



Analysis Report Examination with Cube

Wadud Miah. Numerical Algorithms Group
Thanks to Markus Geimer of JSC

EU H2020 Centre of Excellence (CoE)



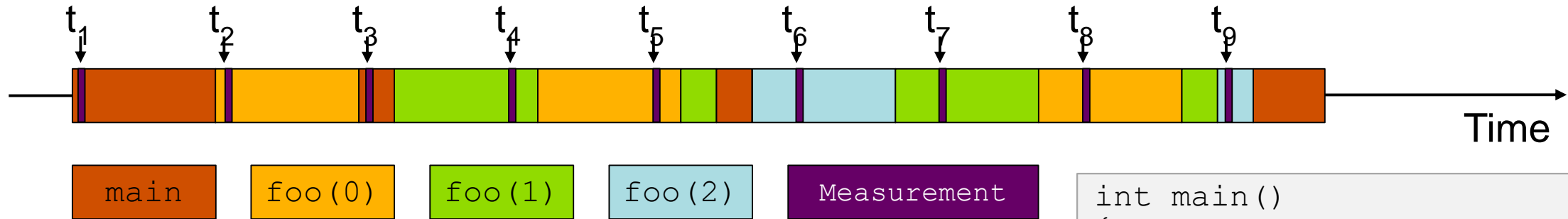
1 October 2015 – 31 March 2018

Grant Agreement No 676553

- 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



- 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

```
int main()
{
    int i;

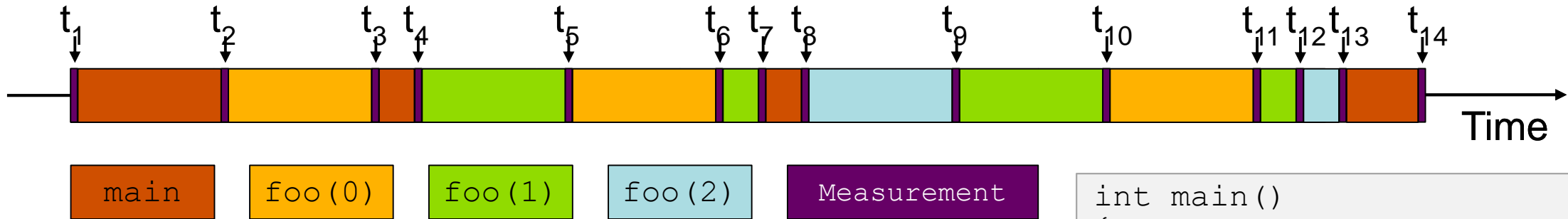
    for (i=0; i < 3; i++)
        foo(i);

    return 0;
}

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



Instrumentation



- 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

```
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");
}
```



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
<code>mpicc -g -c foo.c</code>	<code>scorep mpicc -g -c foo.c</code>
