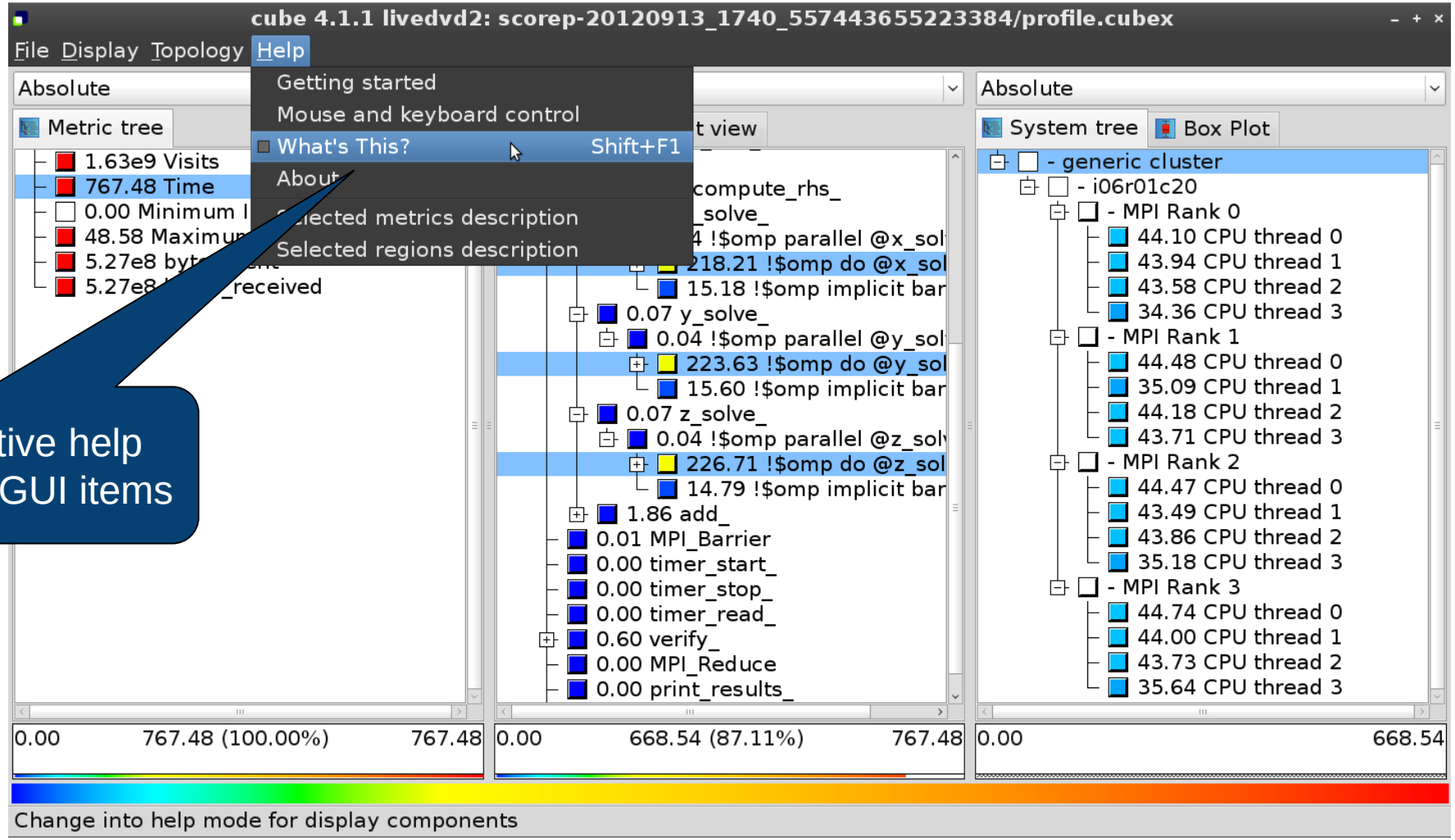
- Absolute
  - Absolute value shown in seconds/bytes/counts
- Selection percent
  - Value shown as percentage w.r.t. the selected node "on the left" (metric/call path)
- Peer percent (system tree only)
  - Value shown as percentage relative to the maximum peer value

