

## Analysis report examination with Cube

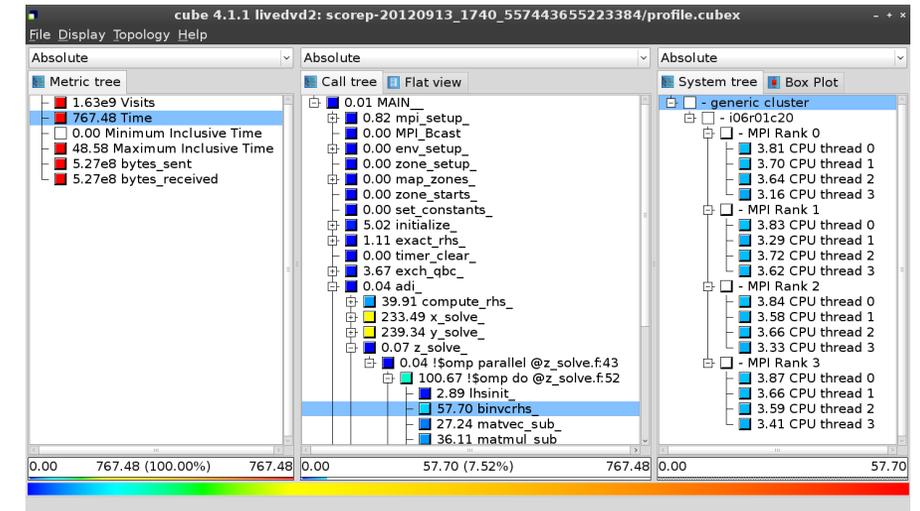
---

Michael Knobloch  
Jülich Supercomputing Centre



# 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  $\geq 4.6$  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 v4.5 (May 2020)



## Note:

Binary packages provided for Windows & MacOS,  
from [www.scalasca.org](http://www.scalasca.org) website in software/Cube-4x

## Cube GUI (goethe)

mailto: scalasca@fz-juelich.de



- Run **remote** (often convenient)
  - start X server (e.g., Xming) locally
  - connect to goethe with X forwarding enabled
    - **-Y** may be faster but is insecure!
  - load cube module and start cube remotely

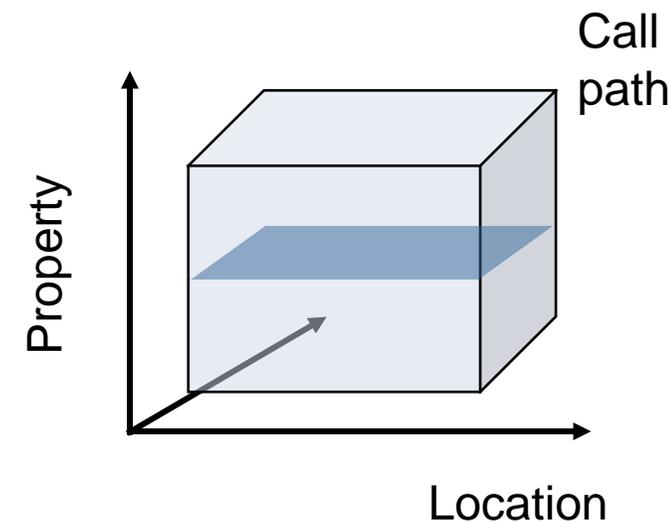
```
desk$ ssh -X goethe
Welcome to goethe..
goethe$ module use /home/vihps/software/modulefiles
goethe$ module load cubegui
goethe$ cube ./scorep_sum/profile.cubex
```

- 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

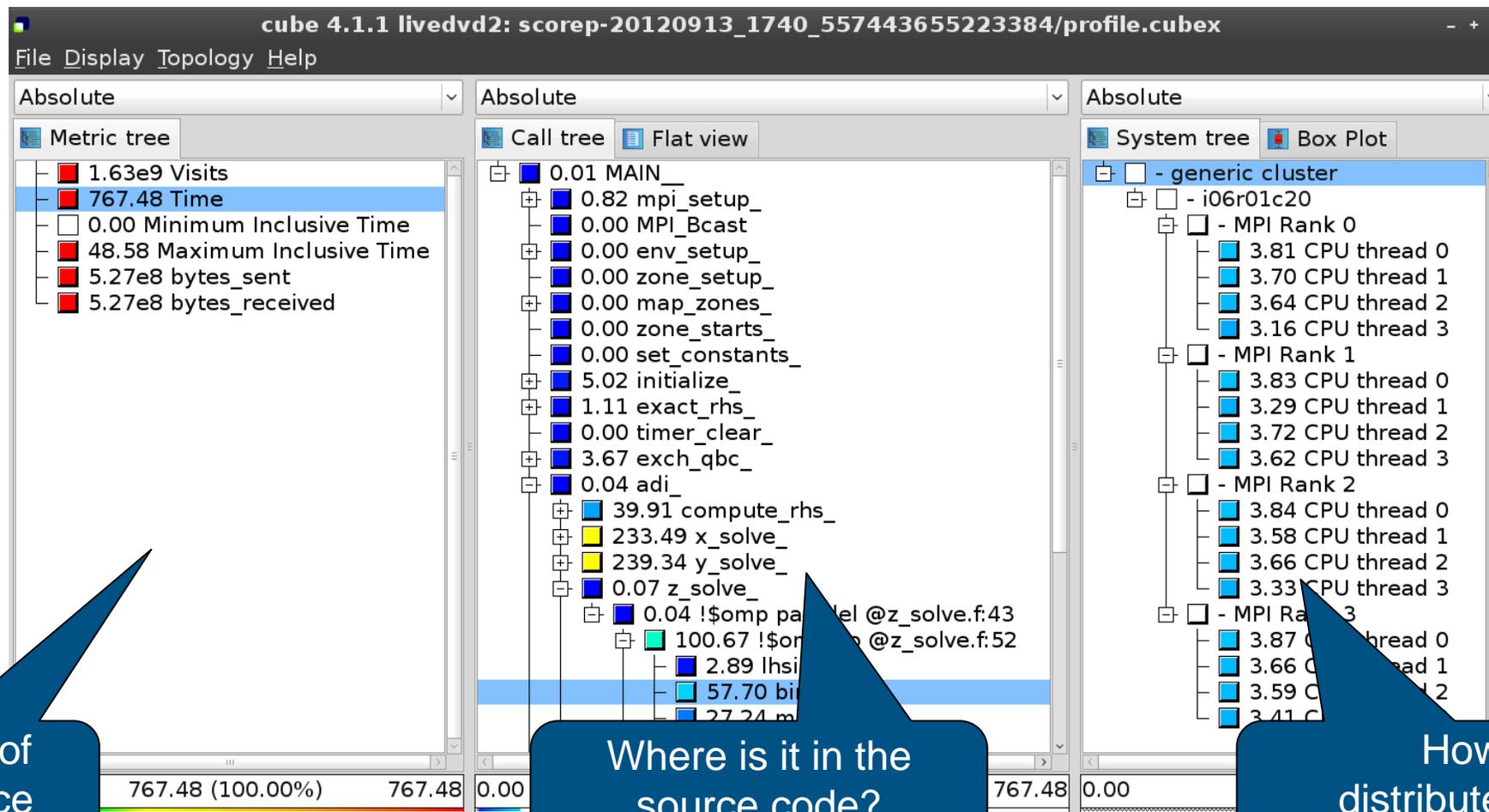
```
desk$ mkdir $HOME/mnt
desk$ sshfs [user@]remote.sys:[dir] $HOME/mnt
desk$ cd $HOME/mnt
desk$ cube ./scorep_sum/profile.cubex
```

# 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

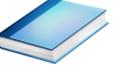


What kind of performance metric?

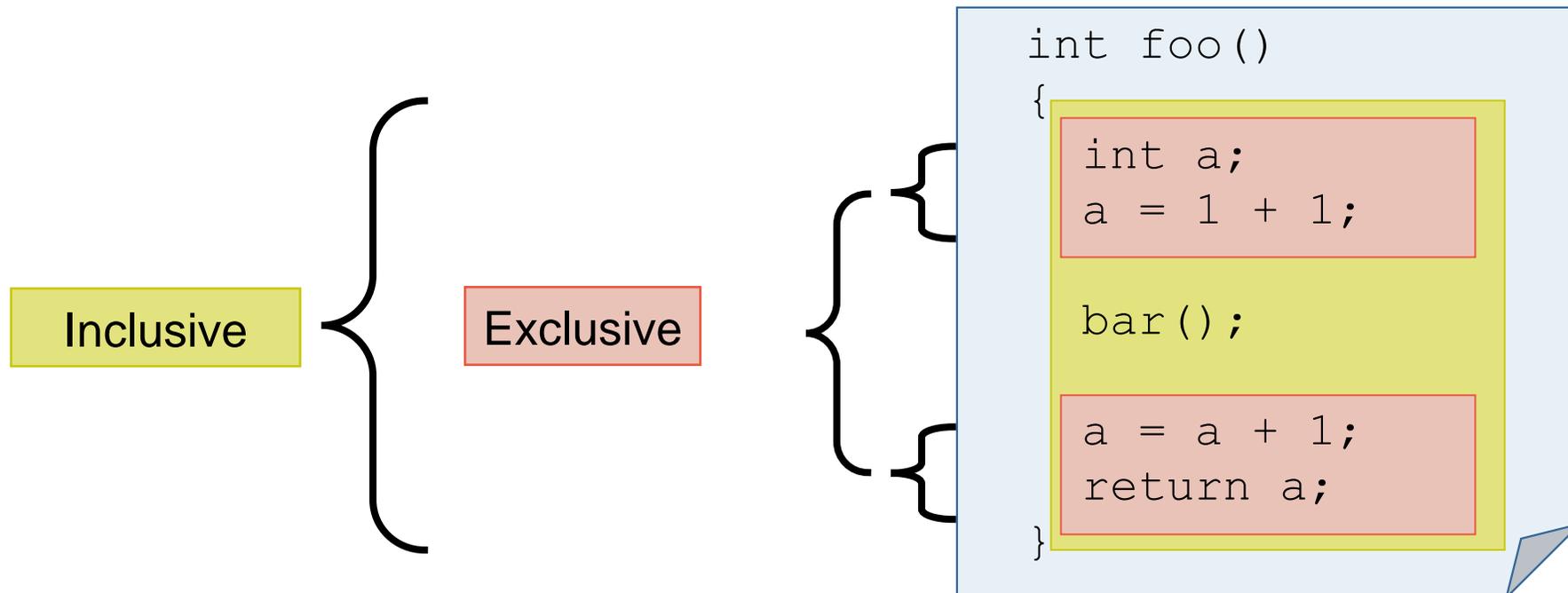
Where is it in the source code?  
In what context?

How is it distributed across the processes/threads?

