Analysis report examination with Cube

Anke Visser & Brian Wylie Jülich Supercomputing Centre







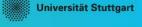






















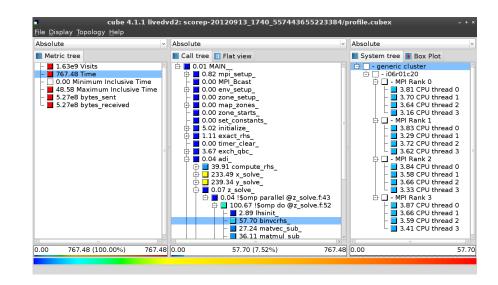






Cube

- 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 ≥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 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
```

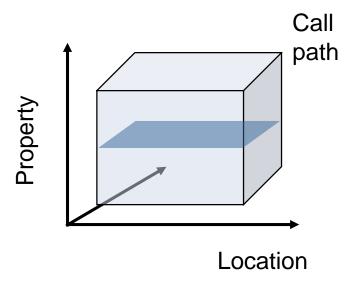
- Install & run *local* (recommended)
 - install Cube GUI locally on desktop
 - binary packages available for MacOS & Windows and externally provided by OpenHPC and various Linux distributions
 - source package available for Linux, requires Qt
 - configure/build/install manually or use your favourite 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
```