<code>mpif90 -g -openmp -o bar bar.f90</code>	<code>scorep mpif90 -g -openmp -o bar bar.f90</code>

- 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

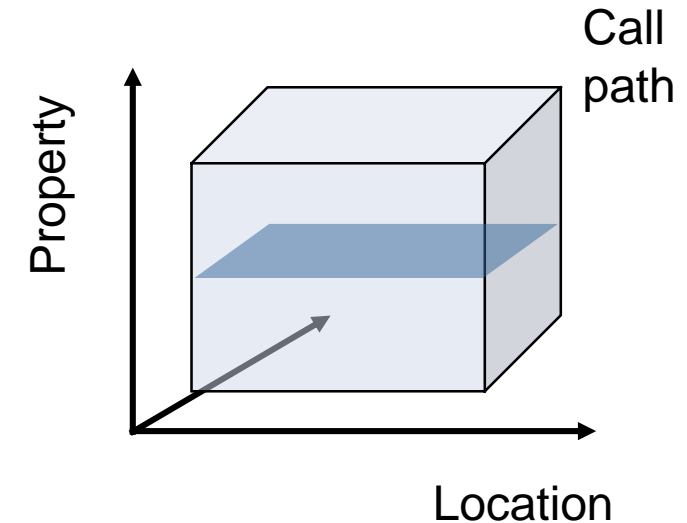


- Summary analysis/report examination
 - `scalasca -examine -s <dir>`
 - Textual output `<dir>/scorep.score` - similar to gprof output
 - `scalasca -examine <dir>`
 - interactive exploration with Cube visualiser





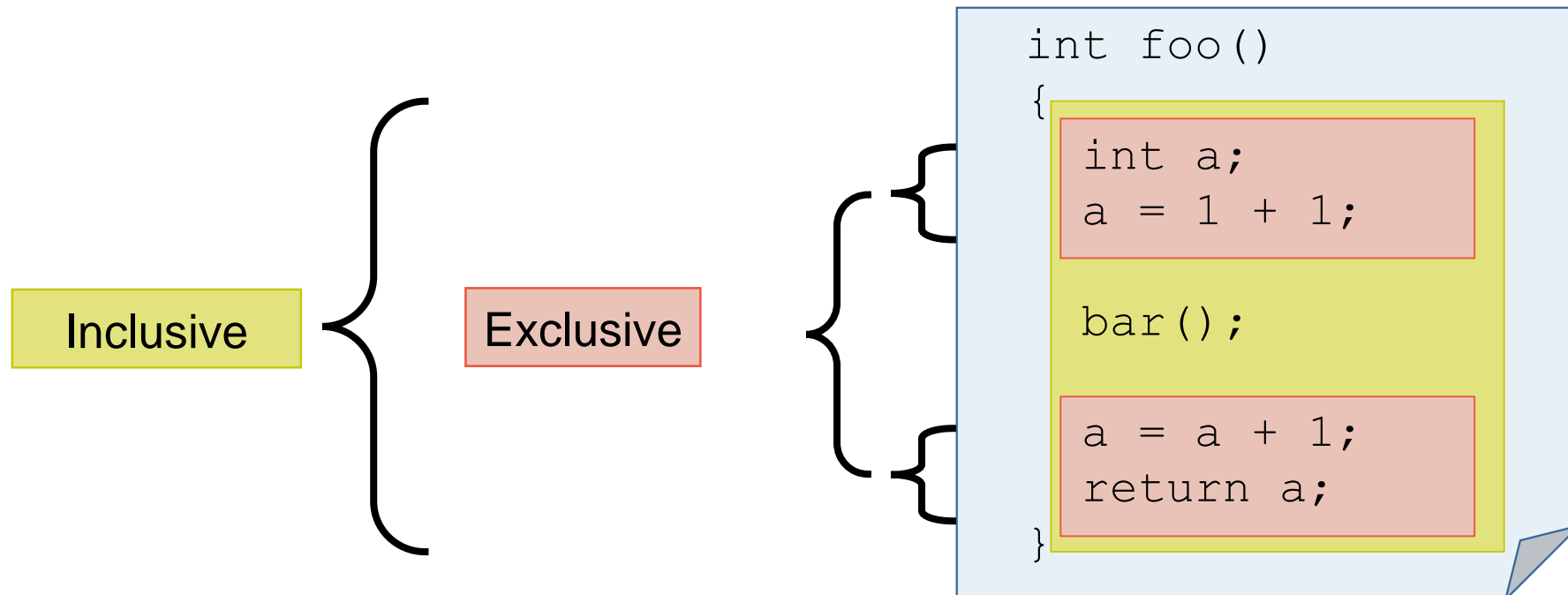
- Representation of values (severity matrix) on three hierarchical axes
 - Performance property (metric)
 - Call path (program location)
 - System location (process/thread)
- Three coupled tree browsers
- Cube displays severities
 - As value: for precise comparison
 - As colour: for easy identification of hotspots