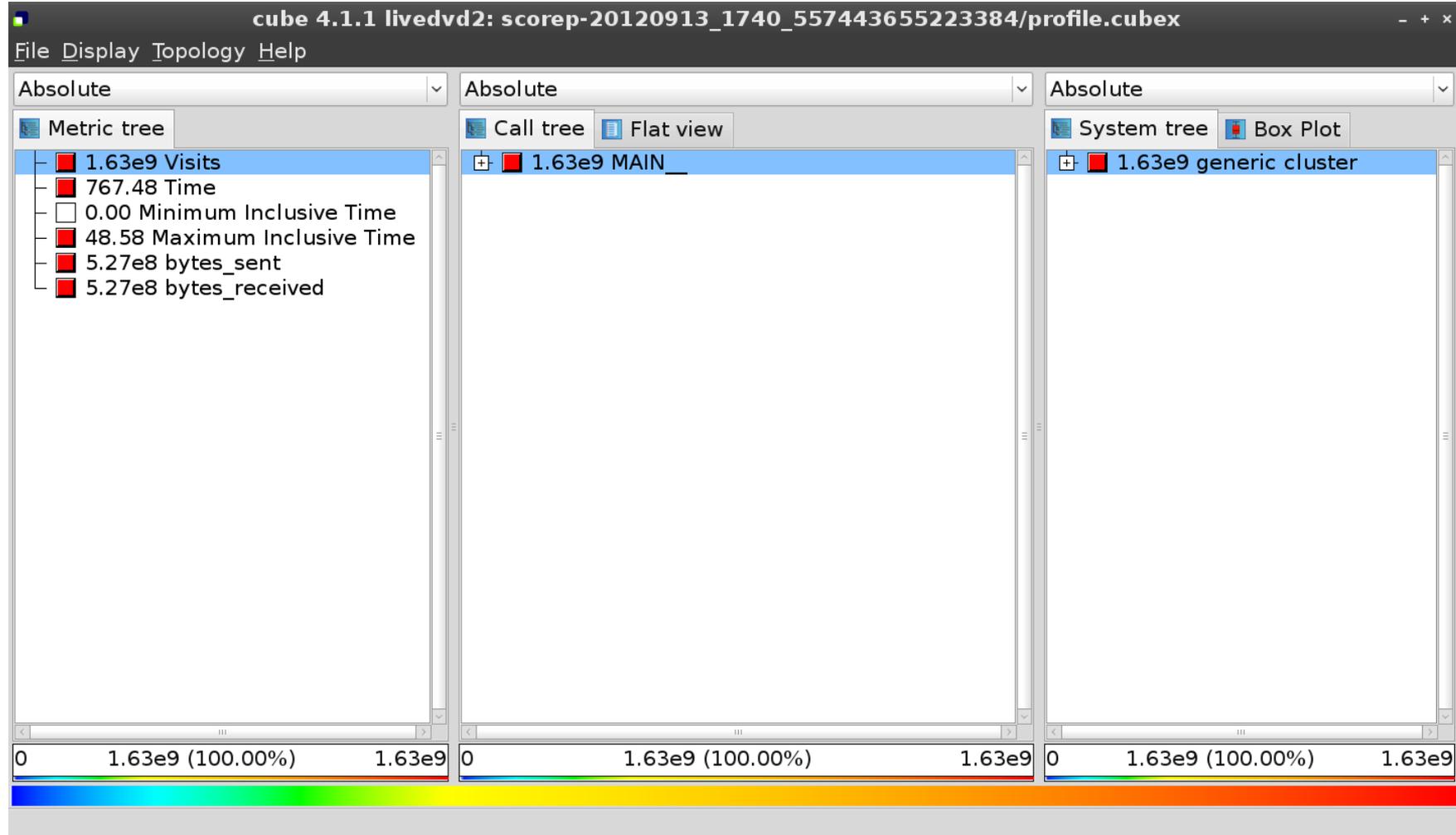
# Inclusive vs. exclusive values



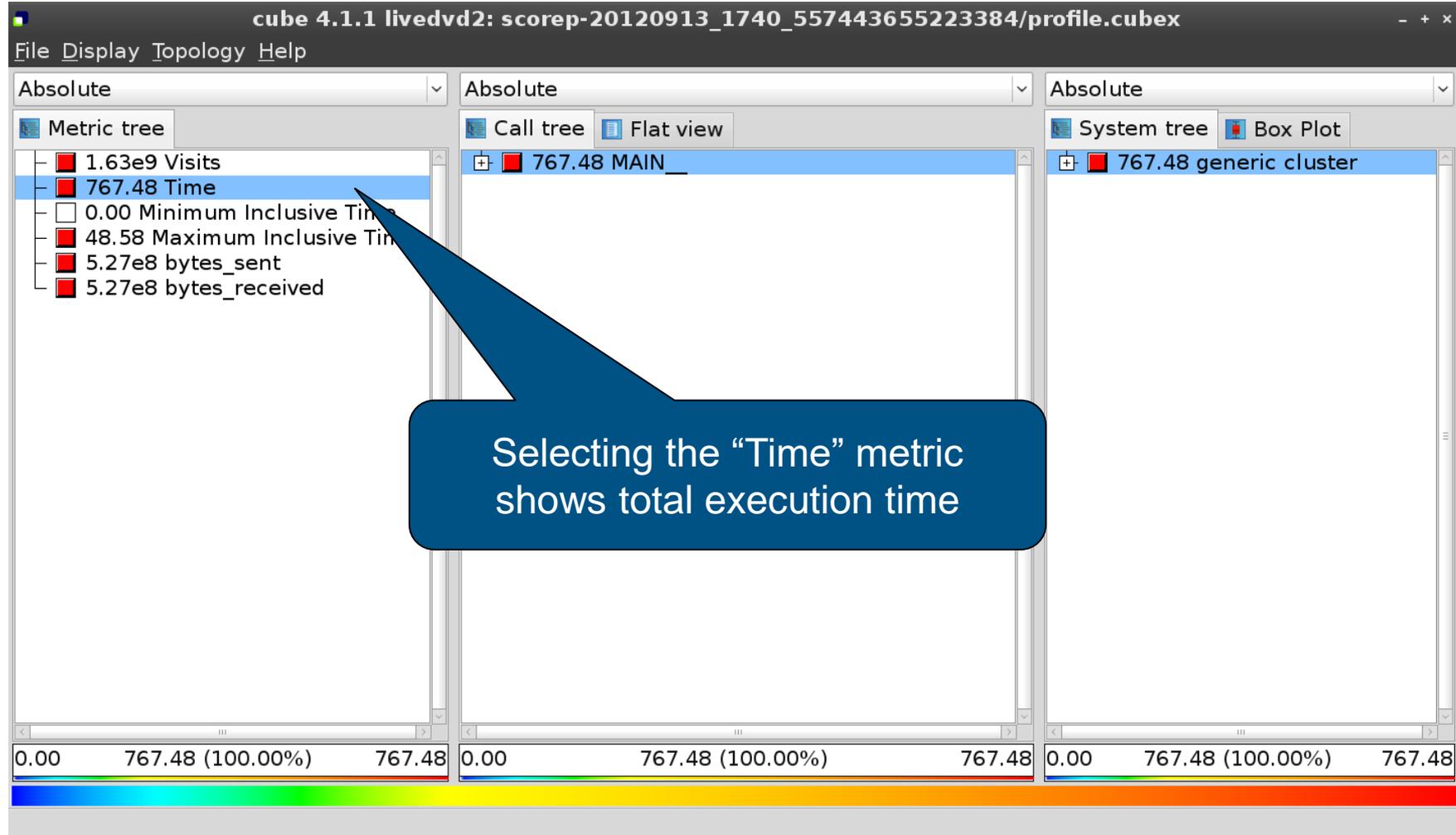
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



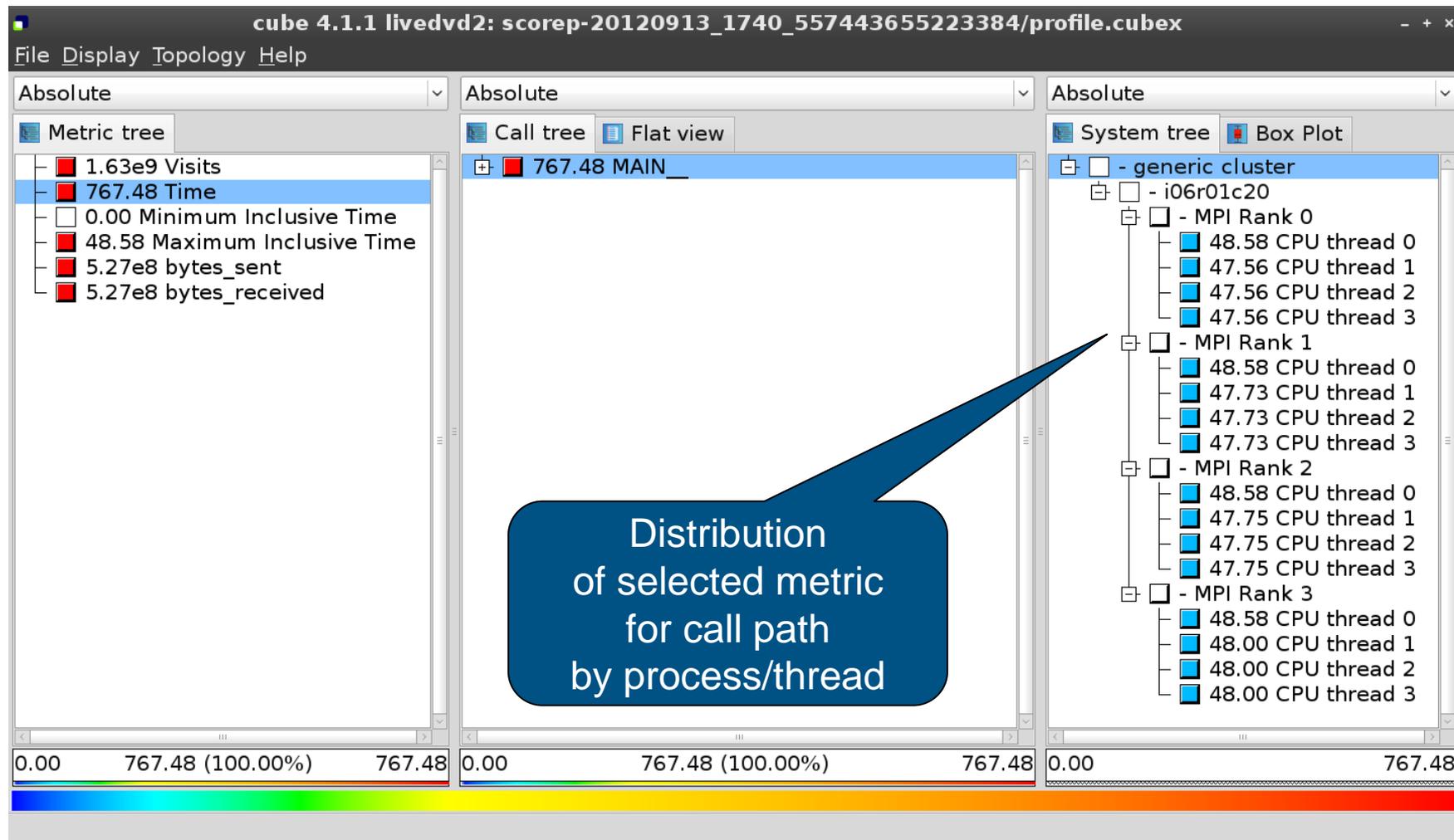
# Score-P analysis report exploration (opening view)