Multiple Selection



Select multiple nodes with Ctrl-click

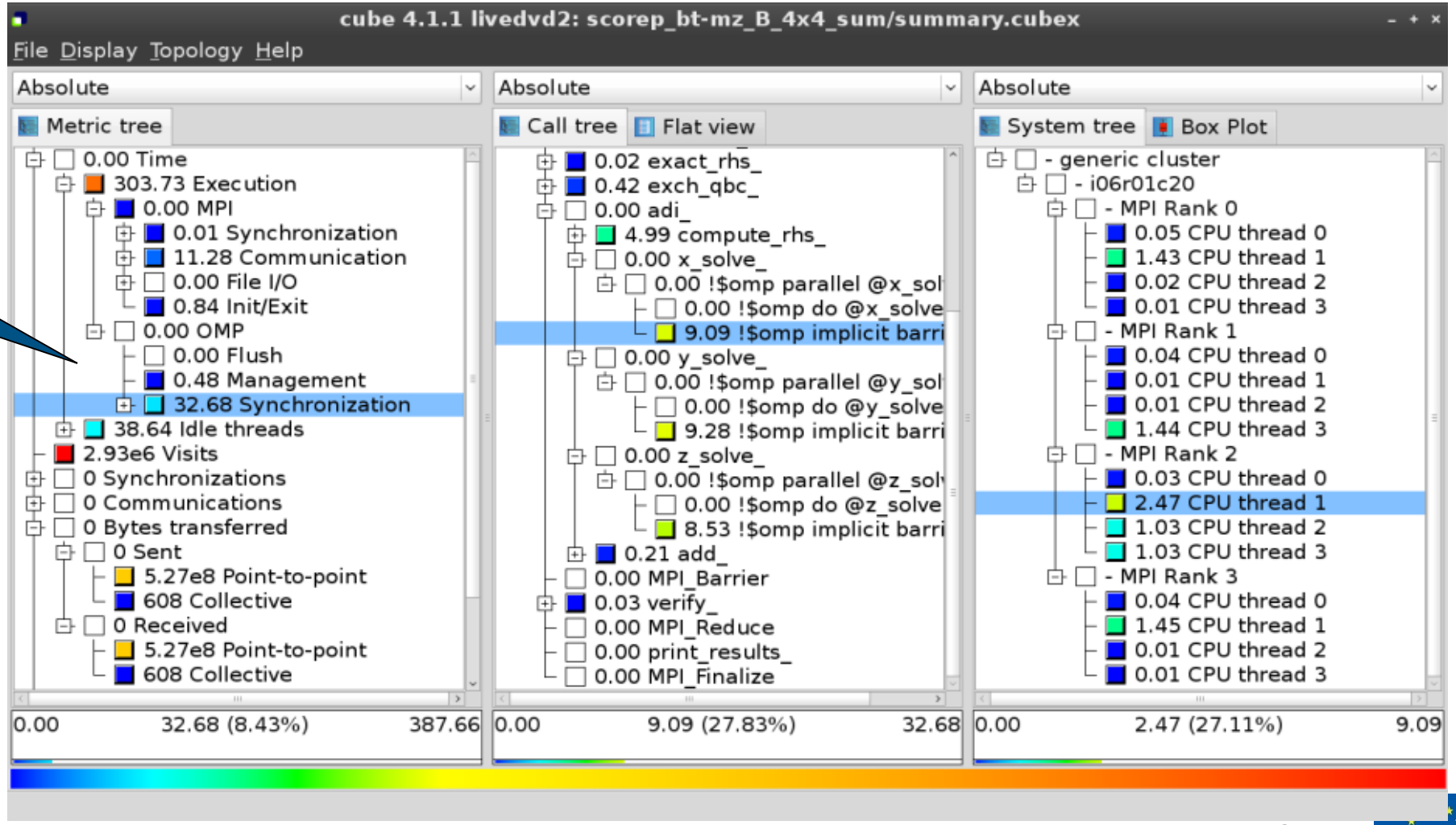
Context Sensitive Help



Context-sensitive help available for all GUI items