 - Inclusive value when closed & exclusive value when expanded
 - Customizable via display modes



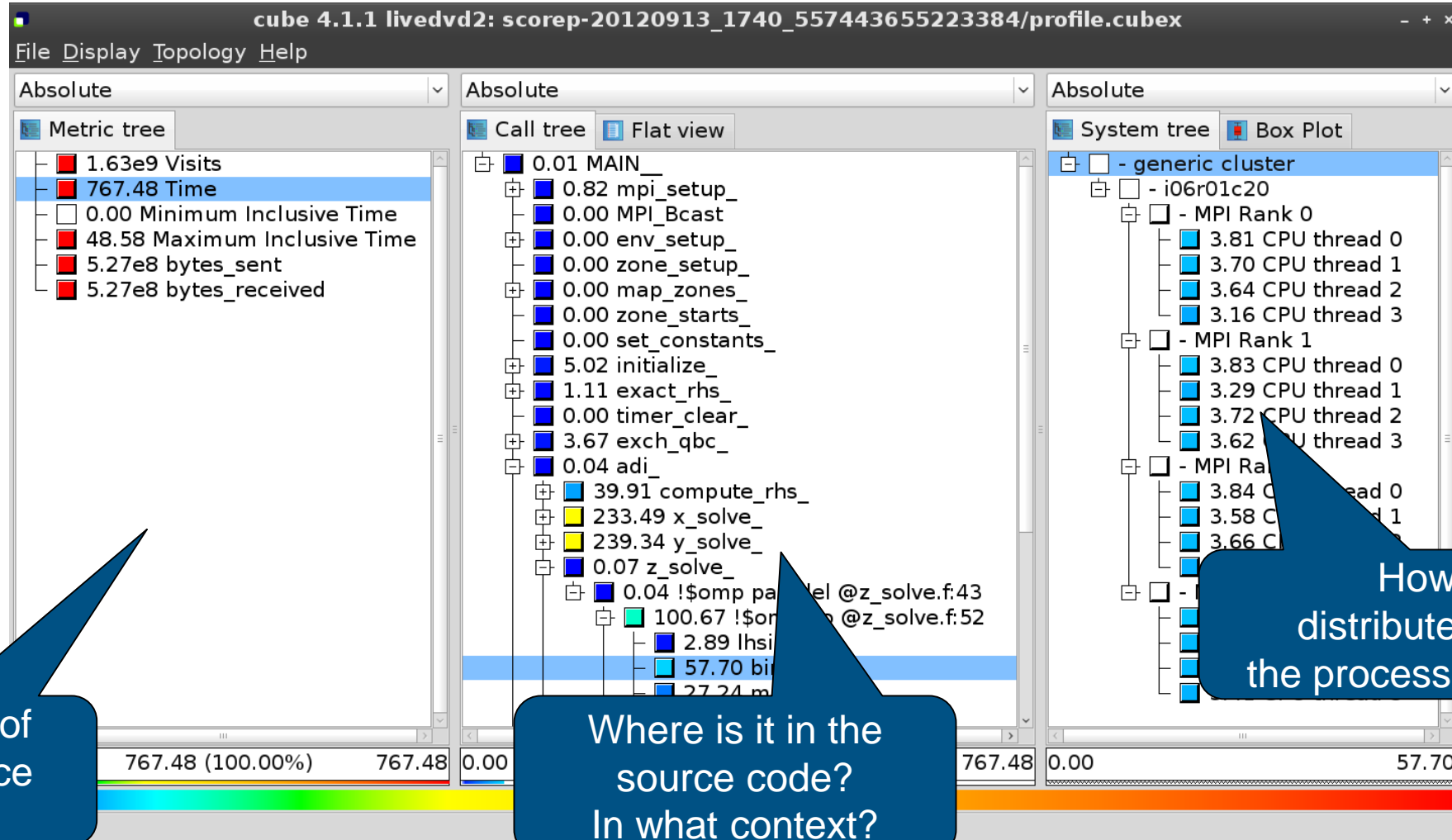
Inclusive vs. Exclusive Values



- 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



cube 4.1.1 livedvd2: scorep-20120913_1740_557443655223384/profile.cubex

File Display Topology Help

Absolute Absolute Absolute

Metric tree Call tree Flat view System tree Box Plot

- 1.63e9 Visits
- 767.48 Time**
- 0.00 Minimum Inclusive Time
- 48.58 Maximum Inclusive Time
- 5.27e8 bytes_sent
- 5.27e8 bytes_received

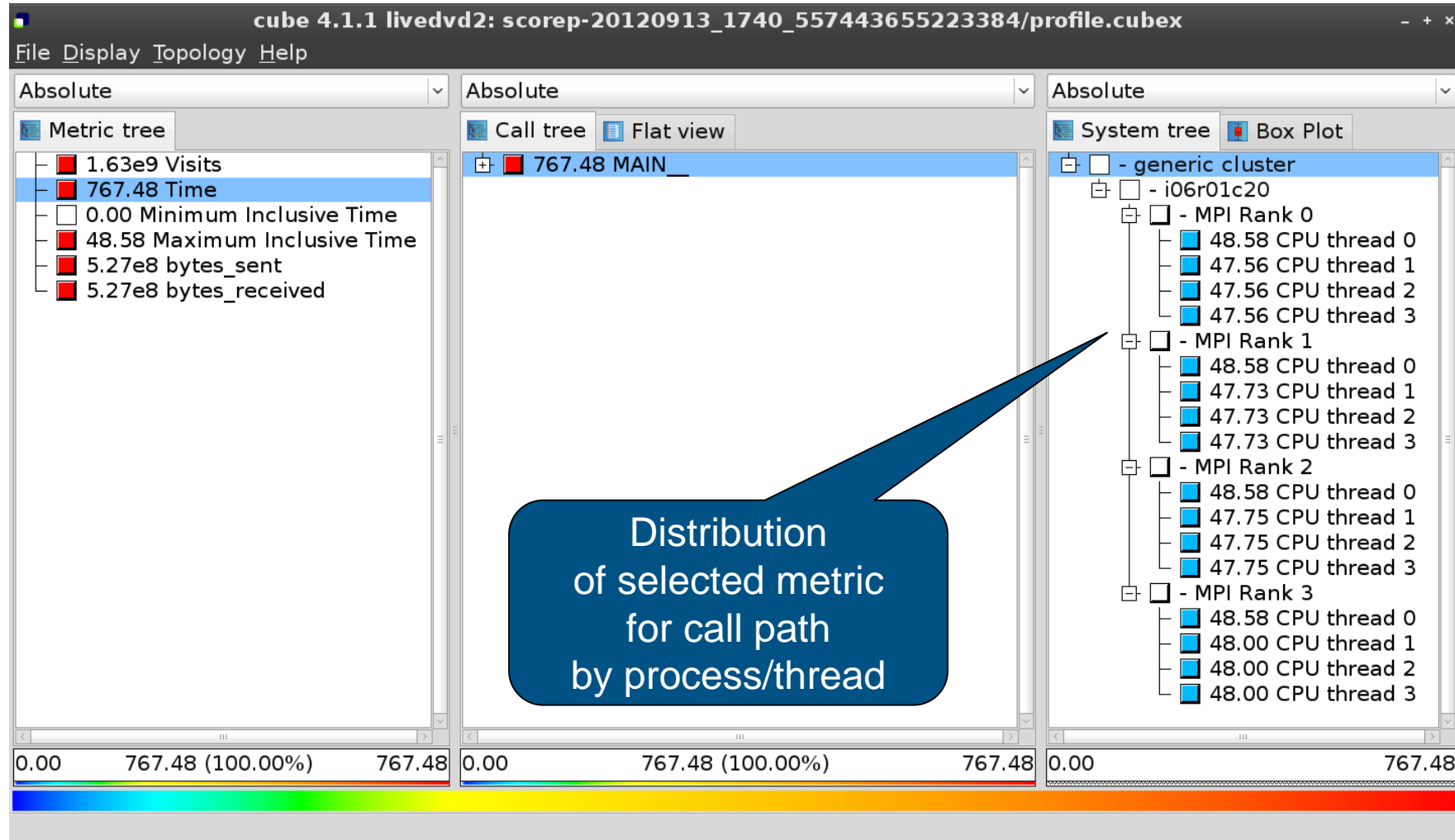
767.48 MAIN_ 767.48 generic cluster

0.00 767.48 (100.00%) 767.48 0.00 767.48 (100.00%) 767.48 0.00 767.48 (100.00%) 767.48

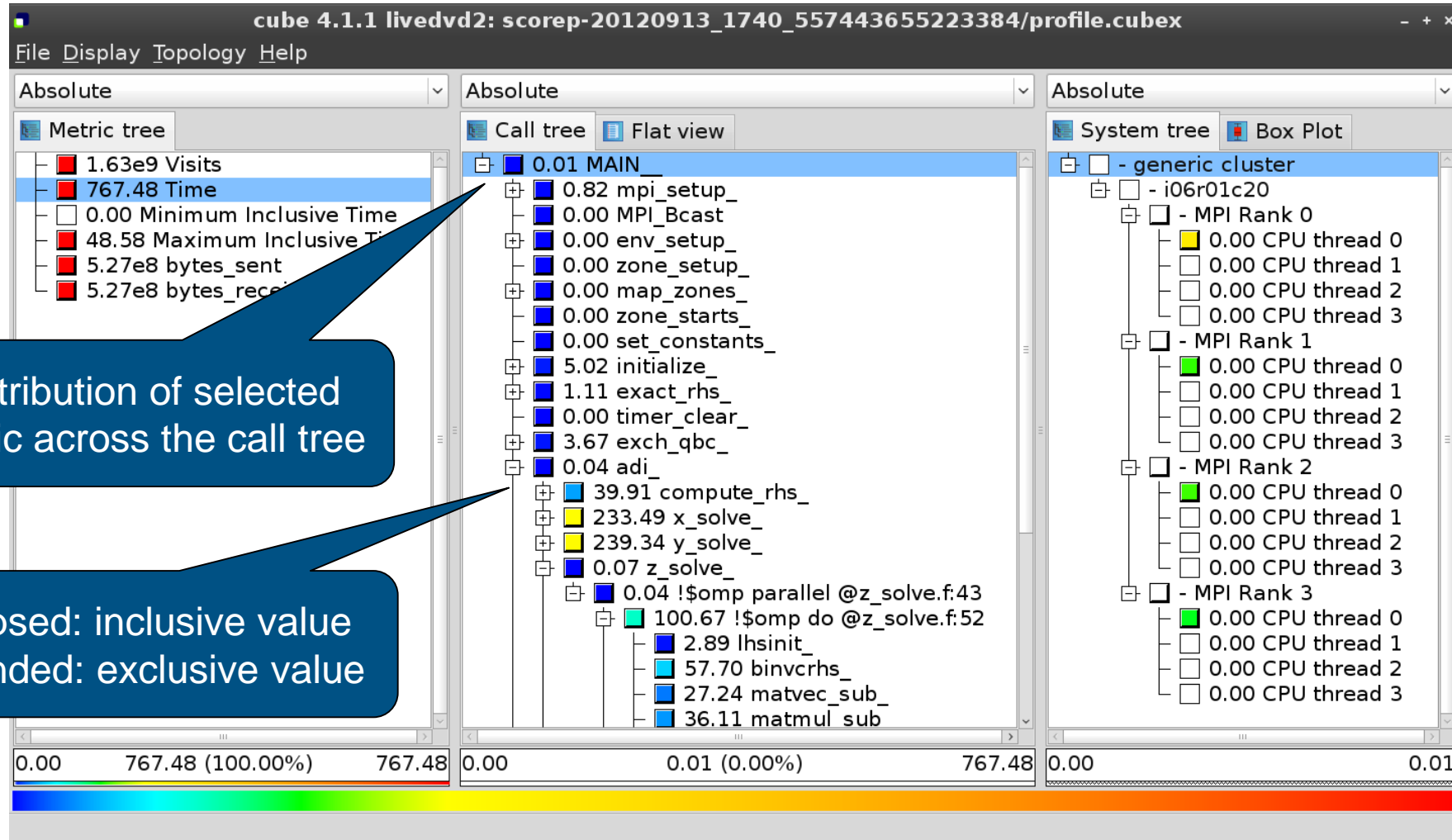
Selecting the "Time" metric shows total execution time