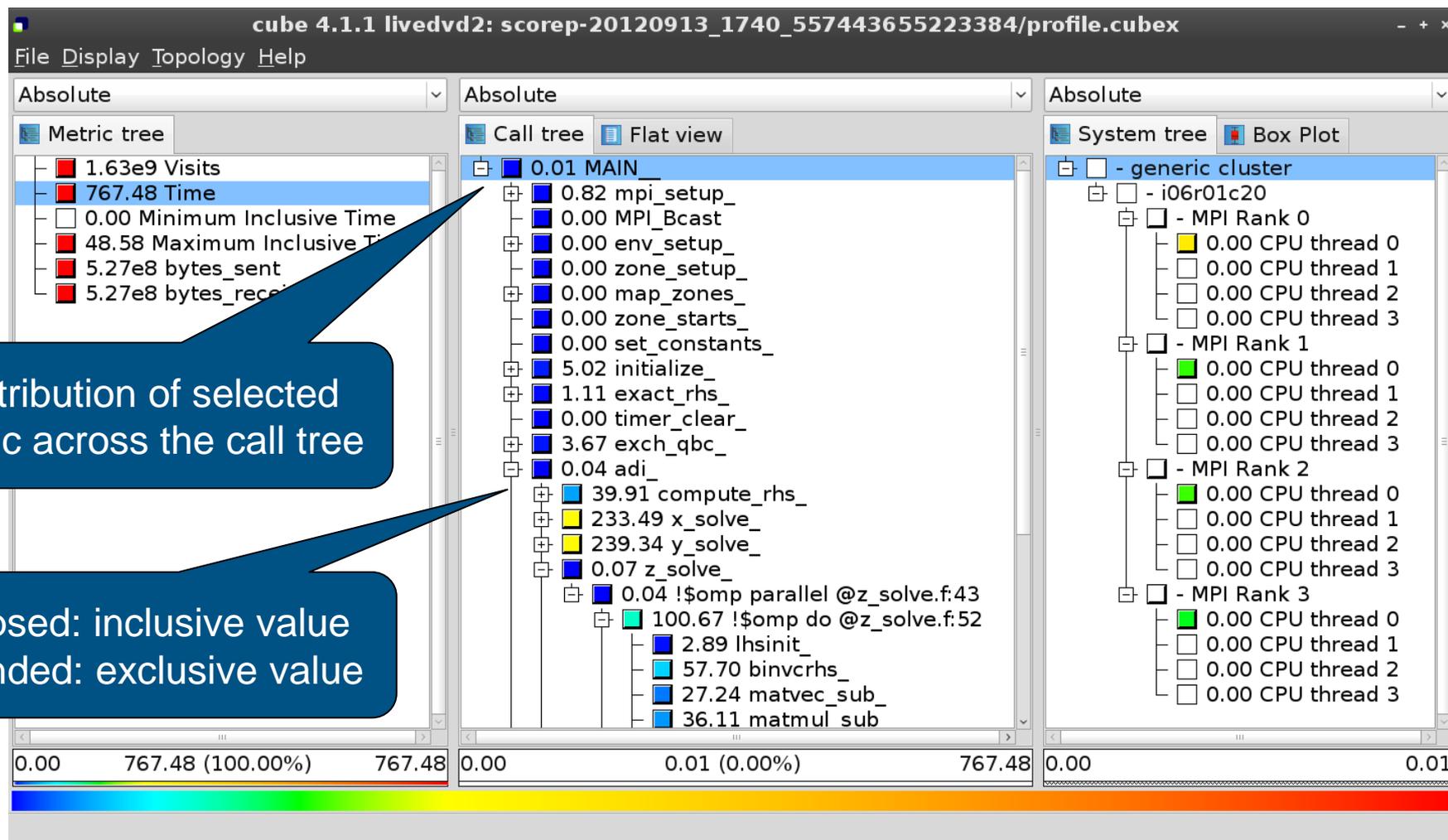
# Metric selection



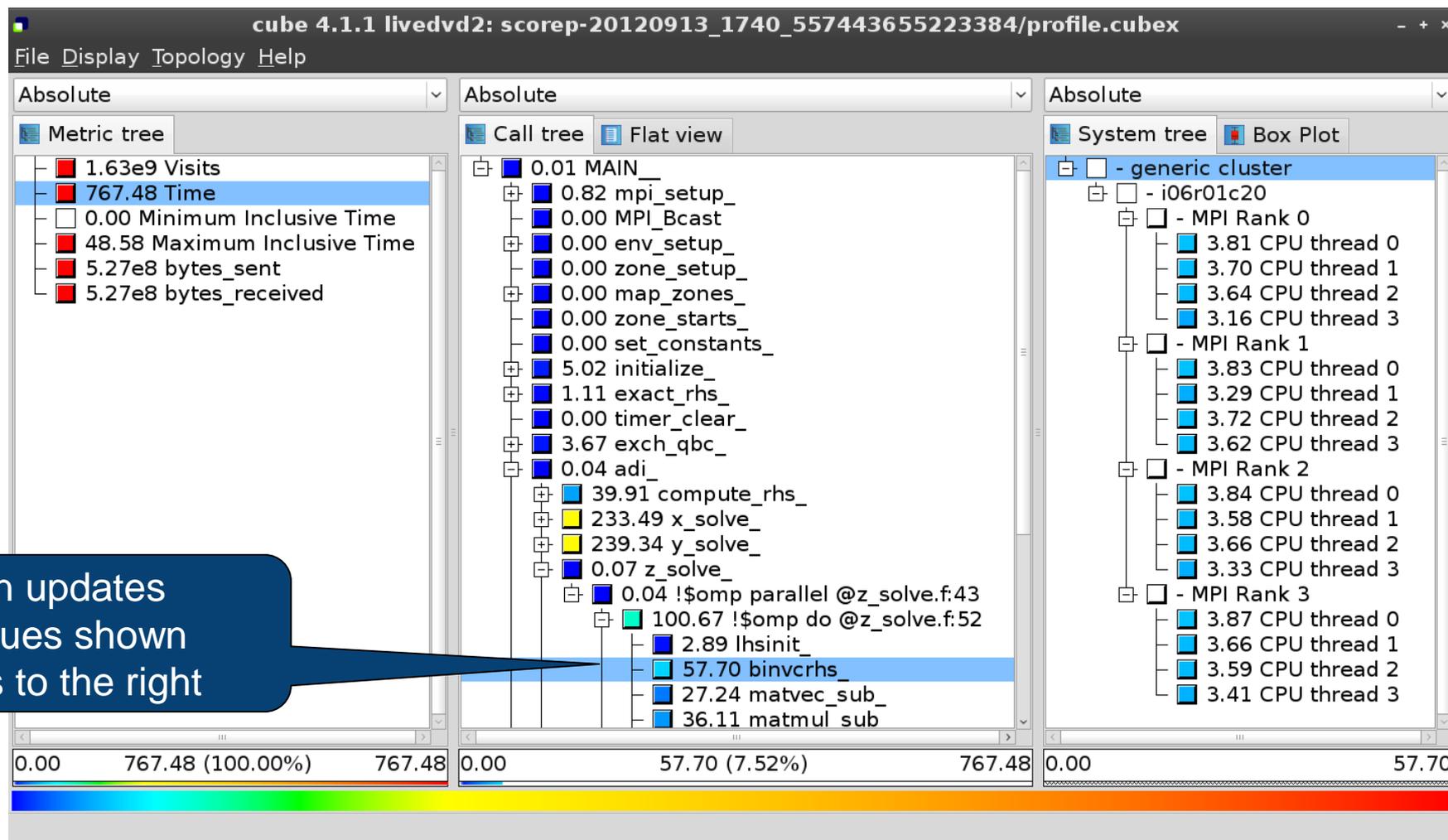
# Expanding the system tree



# Expanding the call tree



# Selecting a call path



# Source-code view via context menu



The screenshot displays the 'cube 4.1.1' performance analysis tool interface. The main window is titled 'cube 4.1.1 livedvd2: scorep-20120913\_1740\_557443655223384/profile.cubex'. It features three main panels: 'Metric tree', 'Call tree', and 'System tree'. The 'Metric tree' on the left shows a list of metrics, with '767.48 Time' selected. The 'Call tree' in the center shows a hierarchical view of function calls, with '57.70 binvcrhs' selected. A context menu is open over the selected item, listing options such as 'Call site', 'Called region', 'Expand/collapse', 'Hiding', 'Cut call tree', 'Find items', 'Find Next', 'Clear found items', 'Copy to clipboard', and 'Min/max values'. The 'Source code' option is highlighted. The 'System tree' on the right shows a hierarchical view of the system components, including 'generic cluster', 'i06r01c20', and 'MPI Rank 0'. A blue callout box with a white border and a pointer to the context menu contains the text 'Right-click opens context menu'. At the bottom of the window, a status bar shows the current selection: 'Shows the source code of the clicked item'.

Right-click opens context menu

Shows the source code of the clicked item

# Source-code view



```
subroutine binvcrhs( 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-----

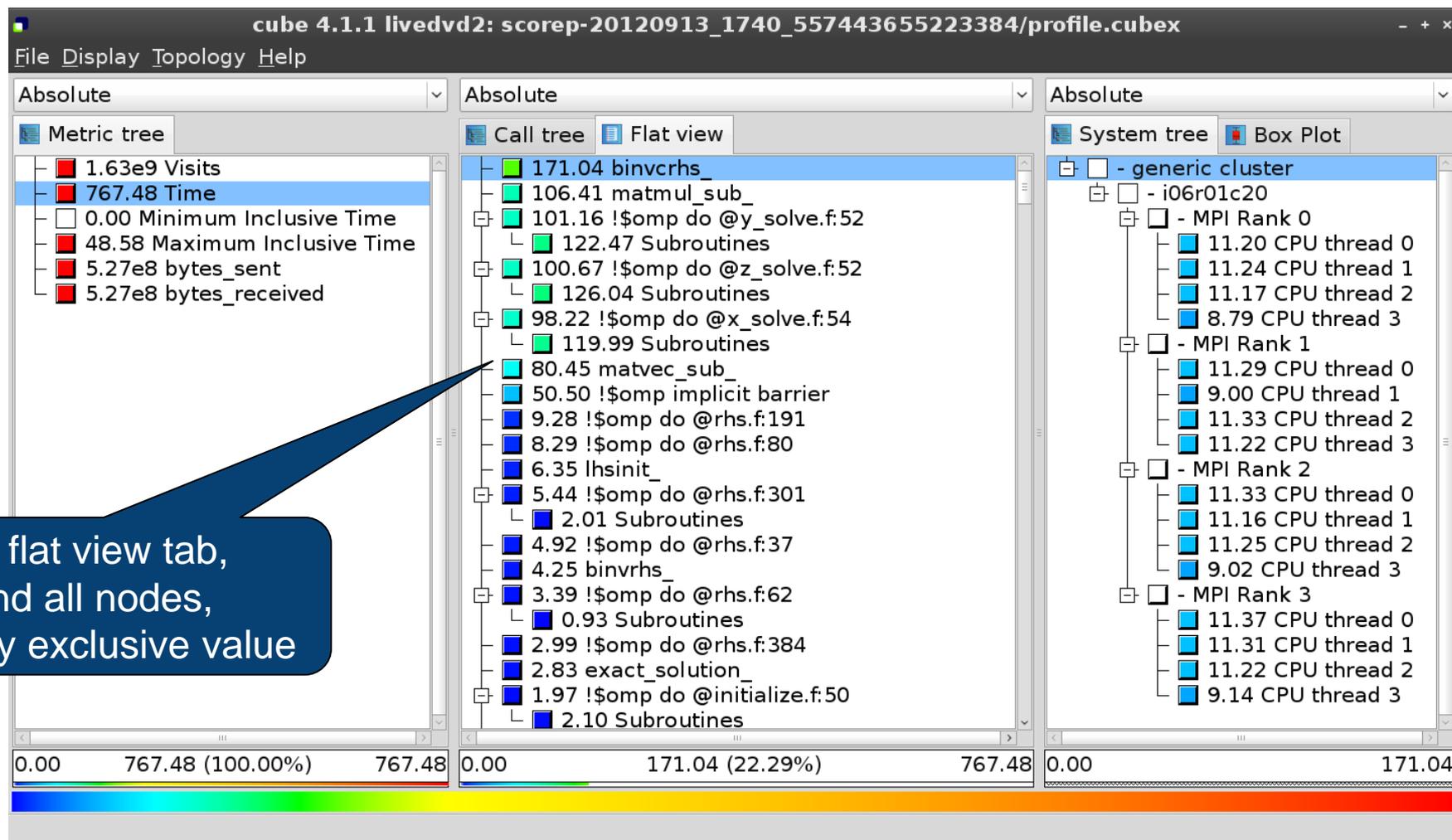
pivot = 1.00d0/lhs(1,1)
lhs(1,2) = lhs(1,2)*pivot
lhs(1,3) = lhs(1,3)*pivot
lhs(1,4) = lhs(1,4)*pivot
lhs(1,5) = lhs(1,5)*pivot
c(1,1) = c(1,1)*pivot
c(1,2) = c(1,2)*pivot
c(1,3) = c(1,3)*pivot
c(1,4) = c(1,4)*pivot
```

Read only Save Save as Font... Close

## Note:

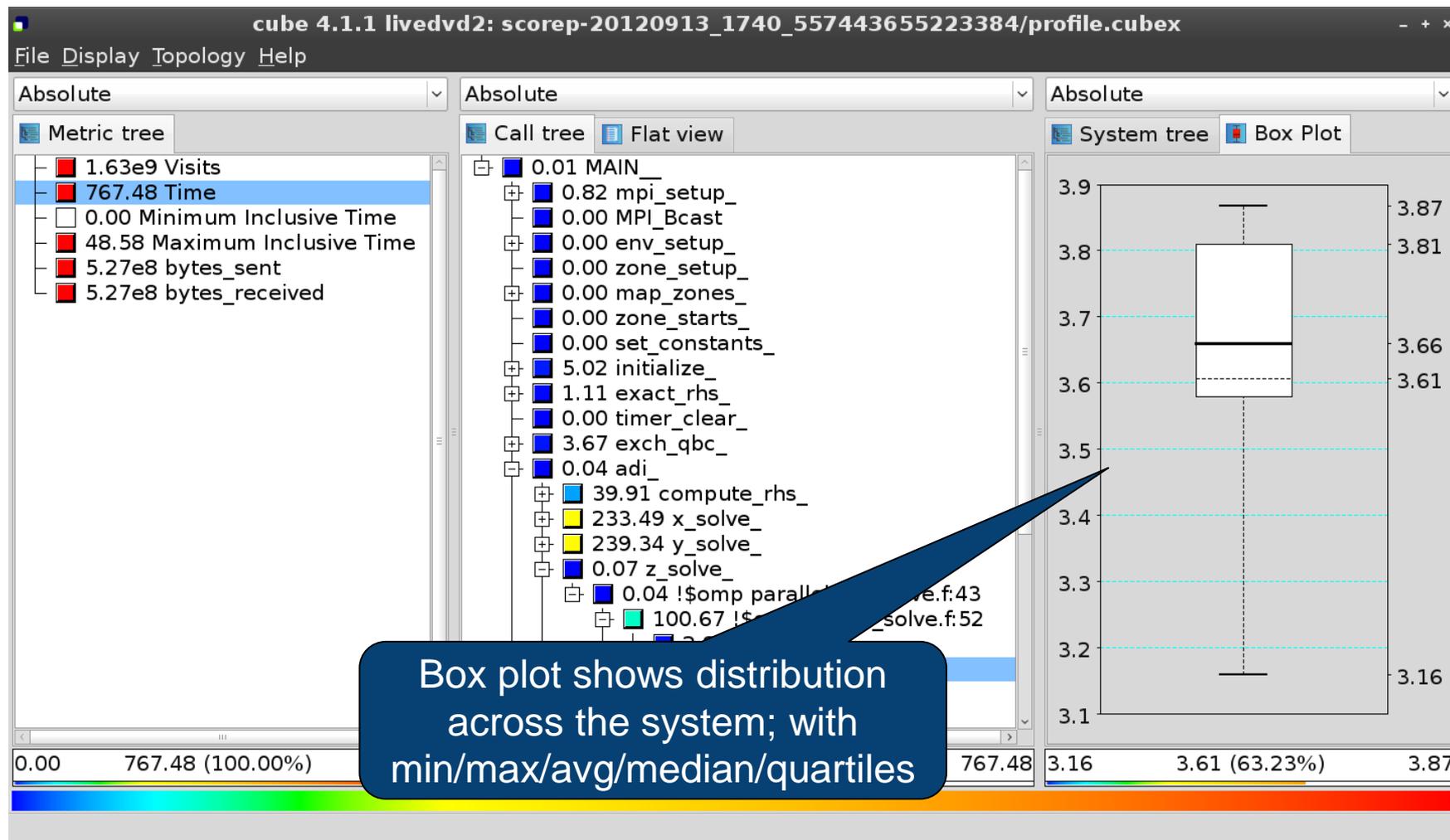
This feature depends on file and line number information provided by the instrumentation, i.e., it may not always be available