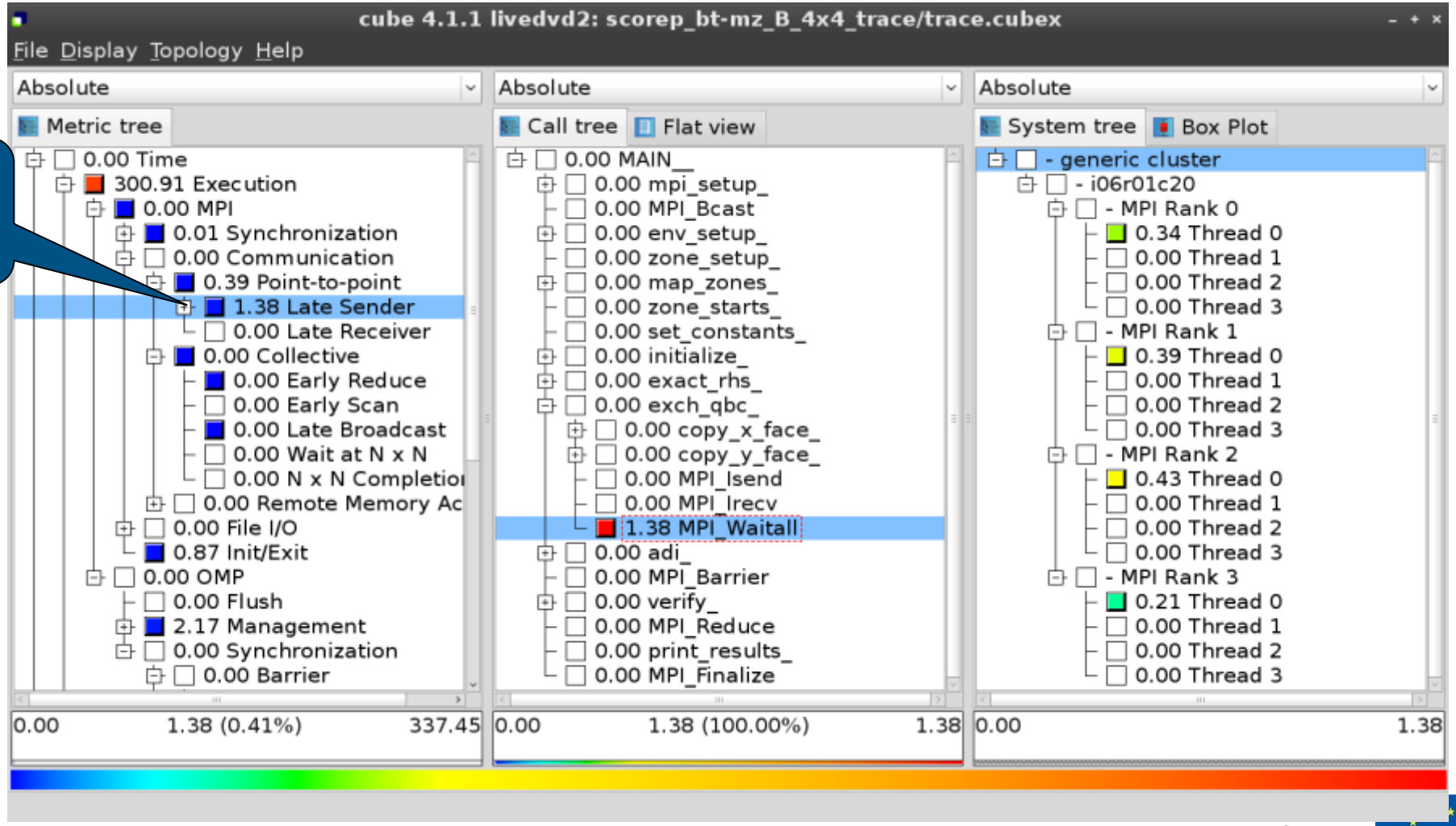
Summary Analysis Report

Split base metrics into more specific metrics



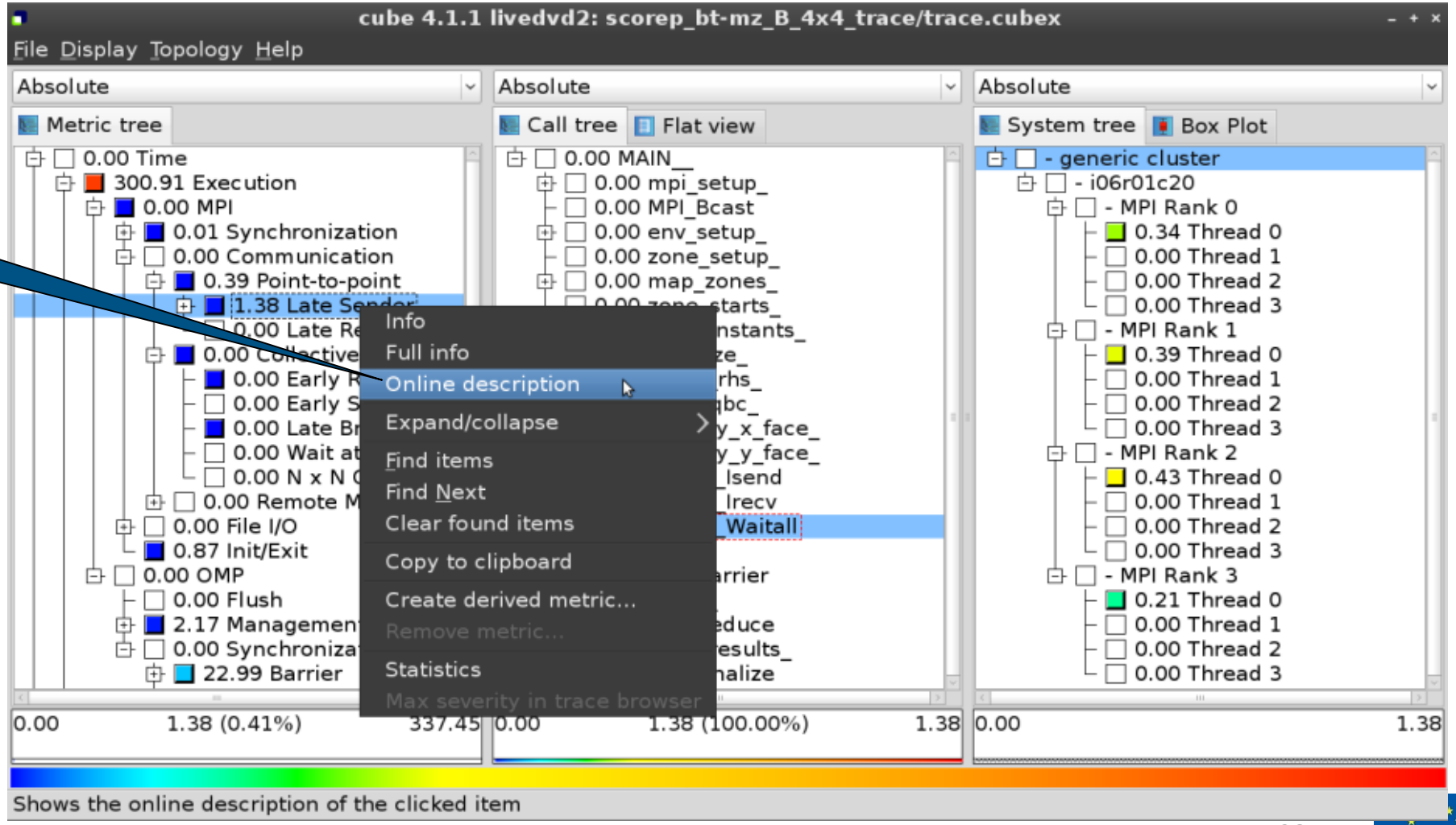
Trace Analysis Report