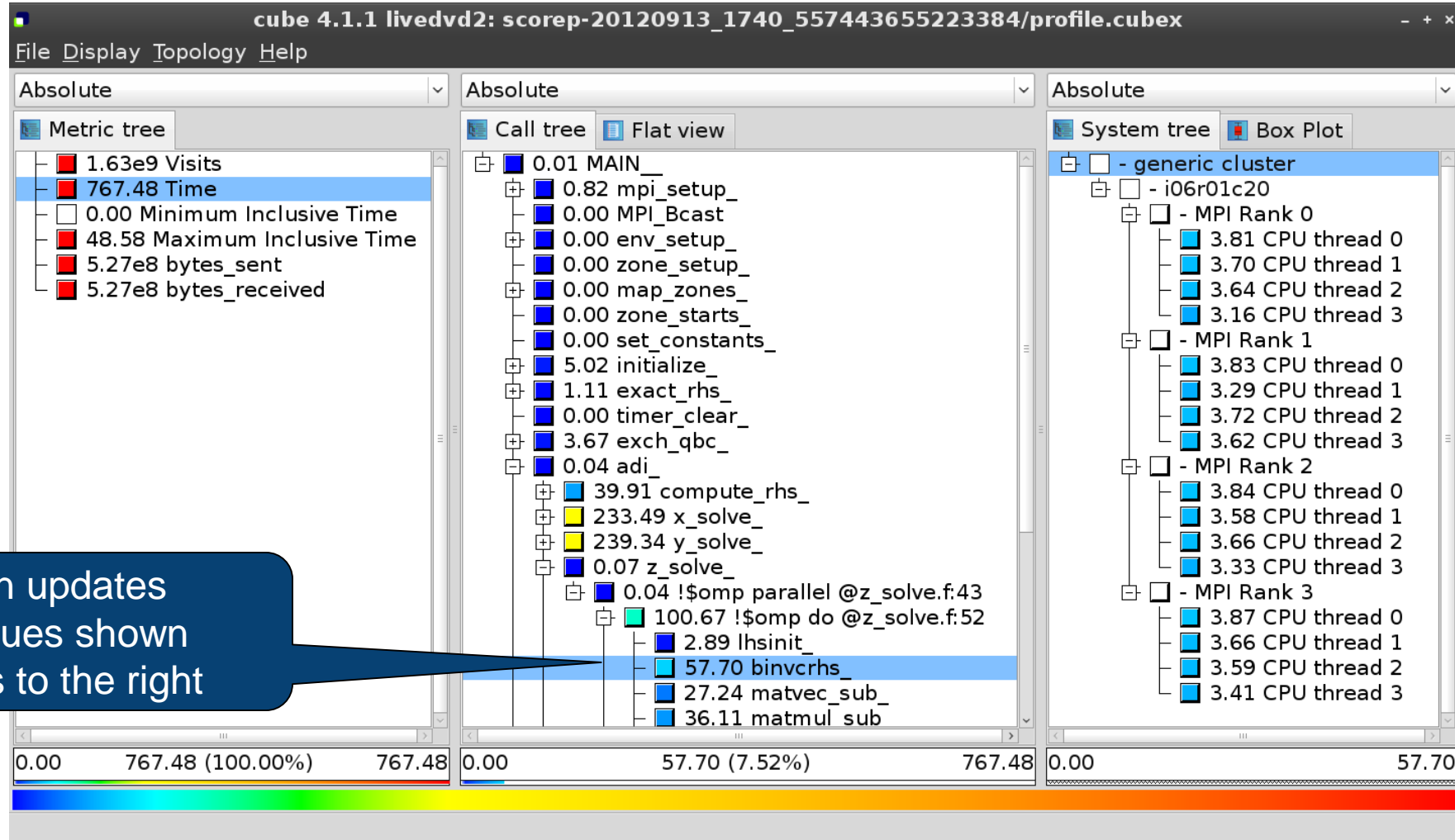
Expanding the System Tree



Expanding the Call Tree



Selecting a Call Path



Selection updates metric values shown in columns to the right



Source Code View via Context Menu



The screenshot shows the 'cube 4.1.1' application interface. It features three main panels: 'Metric tree', 'Call tree', and 'System tree'. The 'Call tree' panel is active, showing a hierarchical view of function calls. A context menu is open over the 'binvcrhs_' function, listing options such as 'Info', 'Online description', 'Location', and 'Source code'. A blue callout box with a white border points to the context menu with the text 'Right-click opens context menu'. The status bar at the bottom indicates the current view is showing the source code of the clicked item.