# Flat profile view

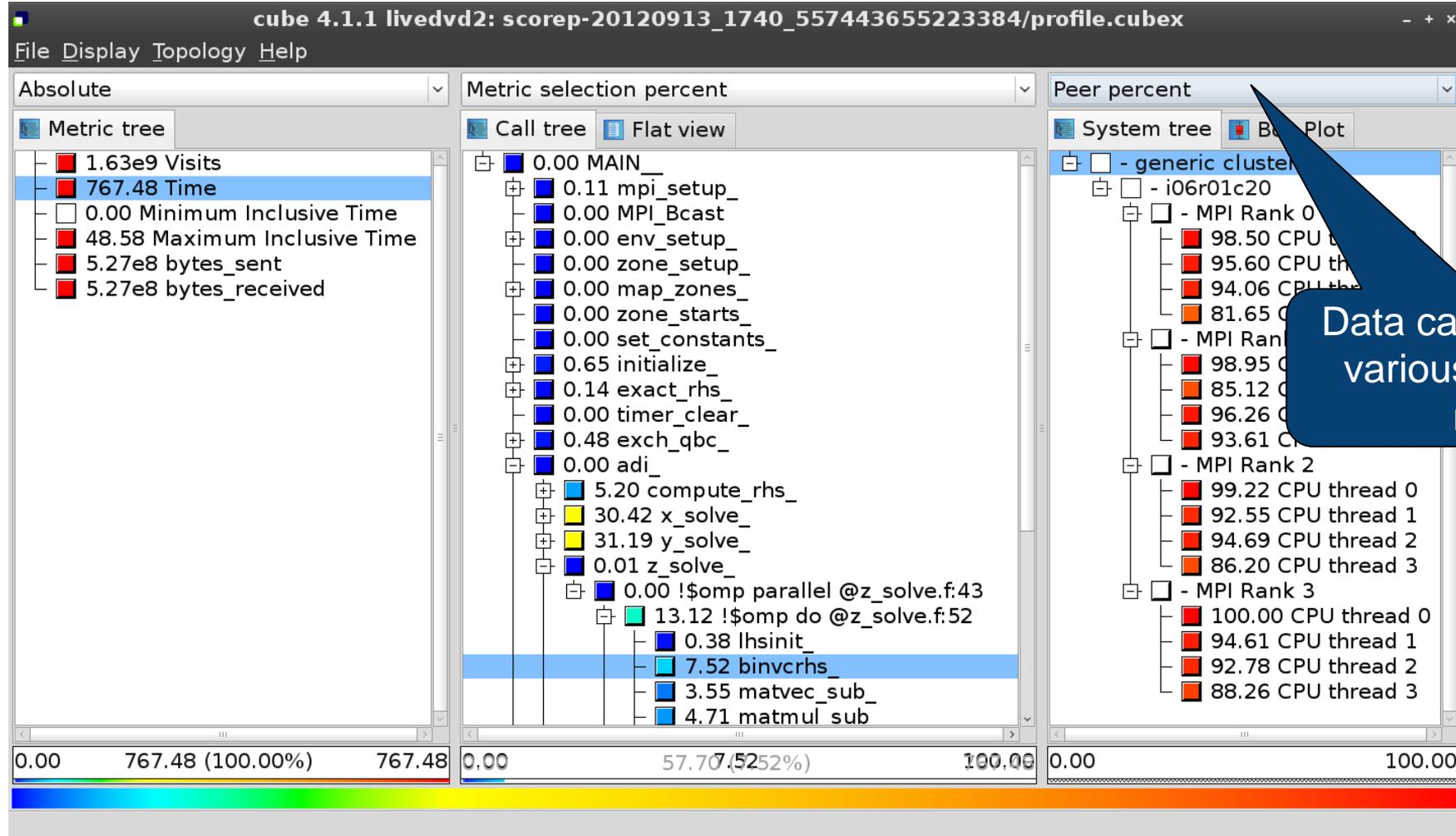


Select flat view tab,  
expand all nodes,  
and sort by exclusive value

# Box plot view

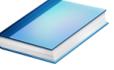


# Alternative display modes



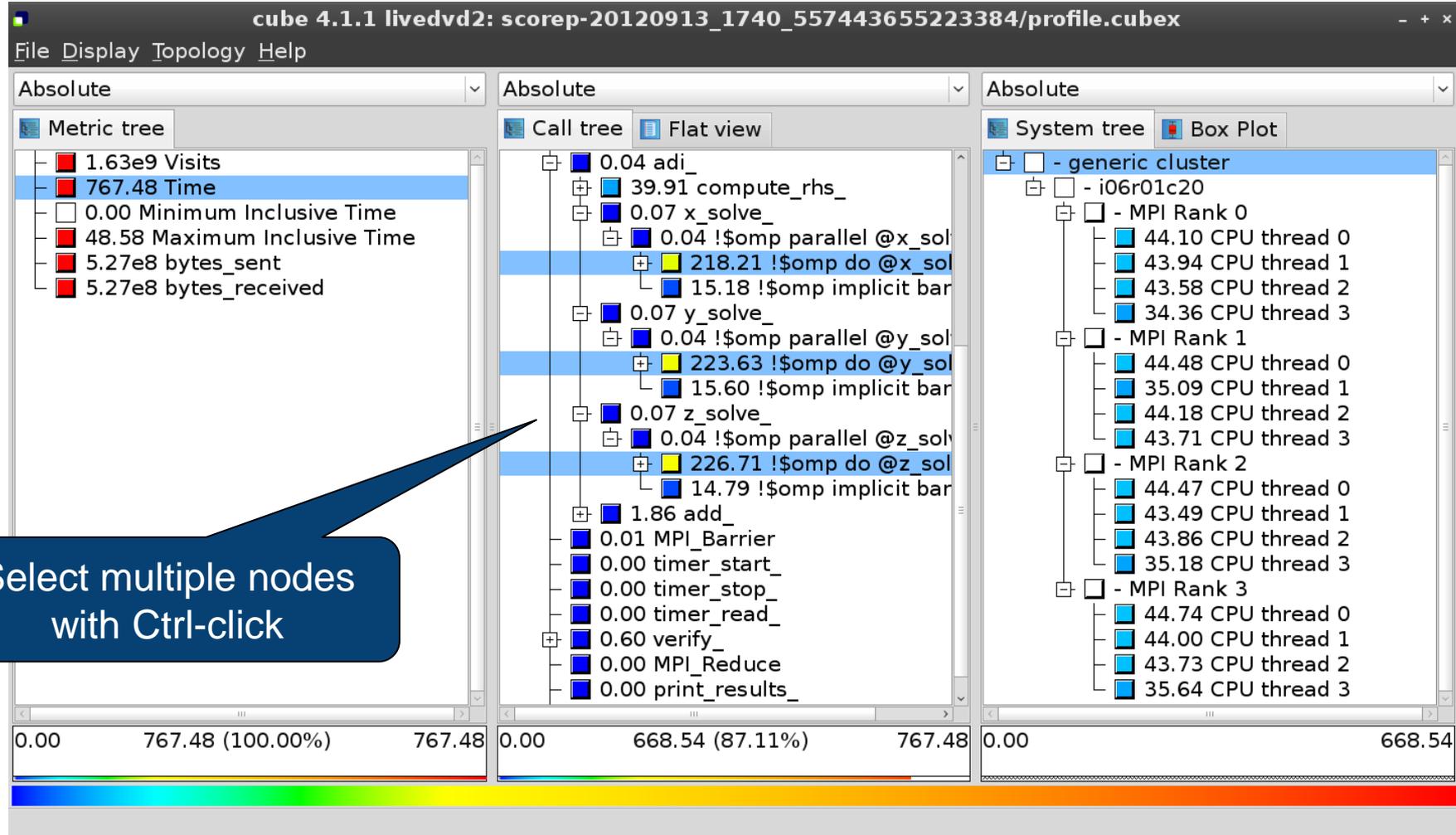
# Important display modes

---



- 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



The screenshot displays the cube 4.1.1 GUI with a 'Help' menu open over the 'Metric tree' panel. The menu options are: Getting started, Mouse and keyboard control, What's This? (highlighted), and About. A tooltip for 'What's This?' shows 'Selected metrics description' and 'Selected regions description'. The 'Metric tree' panel shows a list of metrics, with '767.48 Time' selected. The 'System tree' panel shows a hierarchical view of the system components, including MPI Ranks and CPU threads. A status bar at the bottom indicates 'Change into help mode for display components'.

cube 4.1.1 livedvd2: scorep-20120913\_1740\_557443655223384/profile.cubex

File Display Topology Help

Absolute

Metric tree

- 1.63e9 Visits
- 767.48 Time
- 0.00 Minimum I
- 48.58 Maximum
- 5.27e8 byt
- 5.27e8

Getting started  
Mouse and keyboard control  
What's This? (Shift+F1)  
About

Selected metrics description  
Selected regions description

compute\_rhs\_  
\_solve\_  
4 !\$omp parallel @x\_sol  
218.21 !\$omp do @x\_sol  
15.18 !\$omp implicit bar  
0.07 y\_solve\_  
0.04 !\$omp parallel @y\_sol  
223.63 !\$omp do @y\_sol  
15.60 !\$omp implicit bar  
0.07 z\_solve\_  
0.04 !\$omp parallel @z\_sol  
226.71 !\$omp do @z\_sol  
14.79 !\$omp implicit bar  
1.86 add\_  
0.01 MPI\_Barrier  
0.00 timer\_start\_  
0.00 timer\_stop\_  
0.00 timer\_read\_  
0.60 verify\_  
0.00 MPI\_Reduce  
0.00 print\_results\_

Absolute

System tree Box Plot

- generic cluster
  - i06r01c20
    - MPI Rank 0
      - 44.10 CPU thread 0
      - 43.94 CPU thread 1
      - 43.58 CPU thread 2
      - 34.36 CPU thread 3
    - MPI Rank 1
      - 44.48 CPU thread 0
      - 35.09 CPU thread 1
      - 44.18 CPU thread 2
      - 43.71 CPU thread 3
    - MPI Rank 2
      - 44.47 CPU thread 0
      - 43.49 CPU thread 1
      - 43.86 CPU thread 2
      - 35.18 CPU thread 3
    - MPI Rank 3
      - 44.74 CPU thread 0
      - 44.00 CPU thread 1
      - 43.73 CPU thread 2
      - 35.64 CPU thread 3

0.00 767.48 (100.00%) 767.48 0.00 668.54 (87.11%) 767.48 0.00 668.54

Change into help mode for display components

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

Cube-4.3.1: scorep-20140130\_2053\_439988765917/profile.cubex

File Display Plugins Help

Absolute

Metric tree

- 1.09e8 Visits (occ)
- 1.01e6 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 246.14 Maximum Inclusive Time (sec)
- 7.18e12 bytes\_sent
- 7.18e12 bytes\_received

Absolute

Call tree Flat view

- 0.35 main(int, char \*)
- 2512.10 ugshellInit
- 1.01e6 ug::script::LoadUGScript(const char \*, bool)
- 2.11 ug::script::ParseBuffer(const char \*, const char \*)
- 0.04 ugshellFinalize
- 94.31 MPI\_Finalize

Absolute

System tree Demo Plugin Label Example Plugin Label

- machine Blue Gene/Q
- rack 11
- midplane 1
- nodeboard 8
- nodecard 4
- 0.65 MPI Rank 0
- 0.64 MPI Rank 1

Create new metric as a child of metric

Select metric from collection: Average execution time (kenobi)

Derived metric type: Postderived metric

Display name: Average visit time

Unique name: avg\_visit\_time

Data type: DOUBLE

Unit of measurement: sec

URL:

Description:

Calculates average time of region execution per visit.  
Autor is Michael Knobloch.

Calculation

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

Create metric Cancel

Share this metric with SCALASCA group

1.01e6 (100.00%) 1.01e6 0.00 2512.10

# Example: FLOPS based on PAPI\_FP\_OPS and time



The screenshot displays the Cube-4.3.1 performance analysis tool interface, showing the configuration of a derived metric and its hierarchical breakdown.