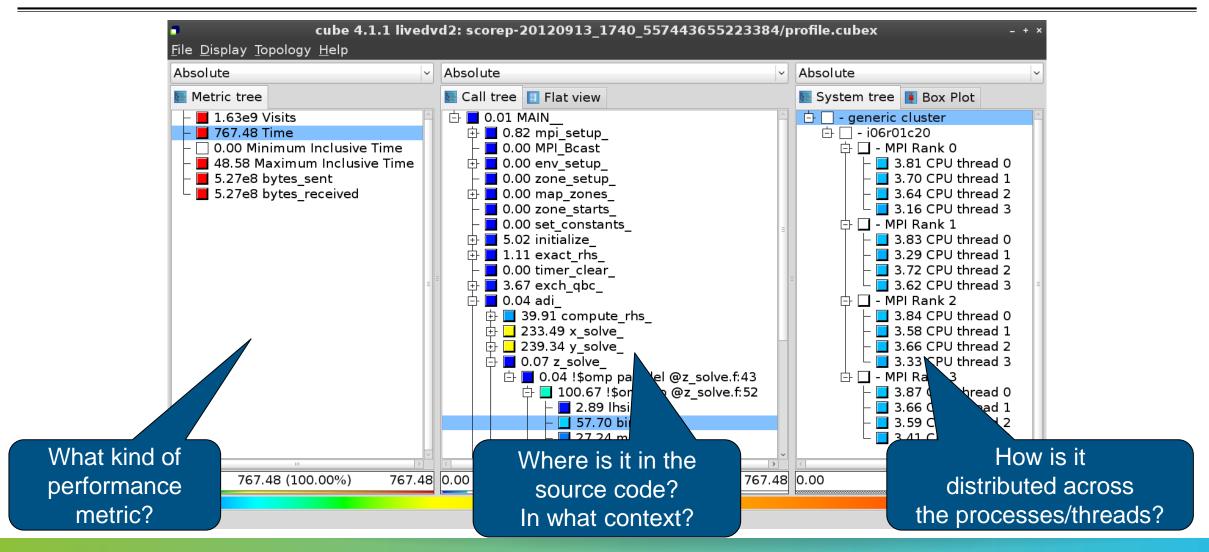
Analysis presentation and exploration

- 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

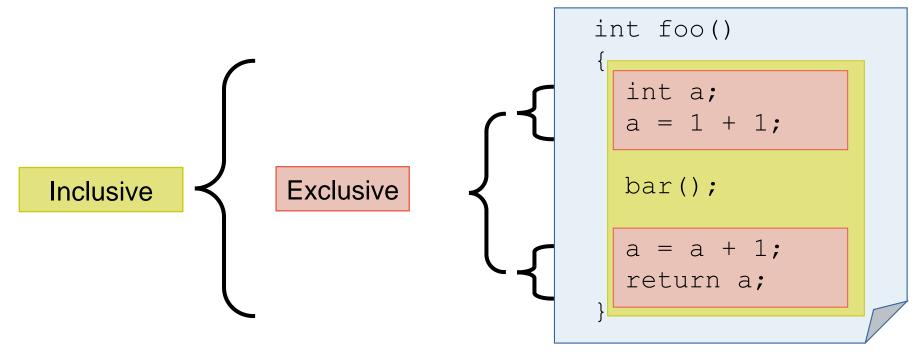




Inclusive vs. exclusive values

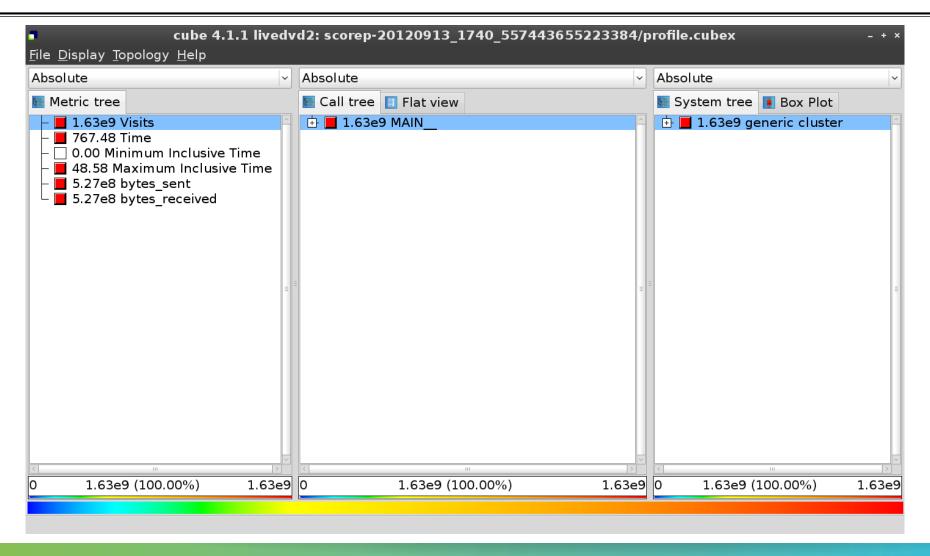