Additional trace-based metrics in metric hierarchy



Online Metric Description

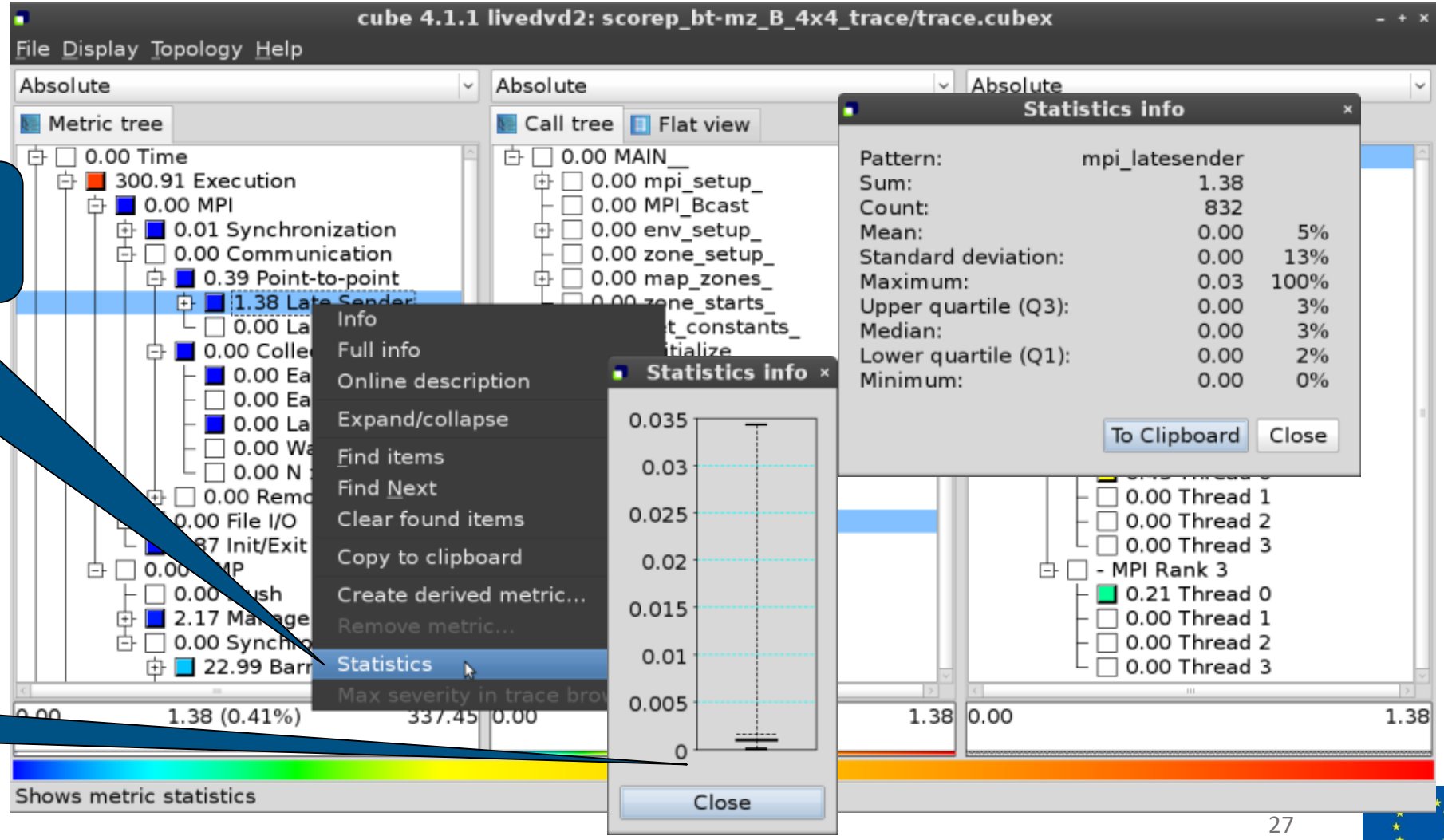
Access online metric description via context menu



Pattern Instance Statistics

Access pattern instance statistics via context menu

Click to get statistics details



Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.