**Left Panel: Edit metric FLOPS (on froggy1)**

- Select metric from collection: --- please select ---
- Derived metric type: Postderived metric
- Display name: FLOPS
- Unique name: flops
- Data type: DOUBLE
- Unit of measurement:
- URL:
- Description:
- Calculation:  $\text{metric::PAPI\_FP\_OPS()} / \text{metric::time()}$

**Metric tree (Absolute):**

- 1.17e7 Visits (occ)
- 1148.49 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 41.57 Maximum Inclusive Time (...)
- 0 bytes\_put (bytes)
- 0 bytes\_get (bytes)
- 5.75e12 PAPI\_TOT\_INS (#)
- 2.69e12 PAPI\_TOT\_CYC (#)
- 2.12e12 PAPI\_FP\_OPS (#)
- 3.12e9 bytes\_sent (bytes)
- 3.12e9 bytes\_received (bytes)
- 1.84e9 FLOPS

**Call tree (Absolute):**

- 3.17e5 MAIN\_
- 7.04e5 mpi\_setup\_
- 6.34e4 MPI\_Bcast
- 2.05e5 env\_setup\_
- 7.39e5 zone\_setup\_
- 9.31e5 map\_zones\_
- 9.39e4 zone\_starts\_
- 6.16e5 set\_constants\_
- 5.91e8 initialize\_
- 0.00 exact\_rhs\_
- 145.62 !\$omp parallel @exac...
  - 2.54e4 !\$omp do @exact\_r...
  - 9.65e8 !\$omp do @exact\_r...
  - 9.62e8 !\$omp do @exact\_r...
  - 8.14e8 !\$omp do @exact\_r...
  - 1.21e5 !\$omp do @exact\_r...
  - 0.00 !\$omp implicit barrier...
- 6.23e4 exch\_qbc\_
- 1.94e9 adi\_
- 2.19e5 MPI\_Barrier
- 1.92e9 <<bt\_iter>> (200 itera...
- 1.98e8 verify\_
- 1.05e5 MPI\_Reduce

**System tree (Absolute):**

- machine Linux
  - node frog6
    - MPI Rank 0
      - 1.17e9 Master thread
      - 9.43e8 OMP thread 1
      - 9.47e8 OMP thread 2
      - 9.47e8 OMP thread 3
    - MPI Rank 1
      - 1.17e9 Master thread
      - 9.87e8 OMP thread 1
      - 9.68e8 OMP thread 2
      - 9.72e8 OMP thread 3
    - MPI Rank 2
      - 1.10e9 Master thread
      - 8.97e8 OMP thread 1
      - 8.77e8 OMP thread 2
      - 8.76e8 OMP thread 3
    - MPI Rank 3
      - 1.09e9 Master thread
      - 9.06e8 OMP thread 1
      - 9.04e8 OMP thread 2
      - 9.02e8 OMP thread 3

**Bottom Panel:**

- 0.00 1.84e9 (100.00%) 1.84e9
- 0.00 9.65e8 (-0.00%) -12858016489314434.00
- 0.00... -179769313486231570814527423731704356798070...

Selected "\$omp do @exact\_rhs.f:46"

# Iteration profiling

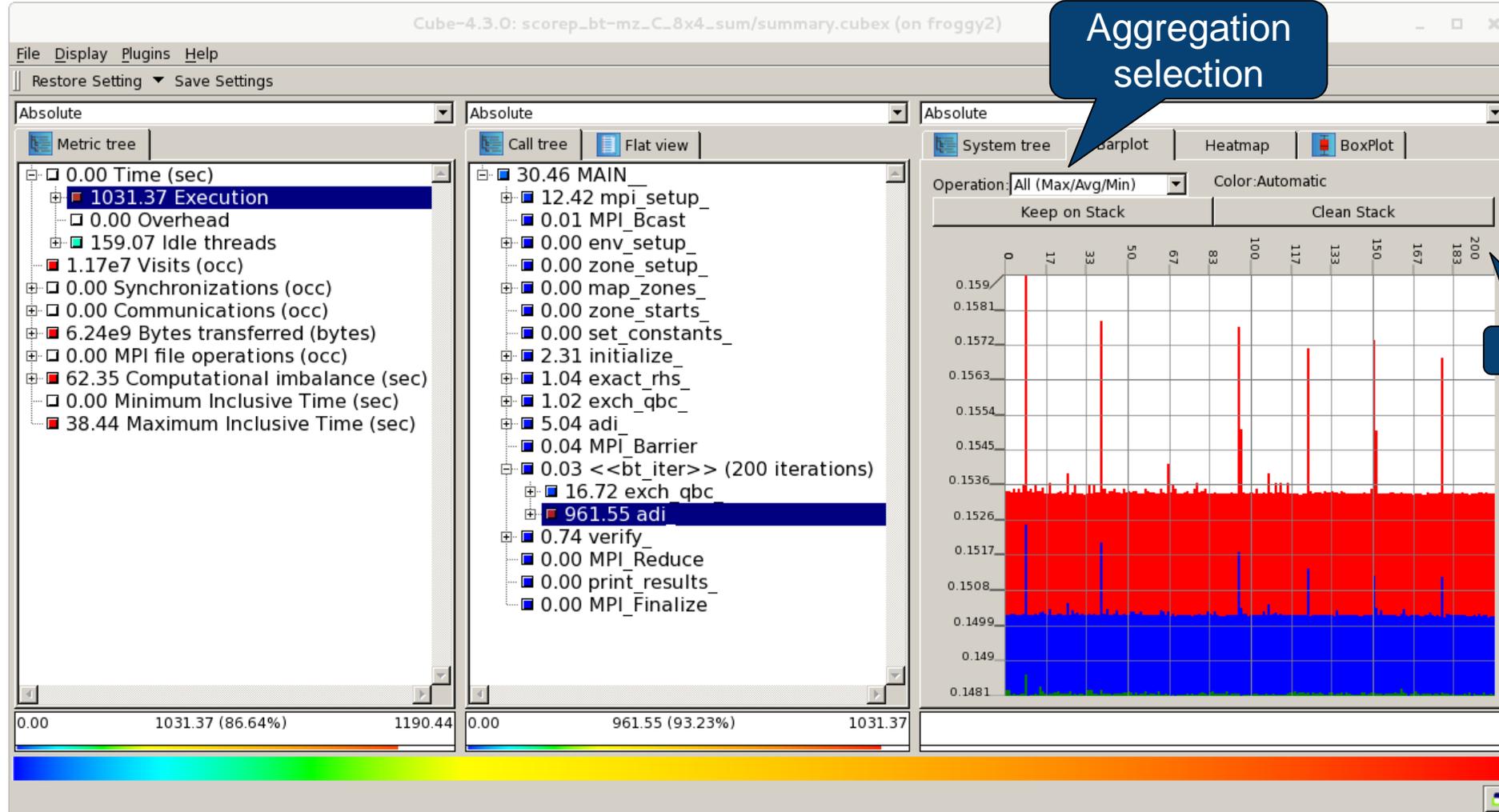


- Show time dependent behavior by “unrolling” iterations
- Preparations:
  - Mark loop body by using Score-P instrumentation API in your source code

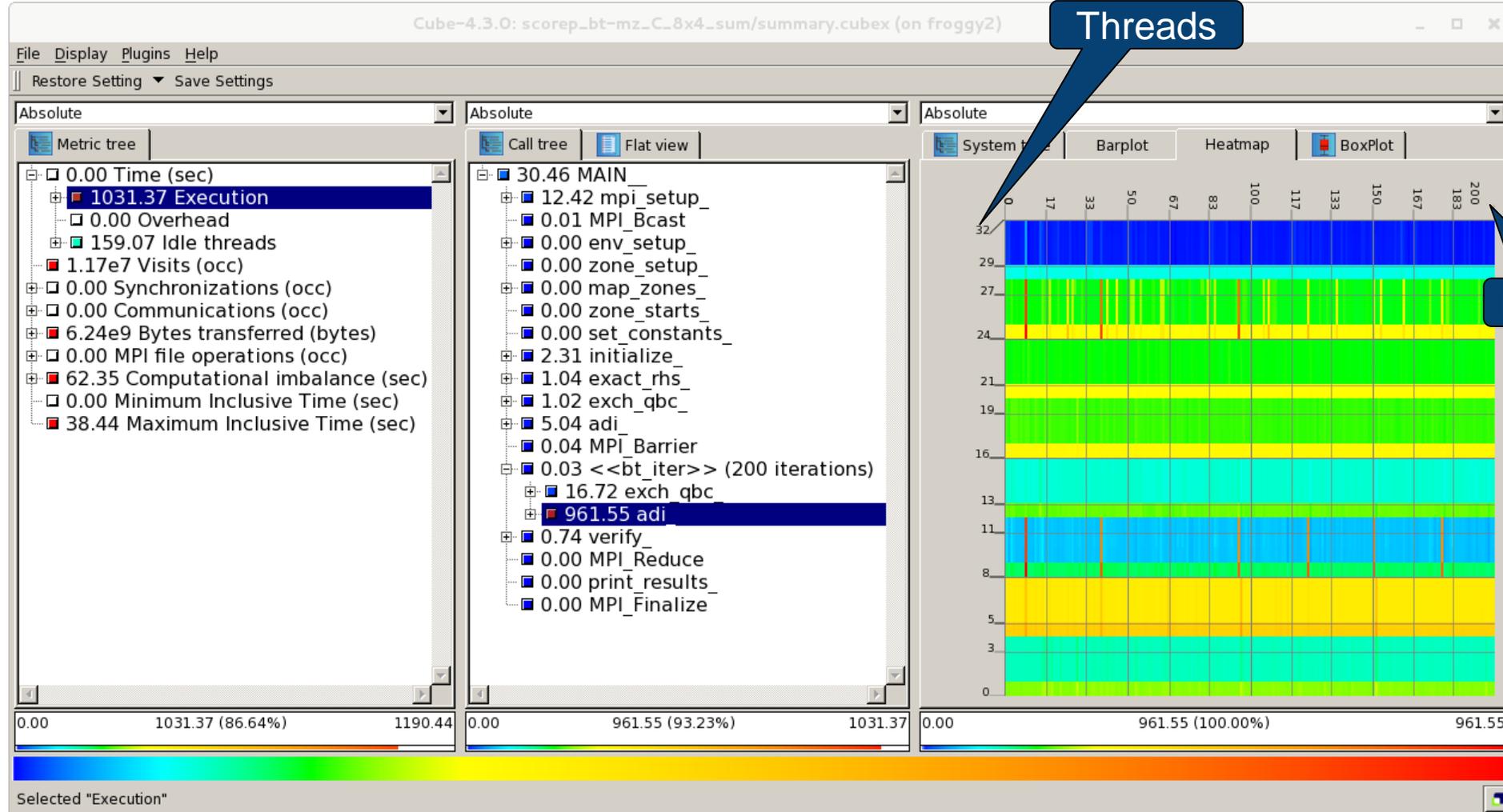
```
SCOREP_USER_REGION_DEFINE( scorep_bt_loop )  
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )  
SCOREP_USER_REGION_END( scorep_bt_loop )
```

- 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



## Cube: Further information

---

- Parallel program analysis report exploration tools
  - Libraries for Cube report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - <http://www.scalasca.org>
- User guide also part of installation:
  - `<prefix>/share/doc/CubeGuide.pdf`
- Contact:
  - mailto: [scalasca@fz-juelich.de](mailto:scalasca@fz-juelich.de)