- Inclusive
 - Information of all sub-elements aggregated into single value
- Exclusive
 - Information cannot be subdivided further



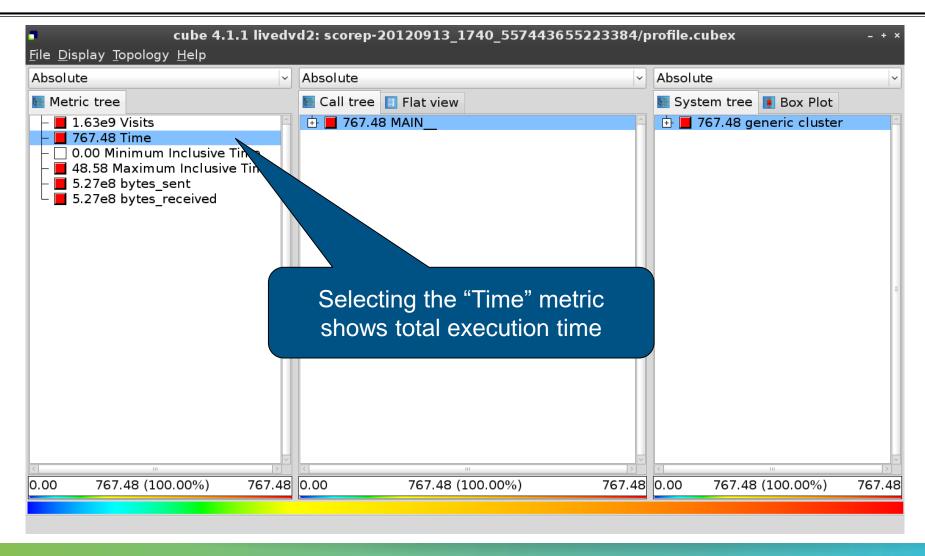
Score-P analysis report exploration (opening view)