Right-click opens context menu

Shows the source code of the clicked item



Source Code View



```
subroutine binvrchs( lhs,c,r )
C-----
C-----
C-----
C
C-----

implicit none

double precision pivot, coeff, lhs
dimension lhs(5,5)
double precision c(5,5), r(5)

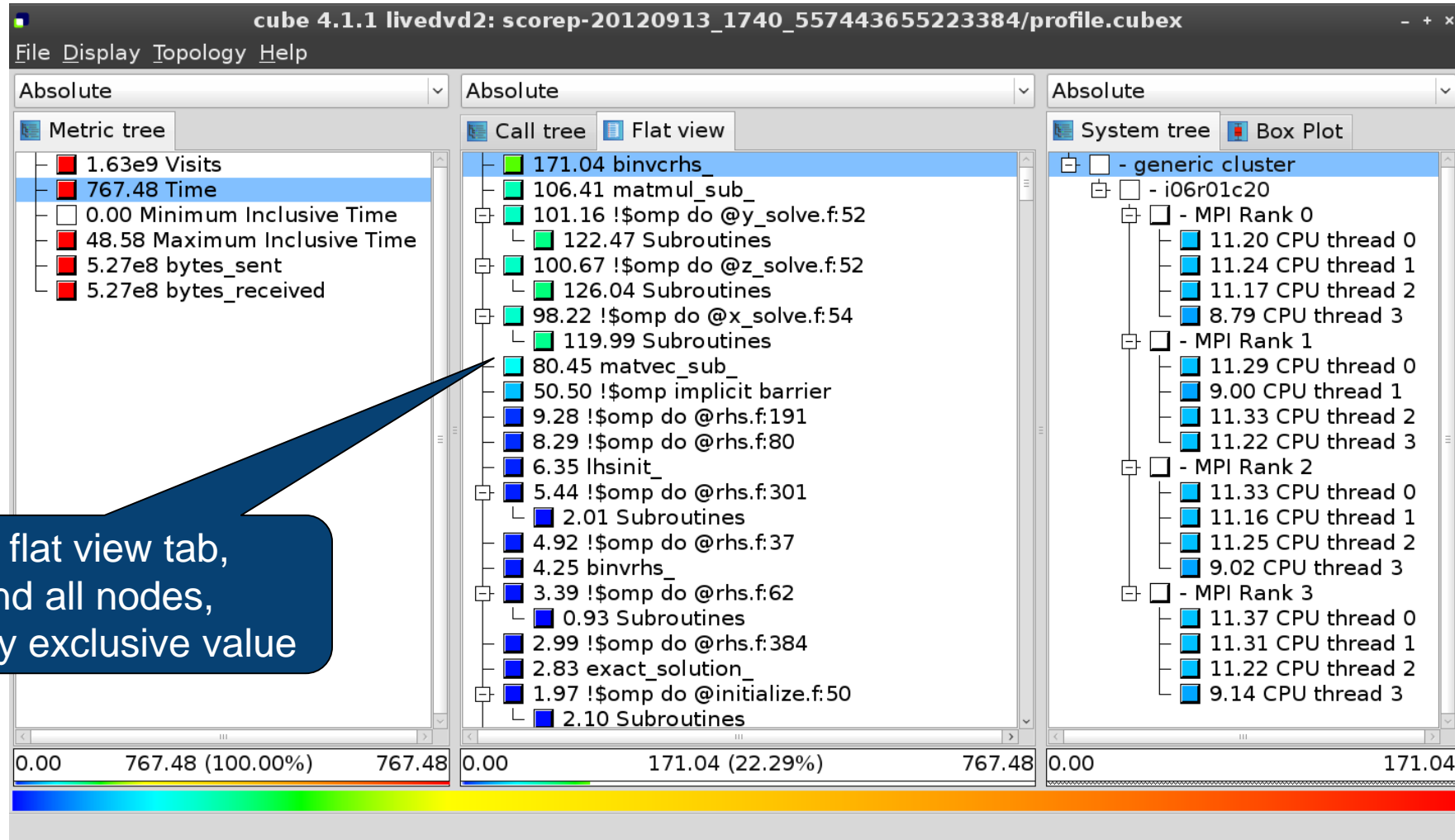
C-----
C
C-----

pivot = 1.00d0/lhs(1,1)
lhs(1,2) = lhs(1,2)*pivot
lhs(1,3) = lhs(1,3)*pivot
lhs(1,4) = lhs(1,4)*pivot
lhs(1,5) = lhs(1,5)*pivot
c(1,1) = c(1,1)*pivot
c(1,2) = c(1,2)*pivot
c(1,3) = c(1,3)*pivot
c(1,4) = c(1,4)*pivot
```

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



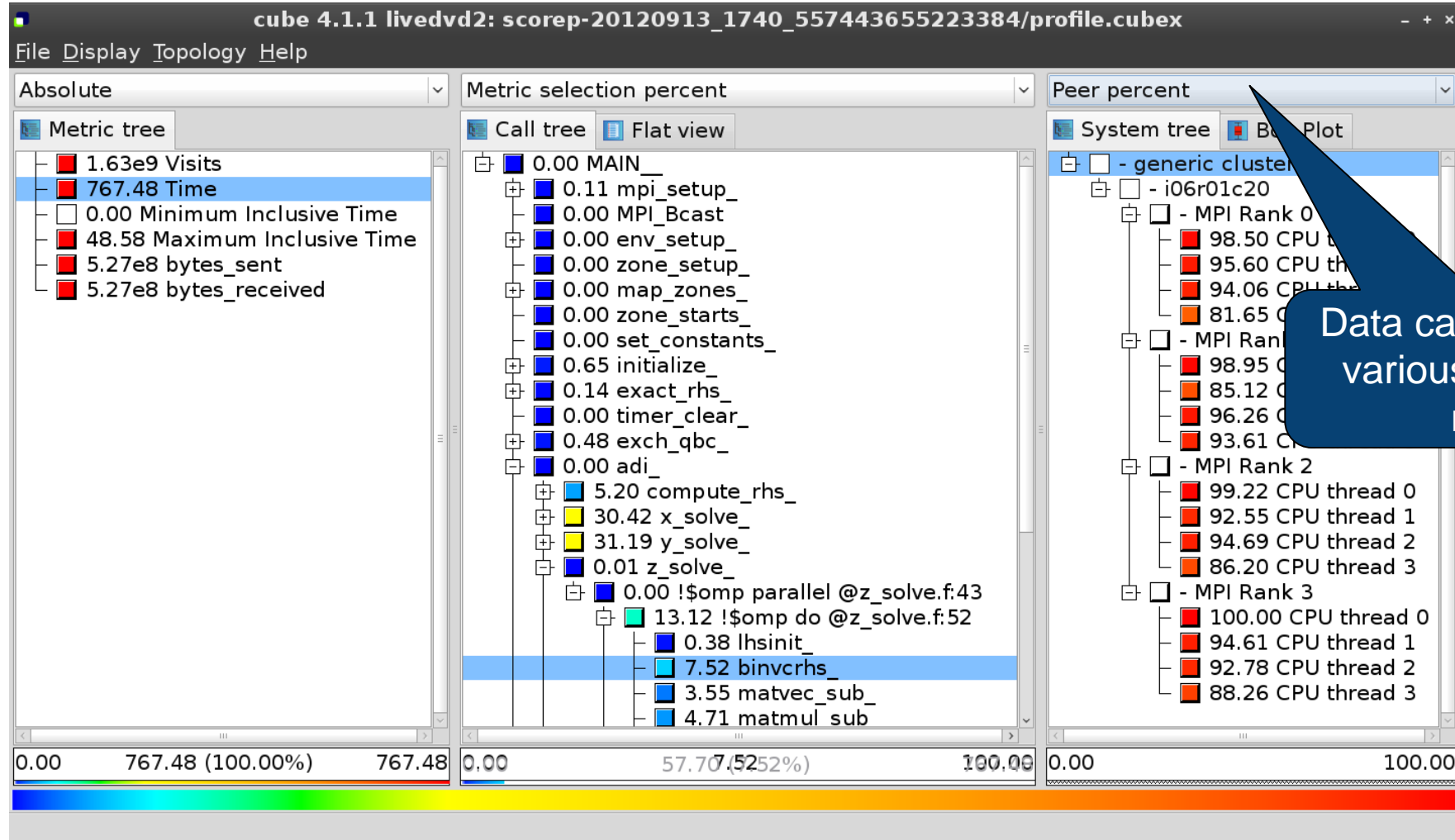
Flat Profile View



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



Alternative Display Modes



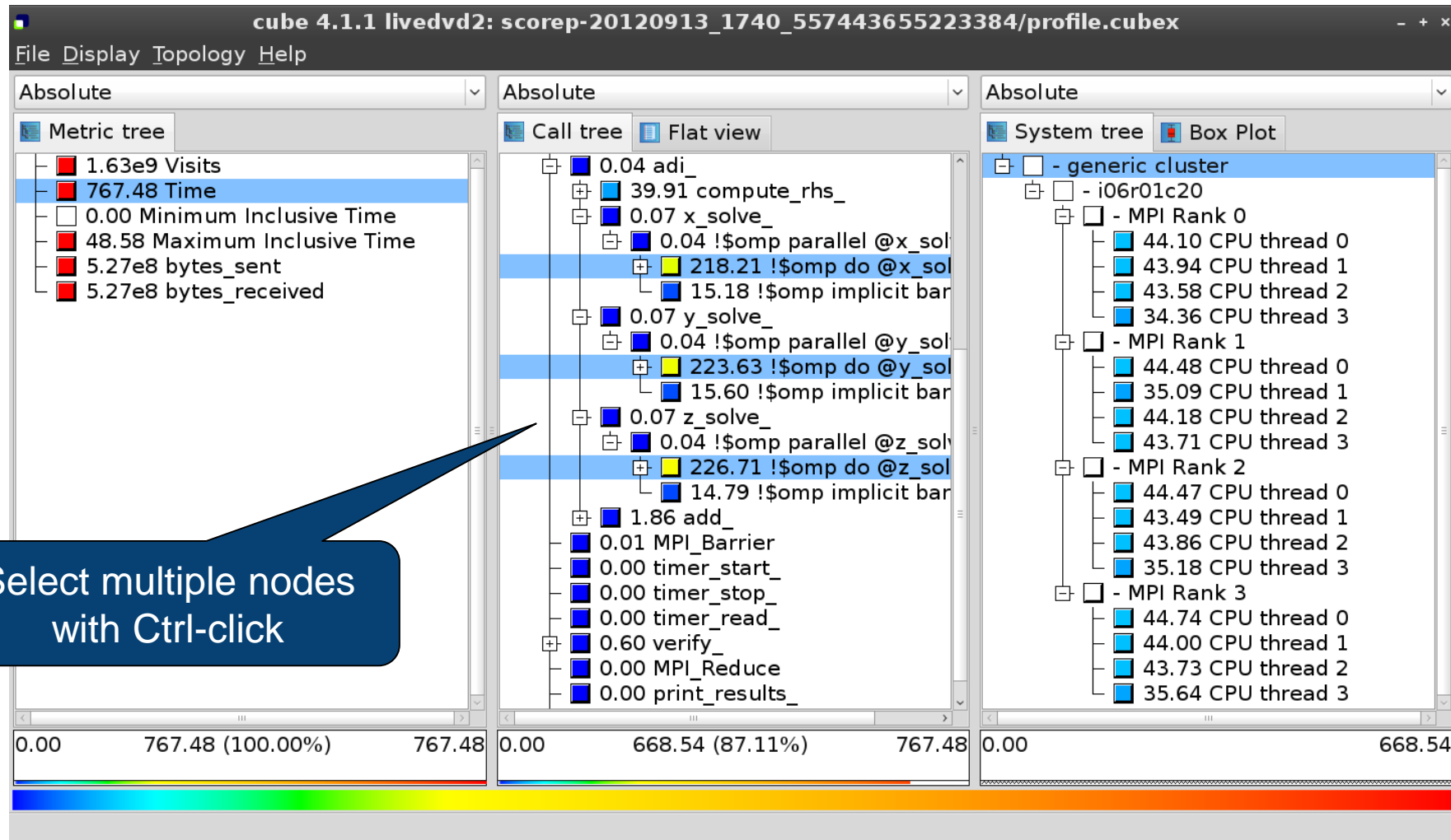
Important Display Modes



- 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