## Score-P/CUBE case study HemeLB

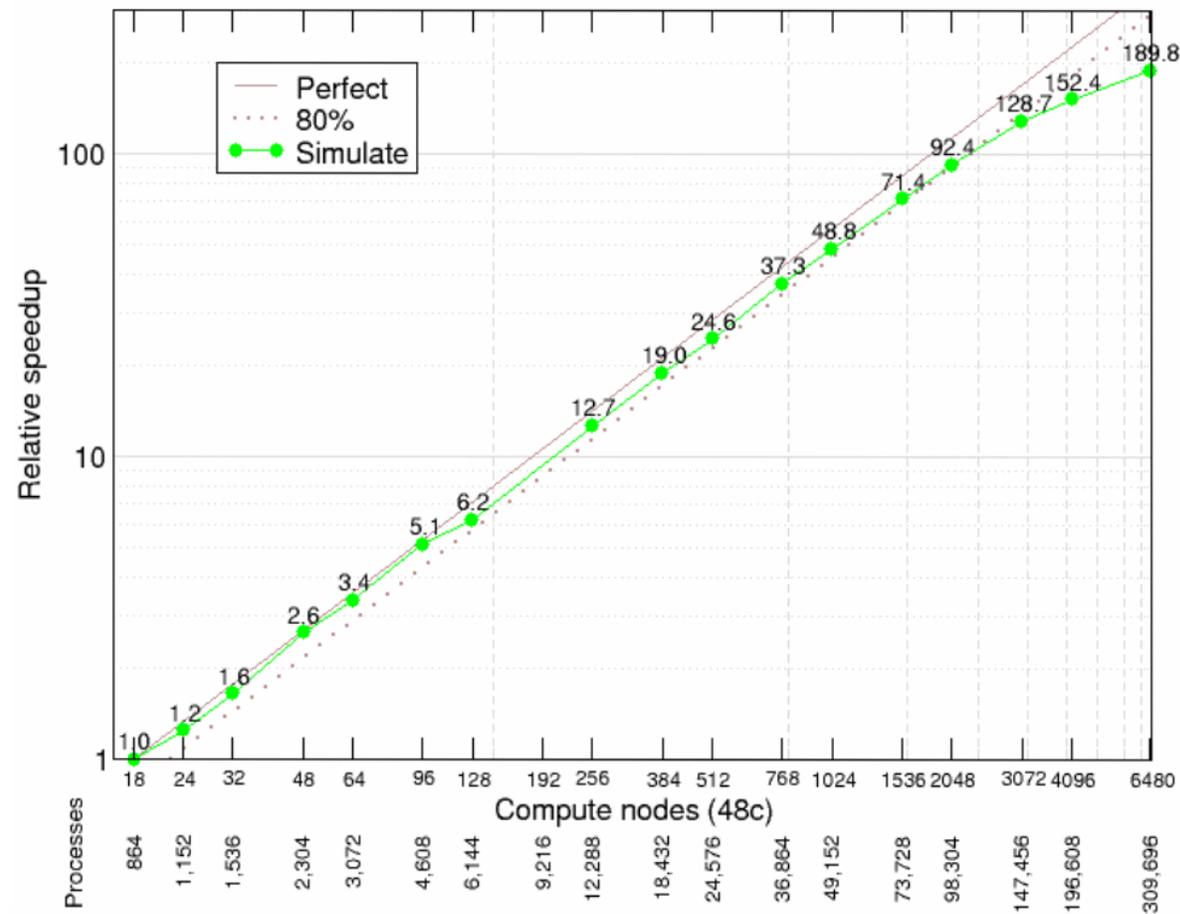
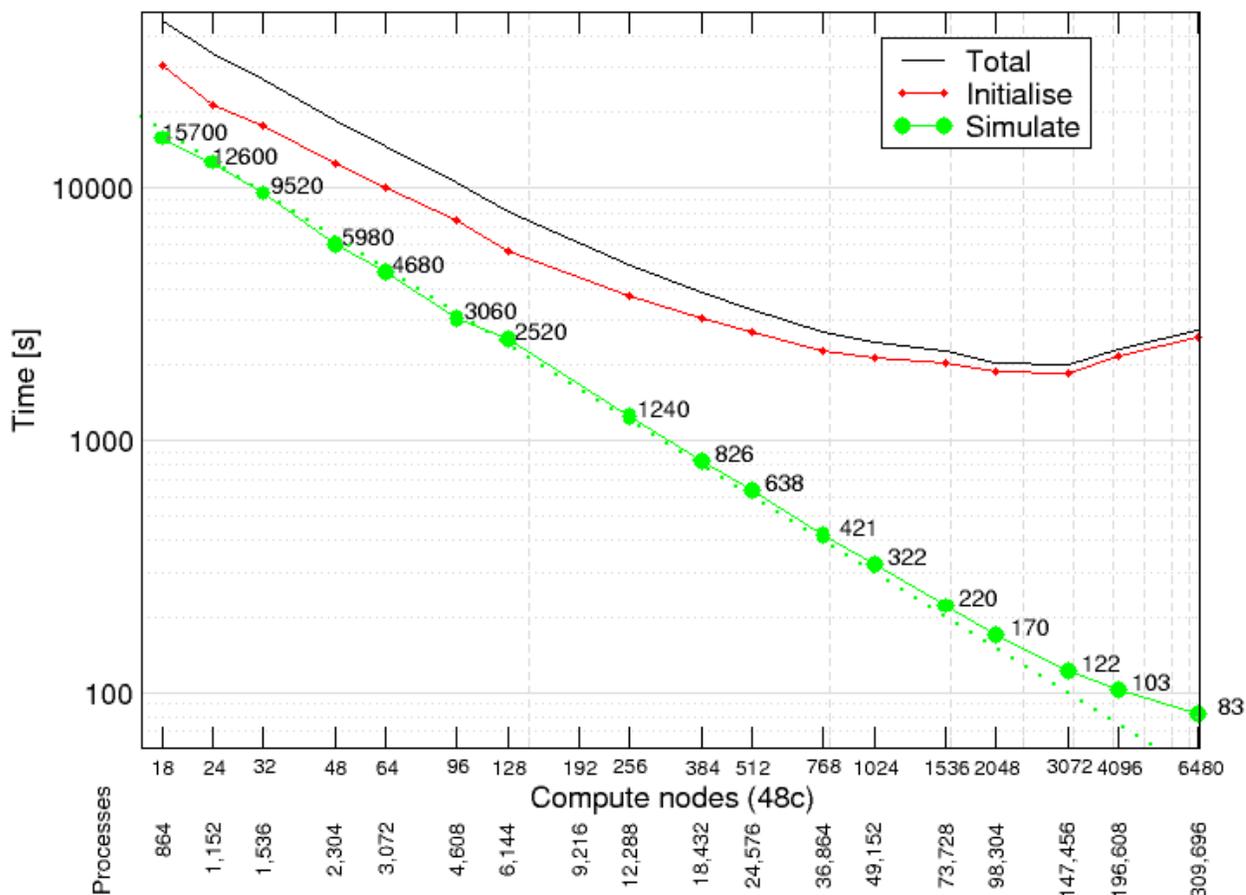
## HemeLB (SuperMUC-NG: no GPUs)

- 3D macroscopic blood flow in human arterial system developed by UC London (UK)
  - lattice-Boltzmann method tracking fluid particles on a lattice grid with complex boundary conditions
  - exascale flagship application of EU H2020 HPC Centre of Excellence for Computational Biomedicine
- HemeLB open-source code and test case: [www.hemelb.org](http://www.hemelb.org)
  - C++ parallelized with MPI [+ CUDA unused]
    - Intel Studio 2019u4 compiler and MPI library (v19.0.4.243)
    - configured with 2 'reader' processes (intermediate MPI file writing disabled)
    - MPI-3 shared-memory model employed within compute nodes
      - to reduce memory requirements when distributing lattice blocks from reader processes
  - Focus of analysis 5,000 time-step (500 $\mu$ s) simulation of cerebrovascular "circle of Willis" geometry
    - 6.4 $\mu$ m lattice resolution (21.15 GiB): 10,154,448,502 lattice sites
- Executed on *SuperMUC-NG* Lenovo ThinkSystem SD650 (LRZ):
  - 2x 24-core Intel Xeon Platinum 8174 ('Skylake') @ 3.1GHz
  - 48 MPI processes/node, 6452 (of 6480) compute nodes: 309,696 MPI processes
  - 190x speed-up from 864 cores: 80% scaling efficiency to over 100,000 cores



⇒ **Identification & quantification of impact of load balance and its variation**

# HemeLB@SNG strong scaling of FOA *RunSimulation*



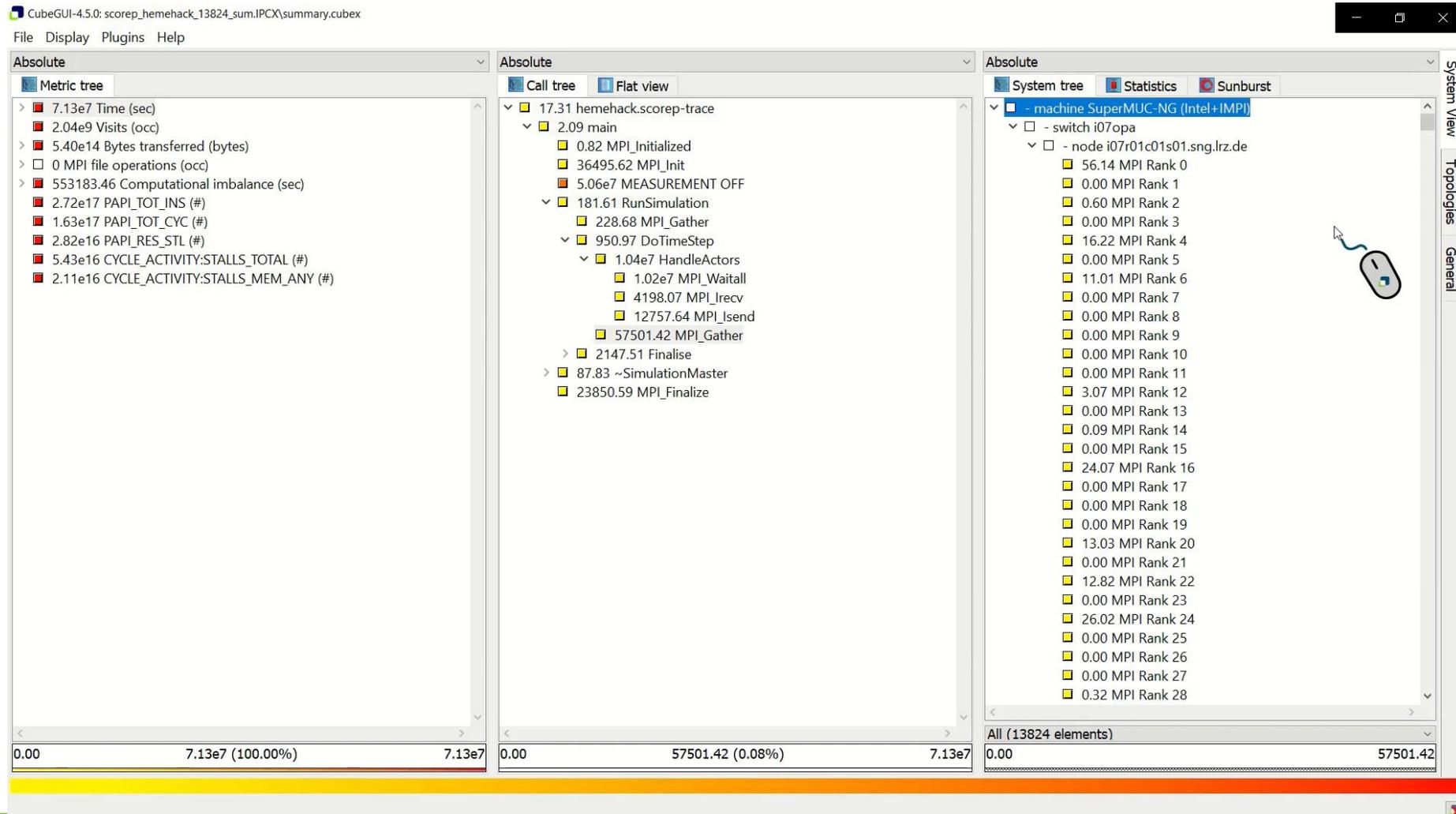
[Execution of 9,216 processes on 192 compute nodes not possible due to insufficient compute nodes with adequate memory in 'fat' partition (768 GiB vs. regular 96 GiB node memory)]



# Initial tree presentation: Time of MPI\_Gather per MPI process



DOI 10.5281/zenodo.4080701





# Advisor: POP efficiency assessment for RunSimulation



CubeGUI-4.5.0: scorep\_hemehack\_13824\_sum.IPCX\summary.cubex

File Display Plugins Help

POP Assessment Runtime threshold: 2.5 % runtime

**Absolute**

Metric tree

- 0.00 Time (sec)
  - 0.00 Execution
    - 6.10e7 Computation
      - 0.00 MPI
        - 60347.38 Management
        - 0.00 Synchronization
        - 0.00 Communication
          - 1.02e7 Point-to-point
            - 59875.44 Collective
            - 0.00 One-sided
            - 0.00 File I/O
          - 0.00 Overhead
        - 2.04e9 Visits (occ)
      - 0 Bytes transferred (bytes)
        - 0 Point-to-point
          - 2.70e14 Sent
          - 2.70e14 Received
        - 3.45e7 Collective
        - 0 Remote Memory Access
      - 0 MPI file operations (occ)
      - 553183.46 Computational imbalance (sec)
        - 2.72e17 PAPI\_TOT\_INS (#)
        - 1.63e17 PAPI\_TOT\_CYC (#)
        - 2.82e16 PAPI\_RES\_STL (#)
        - 5.43e16 CYCLE\_ACTIVITY:STALLS\_TOTAL (#)
        - 2.11e16 CYCLE\_ACTIVITY:STALLS\_MEM\_ANY (#)

0 2.11e16 (100.00%) 2.11e16

**Absolute**

Call tree Flat view

- 9.20e8 hemehack.scorep-trace
  - 9.57e8 main
    - 6.53e7 MPI\_Initialized
    - 1.64e11 MPI\_Init
    - 1.89e16 MEASUREMENT OFF
    - 2.22e15 RunSimulation
    - 5.32e10 ~SimulationMaster
    - 3.16e12 MPI\_Finalize

0 2.22e15 (10.50%) 2.11e16

Advisor Score-P Configuration Source

Recalculate  automatic

POP Assessment : SimulationMaster::RunSimulation

Parallel Efficiency	50%	Fair	0.50	?
Load balance	50%	Fair	0.50	?
Communication Efficiency	99%	Very good	1.00	?
Serialisation Efficiency	0%			!
Transfer efficiency	0%			!
Stalled Resources	57%	Fair	0.58	?
IPC		Value	1.56	?
Instructions (only computation)		Value	3.72e16	?
Computation time		Value	1.04e7	?