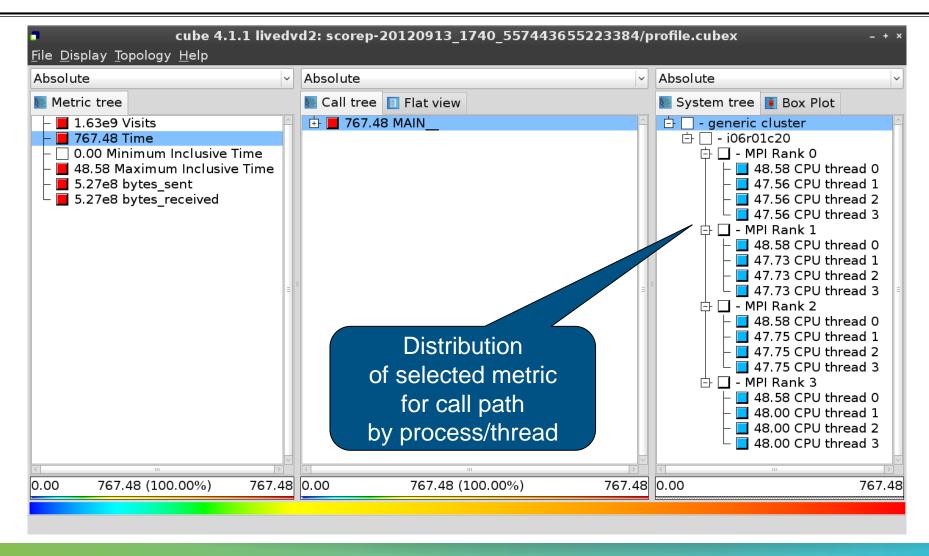
Metric selection





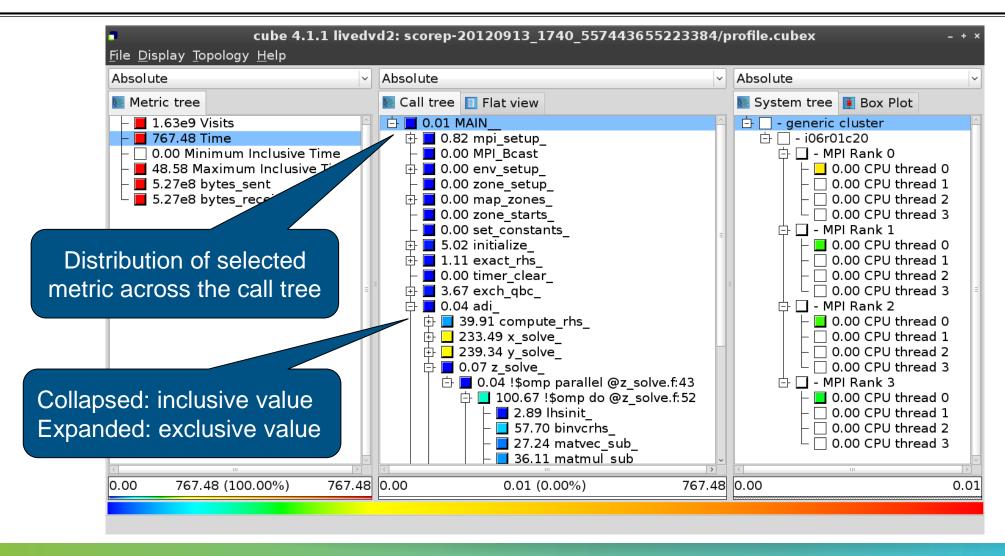
Expanding the system tree





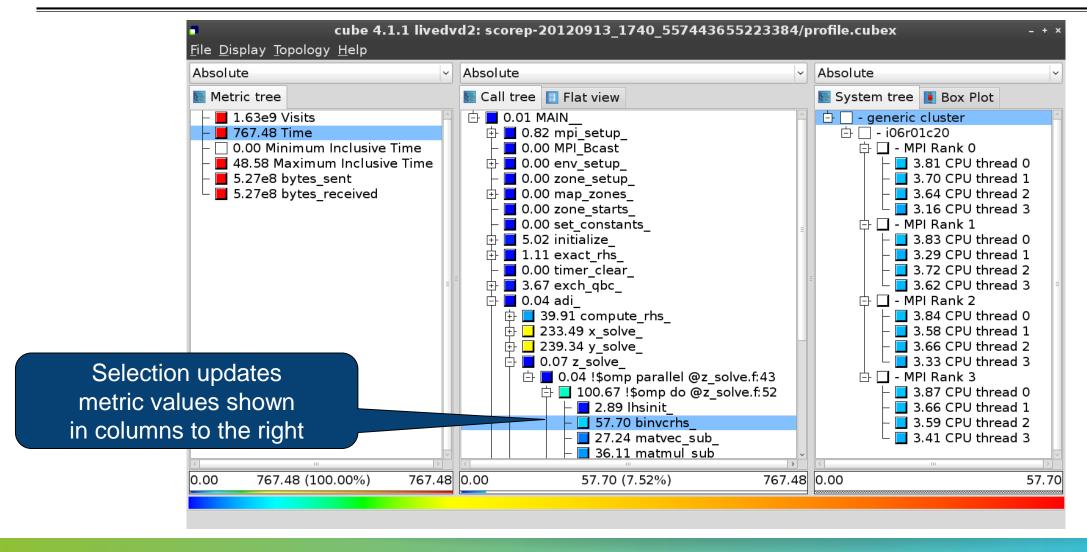
Expanding the call tree





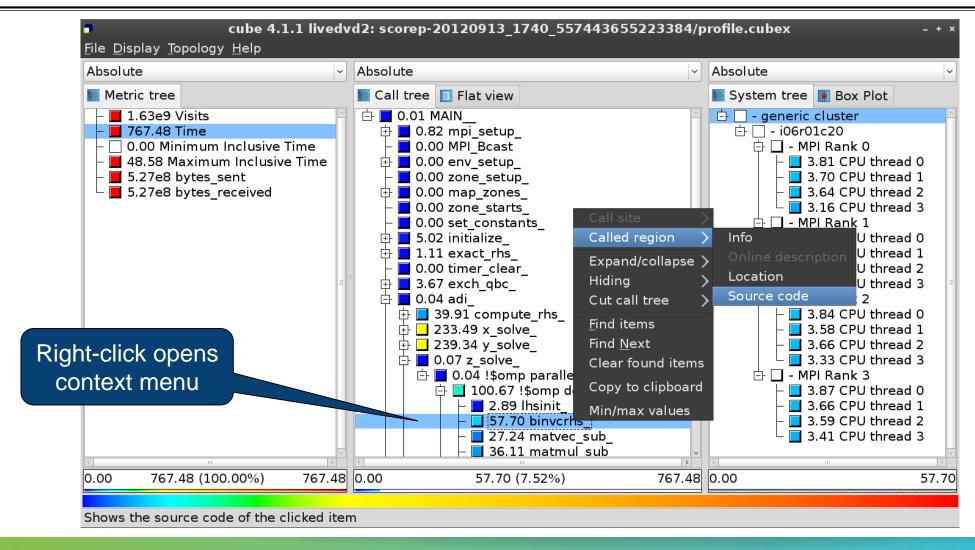