The screenshot shows the 'cube 4.1.1' application window. The 'Help' menu is open, with 'What's This?' selected. A callout box points to the 'What's This?' option, containing the text: 'Context-sensitive help available for all GUI items'. The main window displays a 'Metric tree' on the left, a central 'System tree' showing a hierarchical view of MPI ranks and CPU threads, and a 'Box Plot' on the right. The 'Metric tree' shows a selected metric '767.48 Time'. The 'System tree' shows a selected node '223.63 !\$omp do @y_solve_'. The 'Box Plot' shows a selected node '44.10 CPU thread 0'. The status bar at the bottom indicates 'Change into help mode for display components'.

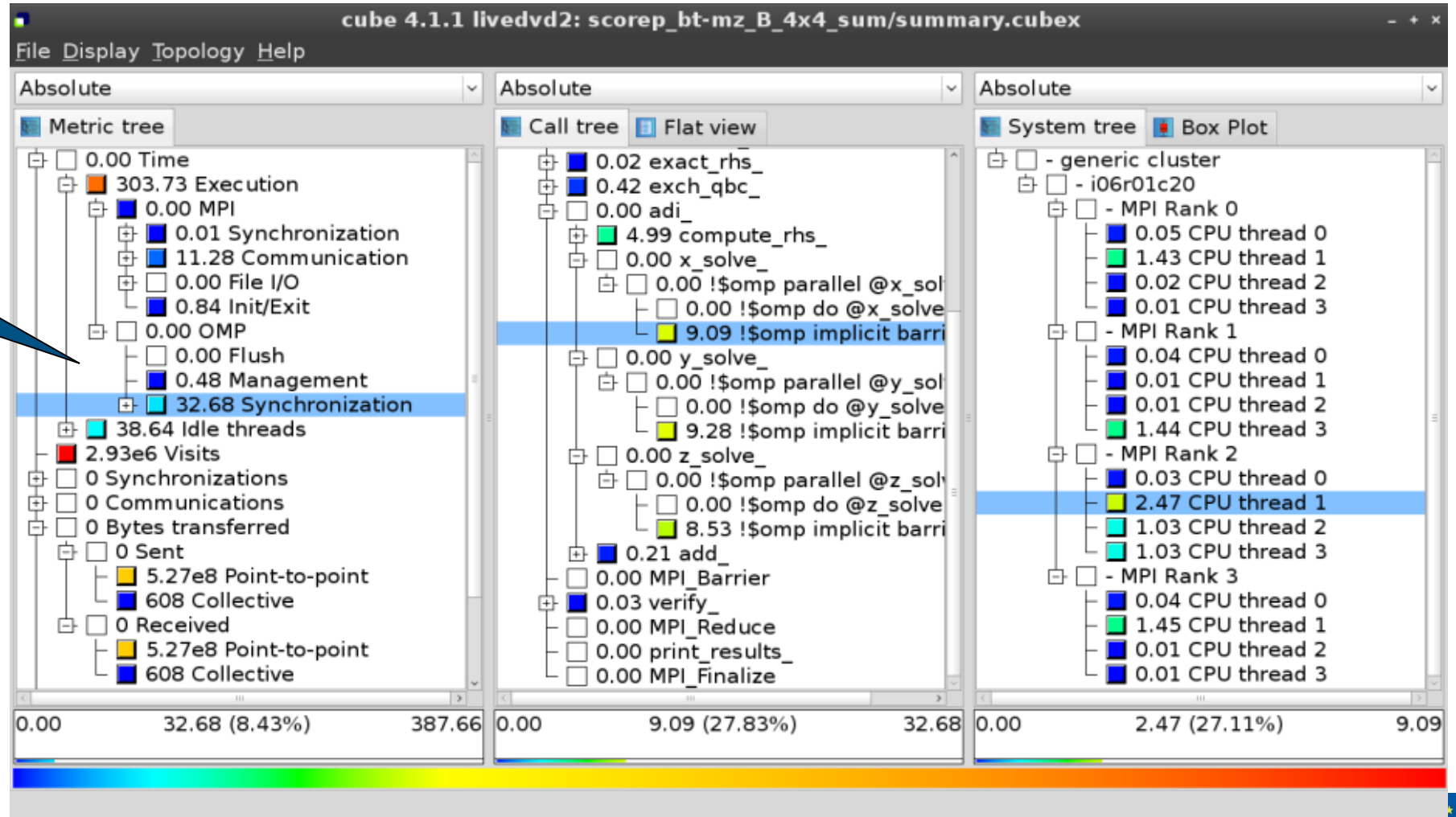
Context-sensitive help available for all GUI items



Summary Analysis Report



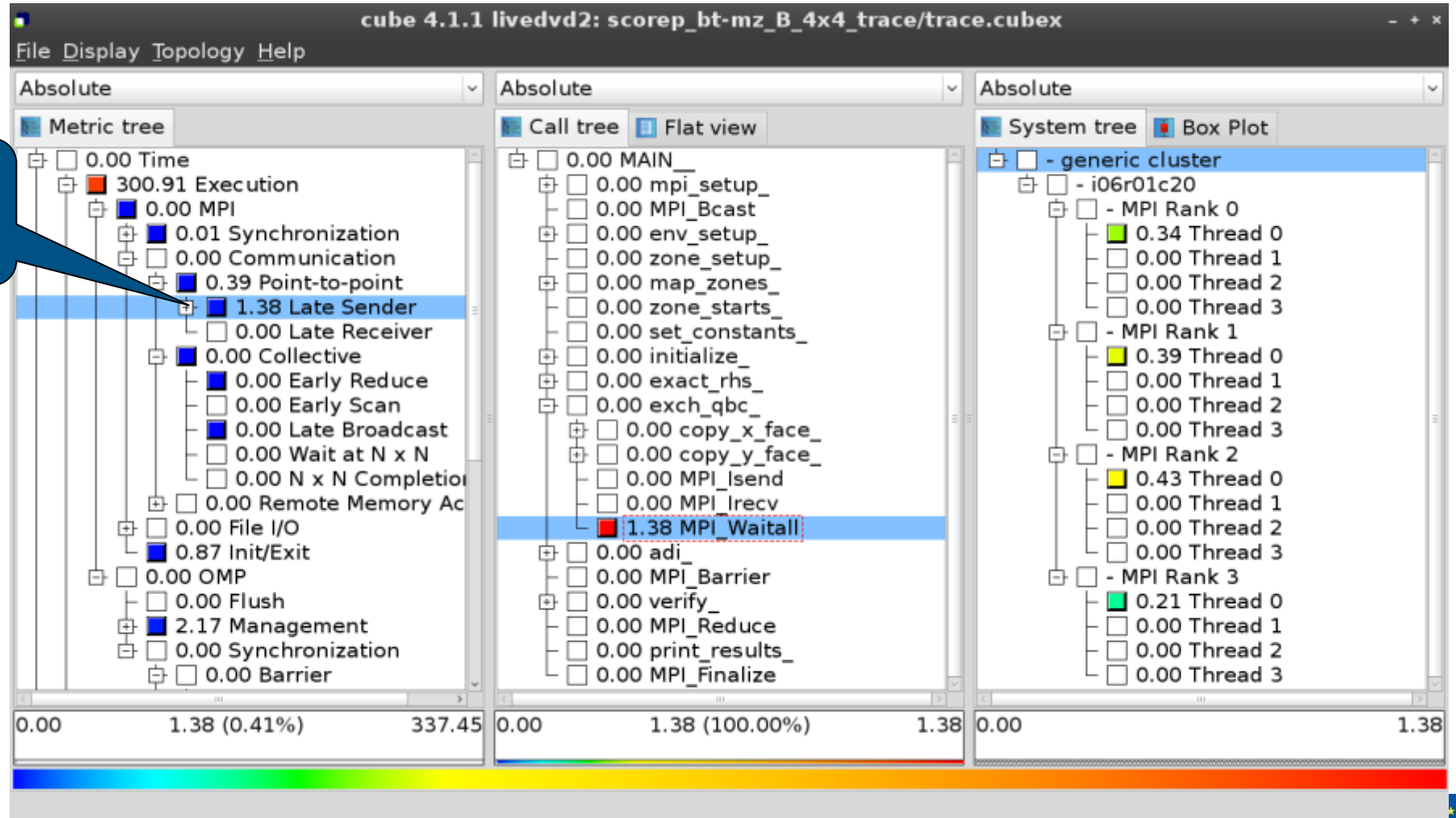
Split base metrics into more specific metrics



Trace Analysis Report



Additional trace-based metrics in metric hierarchy



Online Metric Description