Candidates

Callpath	Issue

Ready

## HemeLB (JUWELS-Volta)

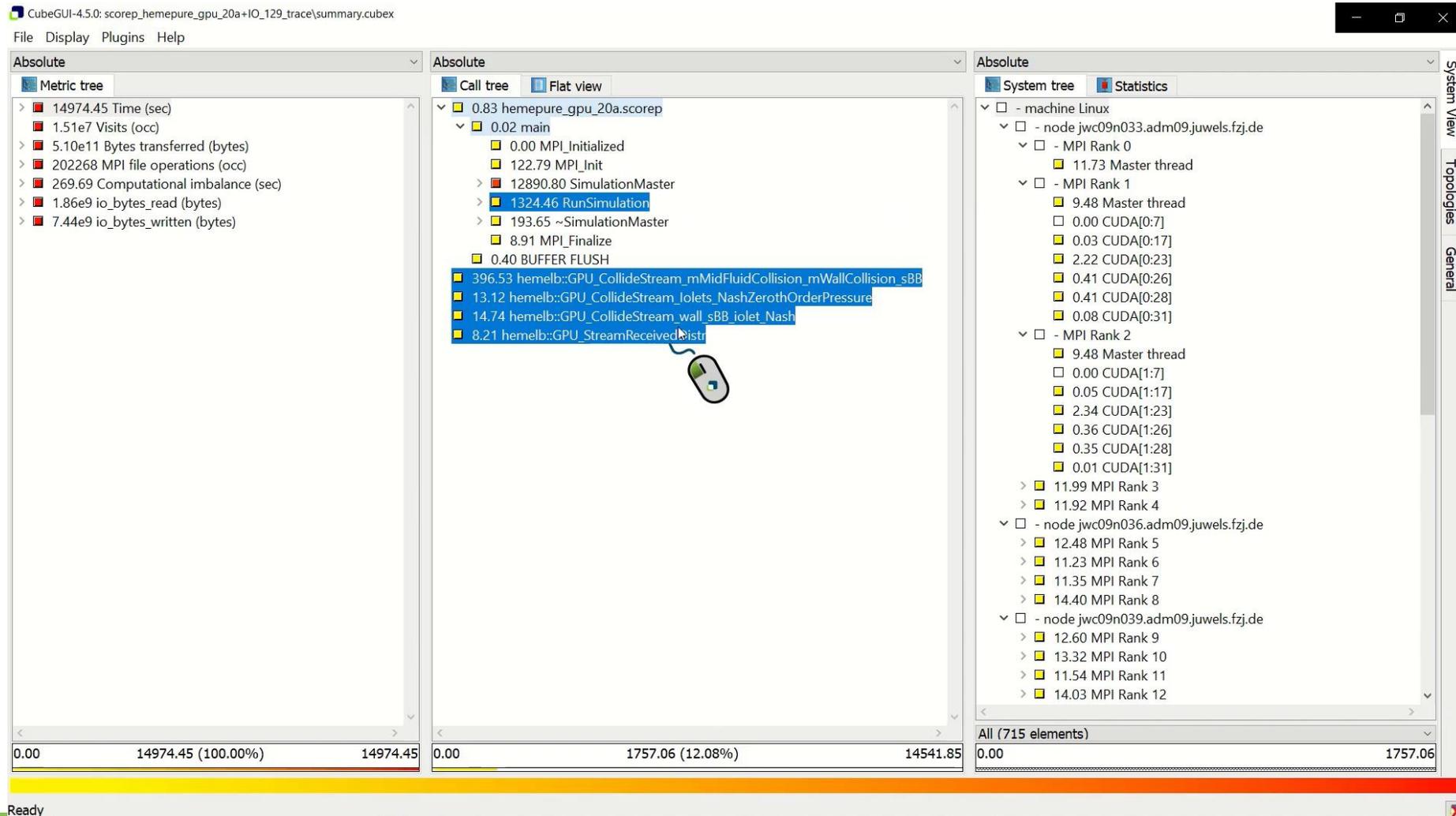
- 3D macroscopic blood flow in human arterial system developed by UC London (UK)
    - lattice-Boltzmann method tracking fluid particles on a lattice grid with complex boundary conditions
    - exascale flagship application of EU H2020 HPC Centre of Excellence for Computational Biomedicine
  - HemeLB open-source code and test case: [www.hemelb.org](http://www.hemelb.org)
    - C++ parallelized with MPI + CUDA (in development)
      - GCC/8.3.0 compiler, CUDA/10.1.105 and ParaStationMPI/5.4 library
      - configured with 2 'reader' processes and intermediate MPI file writing
      - rank 0 'monitor' process doesn't participate in simulation
    - Focus of analysis 2,000 time-step (each 100 $\mu$ s) simulation of CBM2019\_Arteries\_patched geometry
      - 1.78 GiB: 66,401,494 lattice sites, 1+38 iolets
  - Executed on *JUWELS-Volta* (@JSC):
    - 2x 20-core Intel Xeon Platinum 8168 ('Skylake') CPUs + 4 Nvidia V100 'Volta' GPUs
    - 4\* MPI processes/node (one per GPU), 32 (of 56) compute nodes: 129 MPI processes
- $\Rightarrow$  Identification & quantification of impact of load balance and its variation**



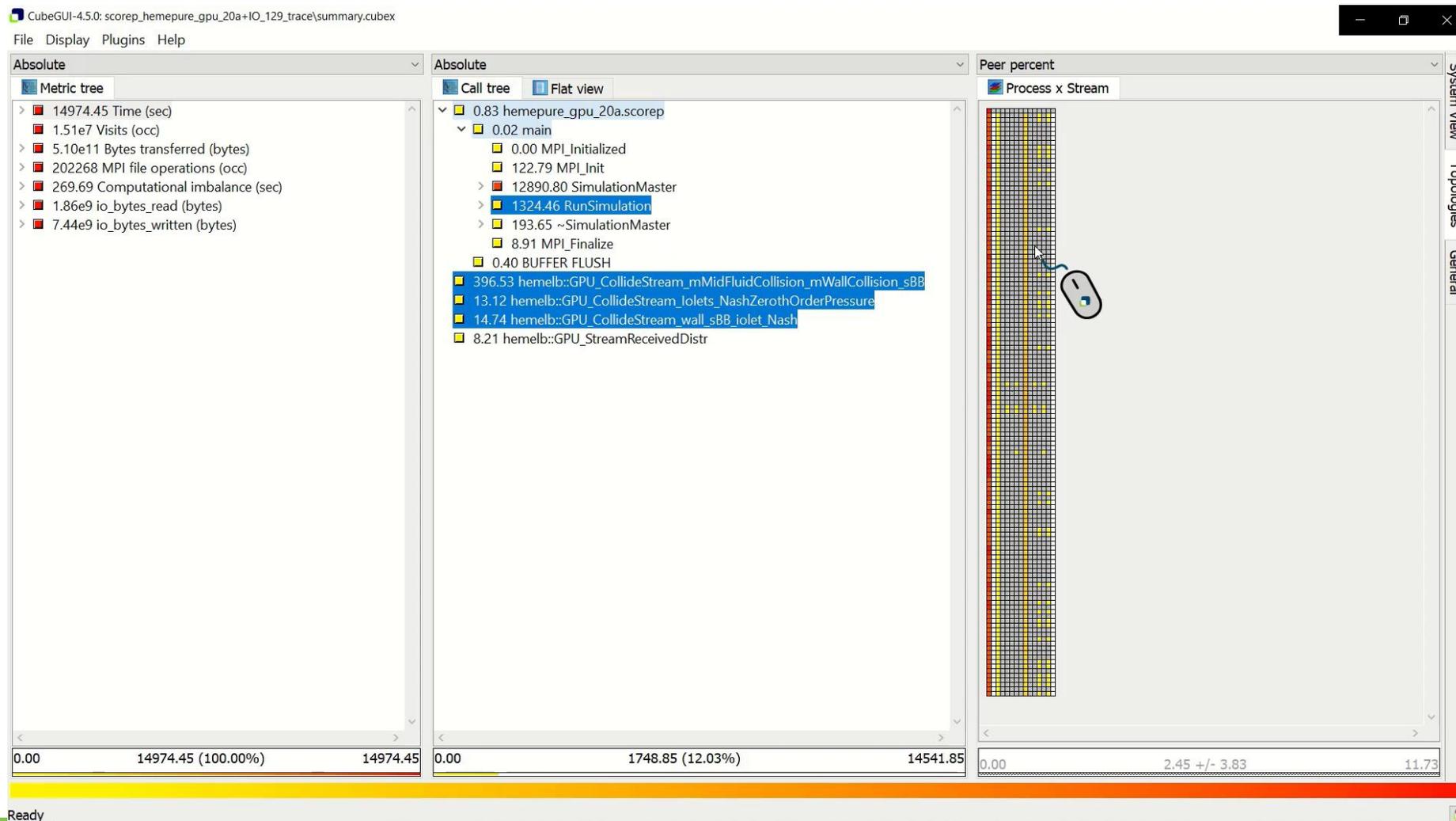
# Tree: Time for asynch. CUDA kernels on separate CUDA streams