Selecting a call path





Source-code view via context menu

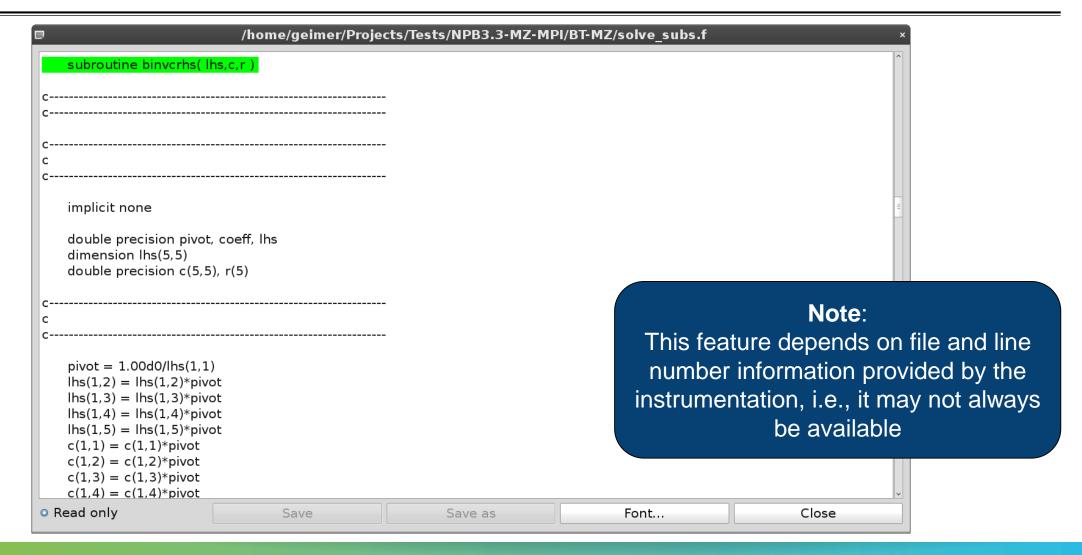






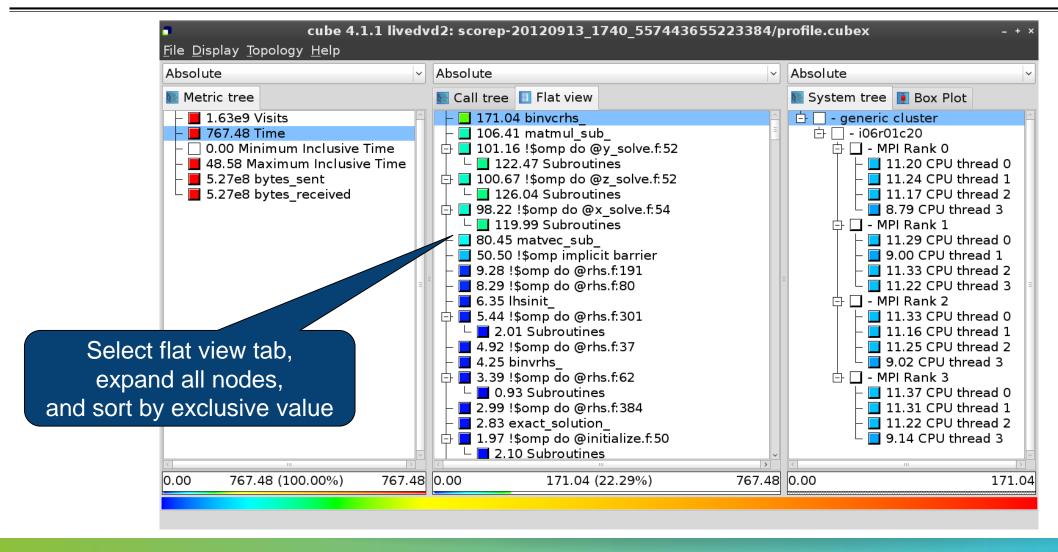
Source-code view





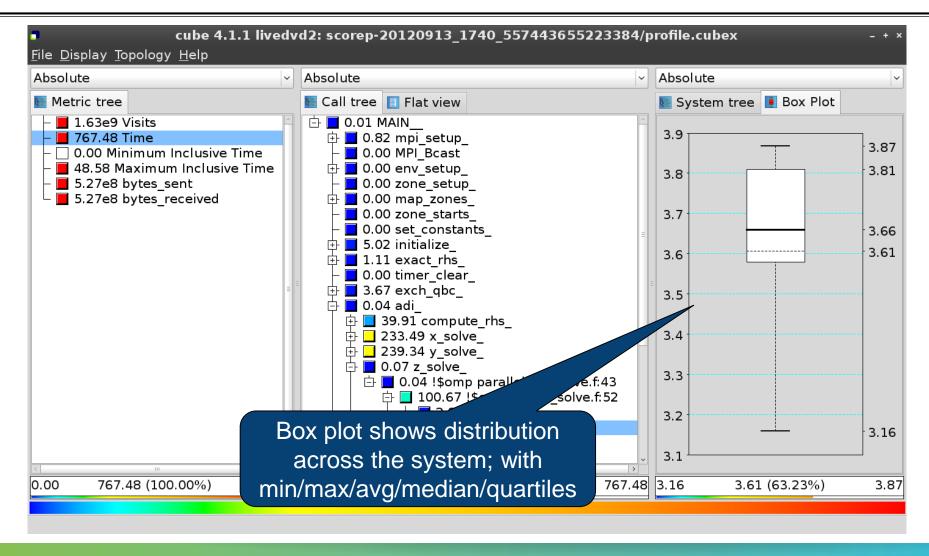