Access online metric description via context menu

cube 4.1.1 livedvd2: scorep_bt-mz_B_4x4_trace/trace.cubex

File Display Topology Help

Absolute Absolute Absolute

Metric tree Call tree Flat view System tree Box Plot

0.00 Time
300.91 Execution
0.00 MPI
0.01 Synchronization
0.00 Communication
0.39 Point-to-point
1.38 Late Sender
0.00 Late Receiver
0.00 Collective
0.00 Early P...
0.00 Early S...
0.00 Late Br...
0.00 Wait at...
0.00 N x N C...
0.00 Remote M...
0.00 File I/O
0.87 Init/Exit
0.00 OMP
0.00 Flush
2.17 Managem...
0.00 Synchroniza...
22.99 Barrier

0.00 1.38 (0.41%) 337.45 0.00 1.38 (100.00%) 1.38 0.00 1.38

Shows the online description of the clicked item



Pattern Instance Statistics



Access pattern instance statistics via context menu

Click to get statistics details

The screenshot displays the 'cube 4.1.1' application interface. The main window shows a 'Metric tree' on the left and a 'Call tree' on the right. A context menu is open over the '1.38 Late Sender' node in the metric tree, with the 'Statistics' option highlighted. A 'Statistics info' dialog box is open, displaying the following data for the 'mpi_latesender' pattern:

Metric	Value	Percentage
Pattern:	mpi_latesender	
Sum:	1.38	
Count:	832	
Mean:	0.00	5%
Standard deviation:	0.00	13%
Maximum:	0.03	100%
Upper quartile (Q3):	0.00	3%
Median:	0.00	3%
Lower quartile (Q1):	0.00	2%
Minimum:	0.00	0%

Below the dialog box, a small histogram shows the distribution of values, with a vertical dashed line at 0.03. The histogram has a y-axis from 0 to 0.035. A 'Close' button is at the bottom of the histogram window.

At the bottom of the main window, a color bar indicates 'Shows metric statistics' with a gradient from blue to red. The histogram window has a 'Close' button at the bottom.





Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

<https://www.pop-coe.eu>

<mailto:pop@bsc.es>