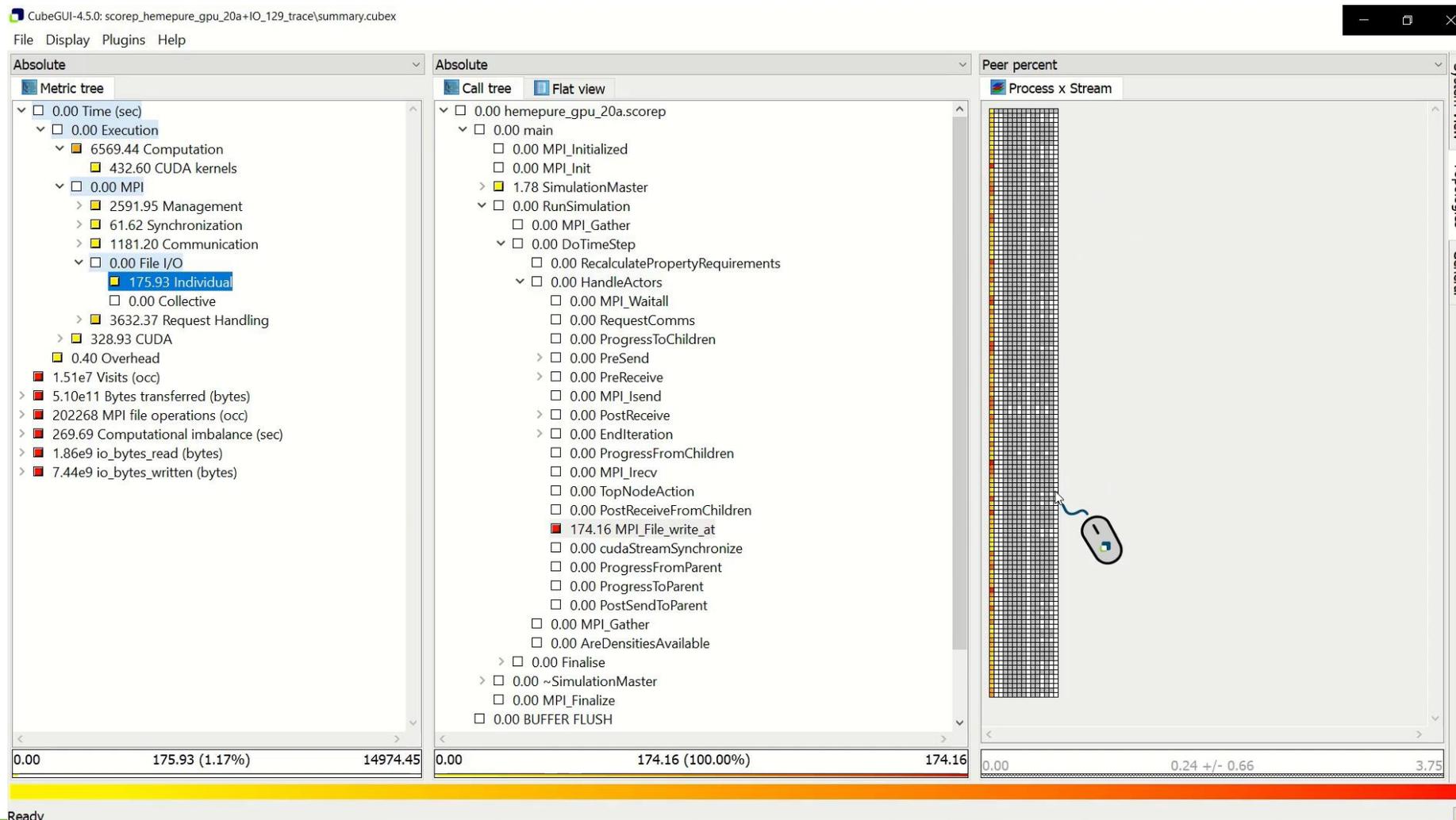
DOI 10.5281/zenodo.4081080



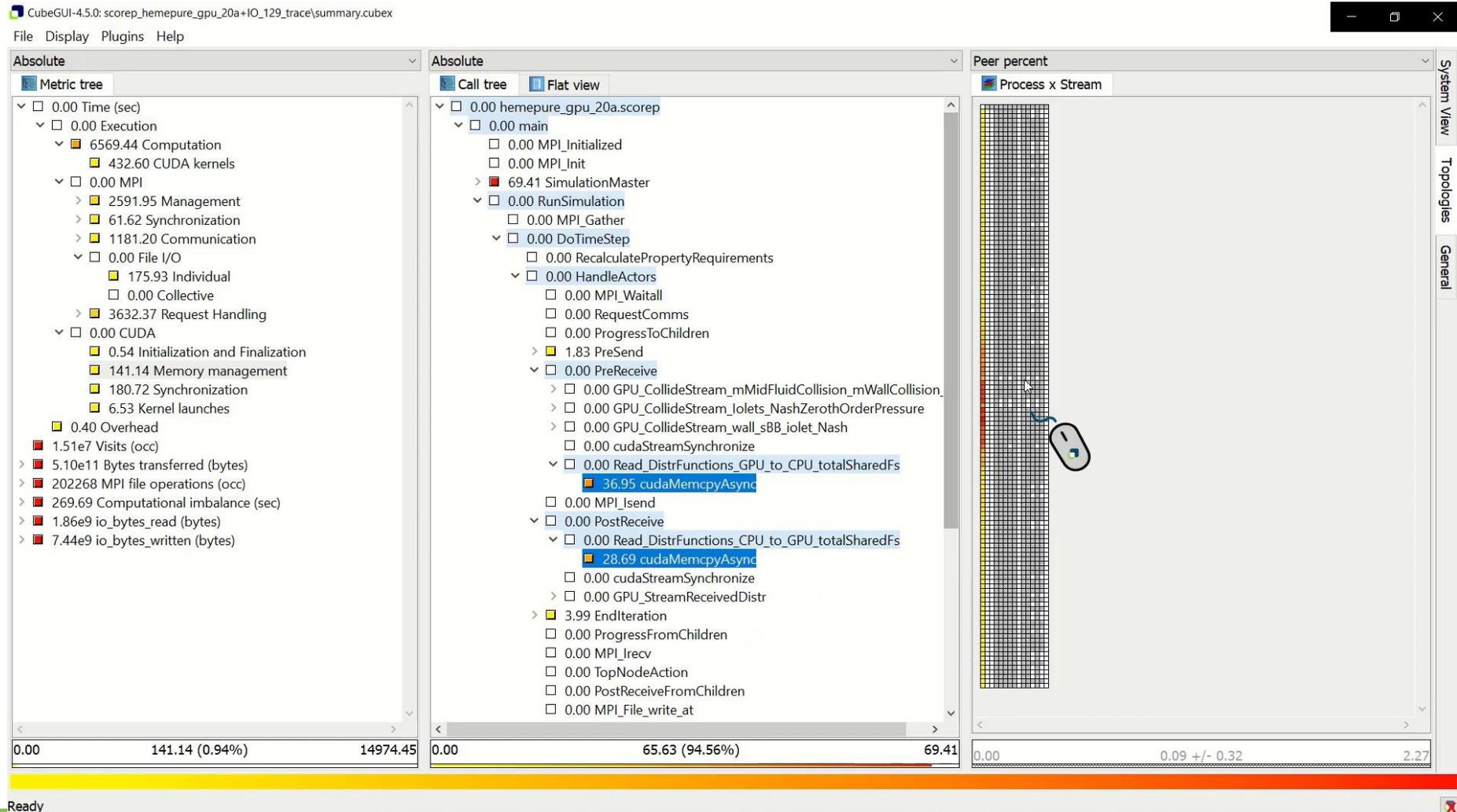
# Topo: Time for asynch. CUDA kernels on separate CUDA streams



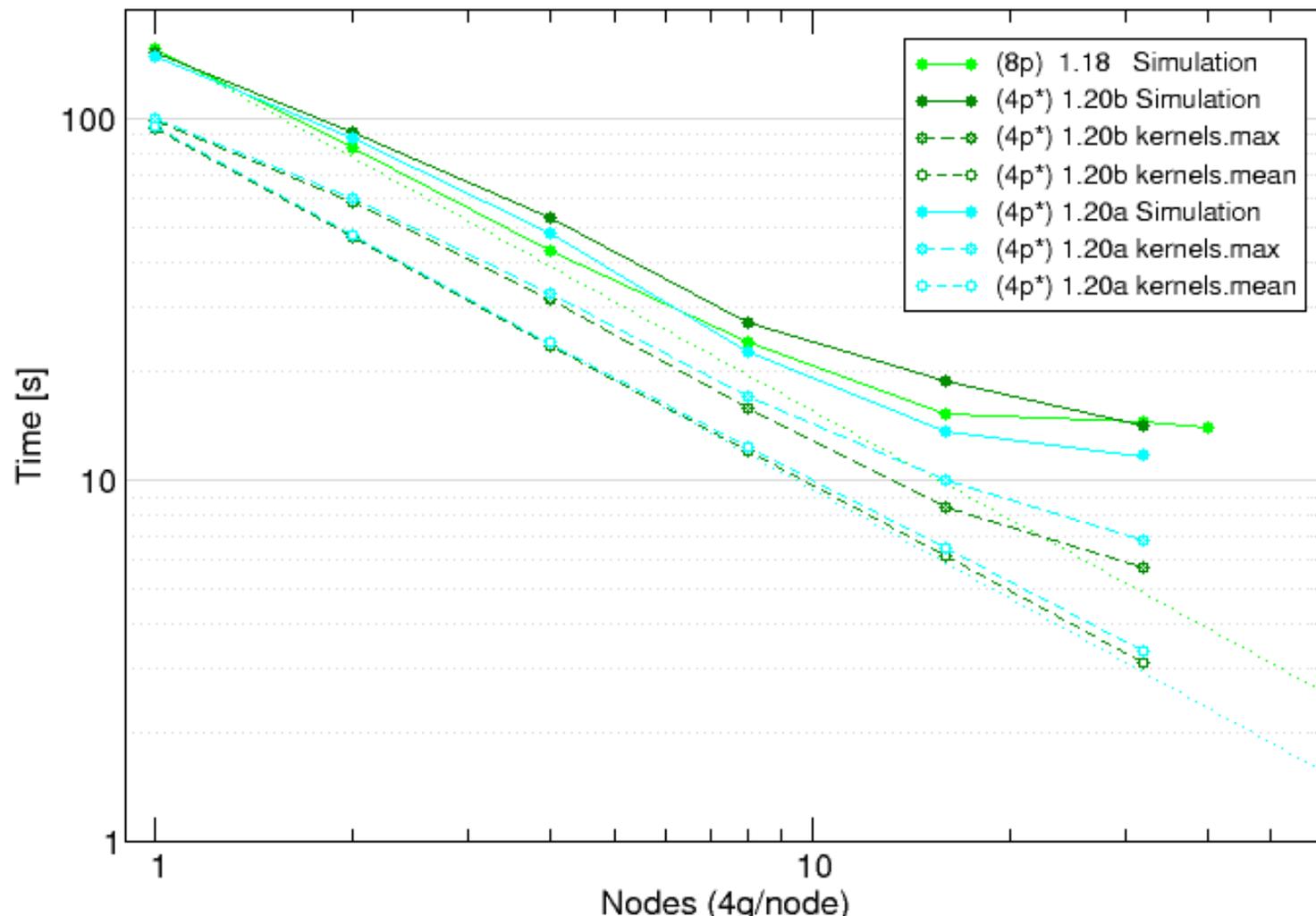
# Topo: Time for MPI file writing on CPU varies per MPI process



# Topo: Time for CUDA asynchronous memory copies is imbalanced



## HemeLB@JUWELS-Volta strong scaling of FOA *RunSimulation*



- Reference execution with 8ppn
  - multiple processes offloading GPU kernels generally unproductive
- Comparison of versions (4ppn)
  - v1.20a generally better
- Synchronous MPI file writing is the primary bottleneck
- CUDA kernels on GPUs
  - less than half of Simulation time (therefore GPUs mostly idle)
  - total kernel time scales very well (0.93 scaling efficiency)
  - load balance deteriorates (0.95 for single node, 0.50 for 32 nodes)

## HemeLB@JUWELS/Volta strong scaling efficiency of *RunSimulation*

	1n 5p	2n 9p	4n 17p	8n 33p	16n 65p	32n 129p	Key:
Simulation time [s]	147.87	88.38	48.13	22.66	13.68	11.67	1.1
Global scaling efficiency	0.64	0.53	0.49	0.52	0.43	0.25	1.0
– Parallel efficiency	0.64	0.53	0.50	0.54	0.47	0.29	0.9
– – Load balance efficiency (GPU)	0.95	0.78	0.73	0.73	0.65	0.50	0.8
– – Communication efficiency (GPU)	0.67	0.68	0.68	0.75	0.73	0.58	0.7
– Computation scaling (GPU)	1.00	1.00	0.99	0.96	0.92	0.87	0.6
							0.5
							0.4
							0.3
							0.2
							0.1
							0.0

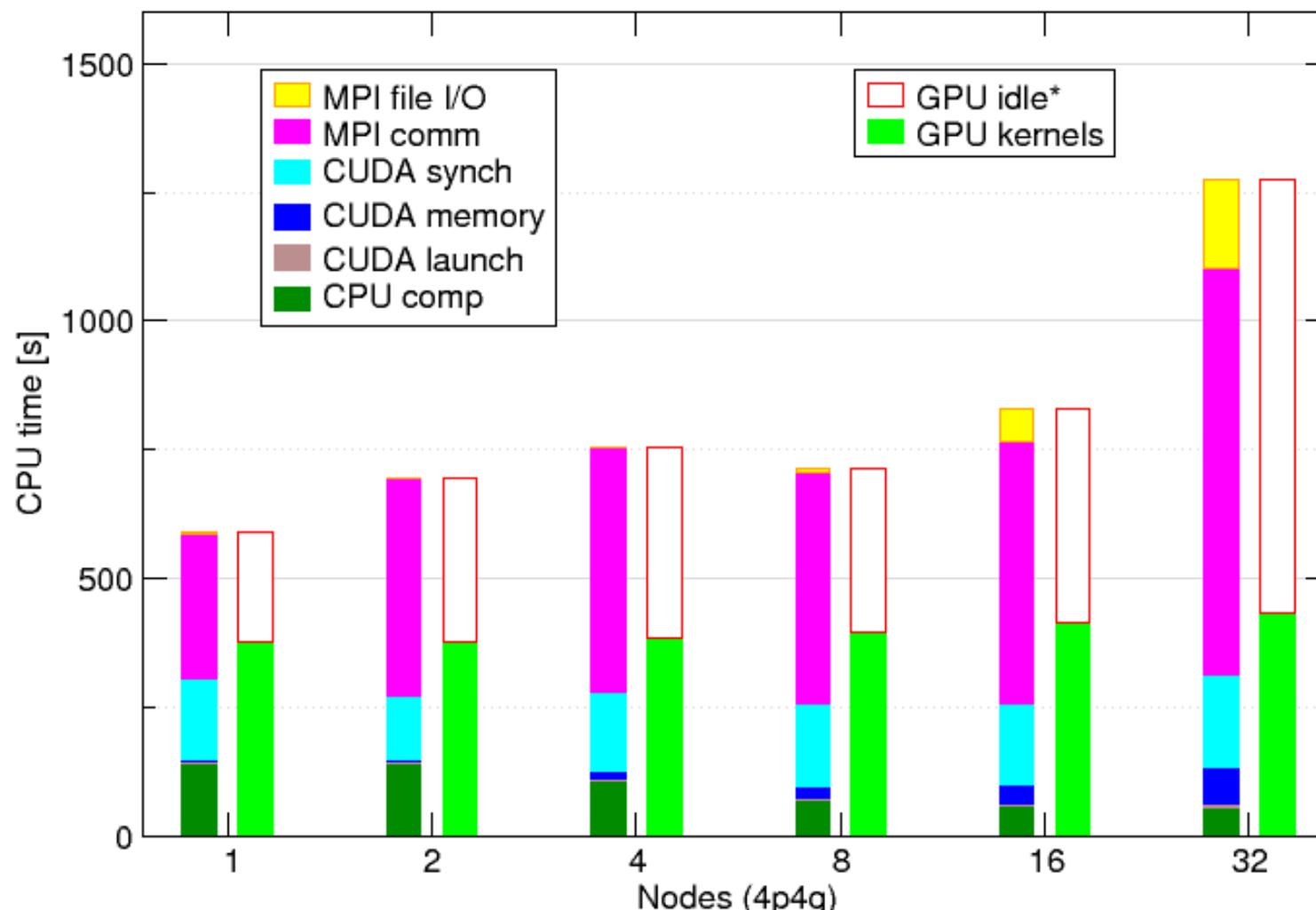
Only considering GPUs (ignoring all CPU cores, 90% of which are completely unused)

- Single (quad-GPU) node already suffers significant communication inefficiency
  - includes MPI file writing, but doesn't degrade much as additional nodes are included
- Load balance of GPUs deteriorates progressively
- GPU computation scaling remains reasonably good

[POP CoE scaling efficiency model: [www.pop-coe.eu](http://www.pop-coe.eu)]



## HemeLB@JUWELS-Volta strong scaling of FOA *RunSimulation*



- CPU+GPU time breakdown
- CUDA kernels on GPUs
  - less than half of Simulation time (therefore GPUs mostly idle)
  - total kernel time scales very well (0.87 scaling efficiency)
- MPI processes on CPUs
  - computation time decreases
  - CUDA synchronization time fairly constant, but time for memory management increases somewhat
  - MPI communication time dominates, with much more time for file writing with 16+ nodes