Flat profile view





Box plot view

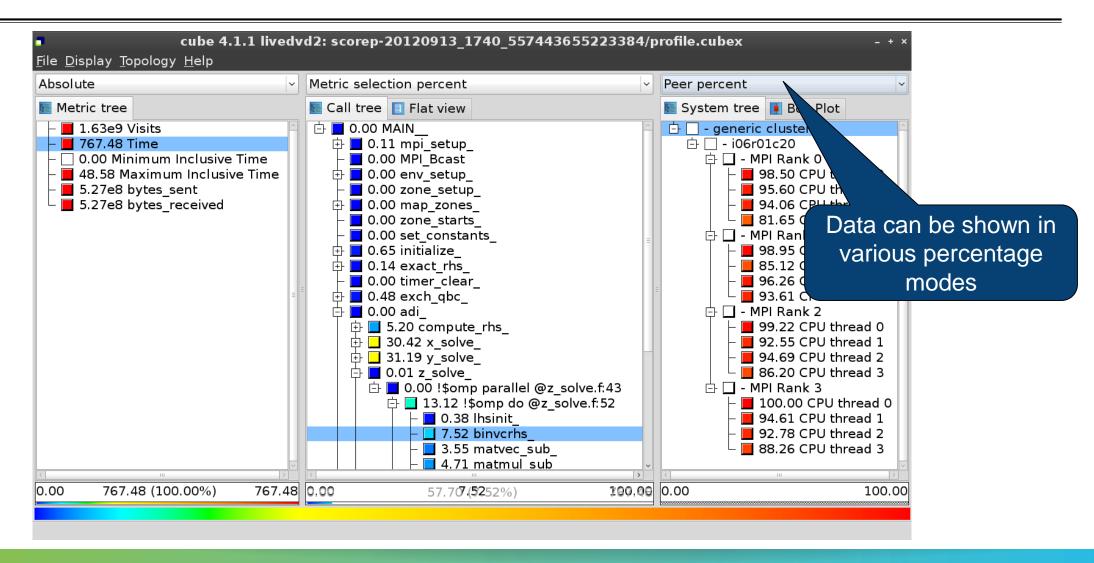






Alternative display modes





Important display modes

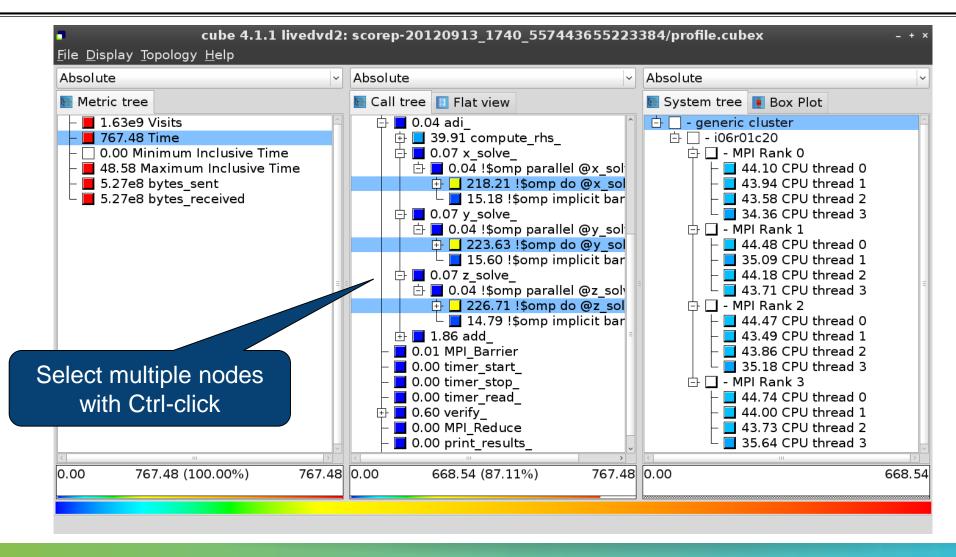


- Absolute
 - Absolute value shown in seconds/bytes/counts
- Selection percent
 - Value shown as percentage w.r.t. the selected node "on the left" (metric/call path)
- Peer percent (system tree only)
 - Value shown as percentage relative to the maximum peer value



Multiple selection

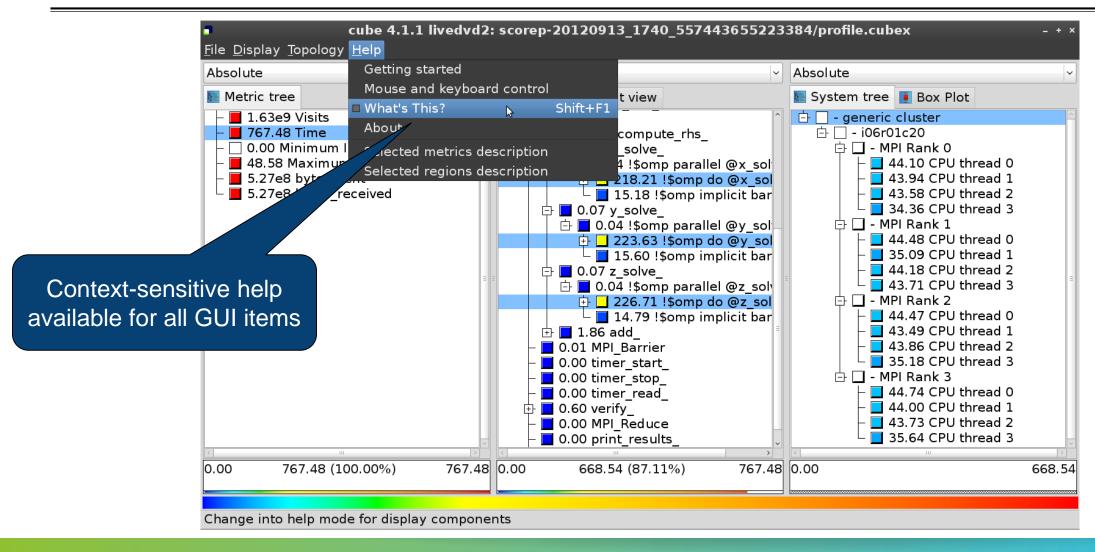






Context-sensitive help





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

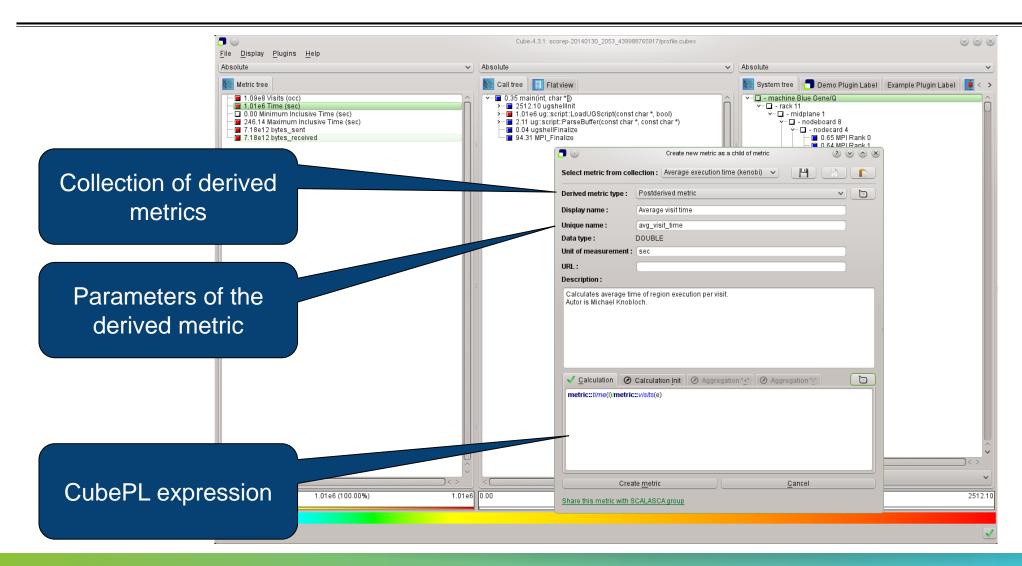
metric::time(i)/metric::visits(e)

"Number of FLOP per second": Postderived metric with expression

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

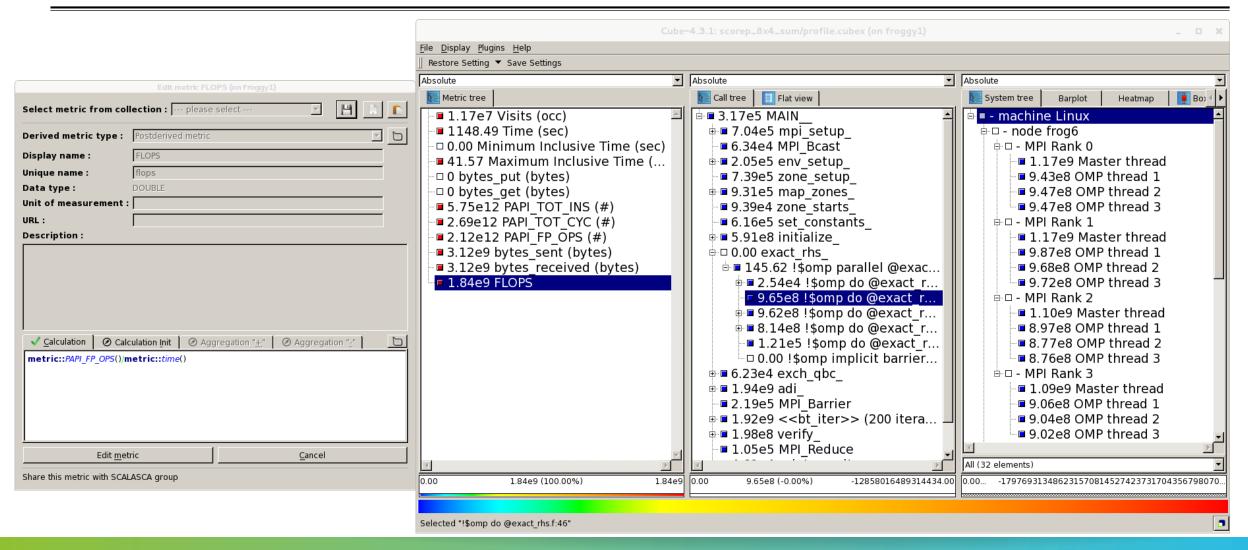
Derived metrics in Cube GUI





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

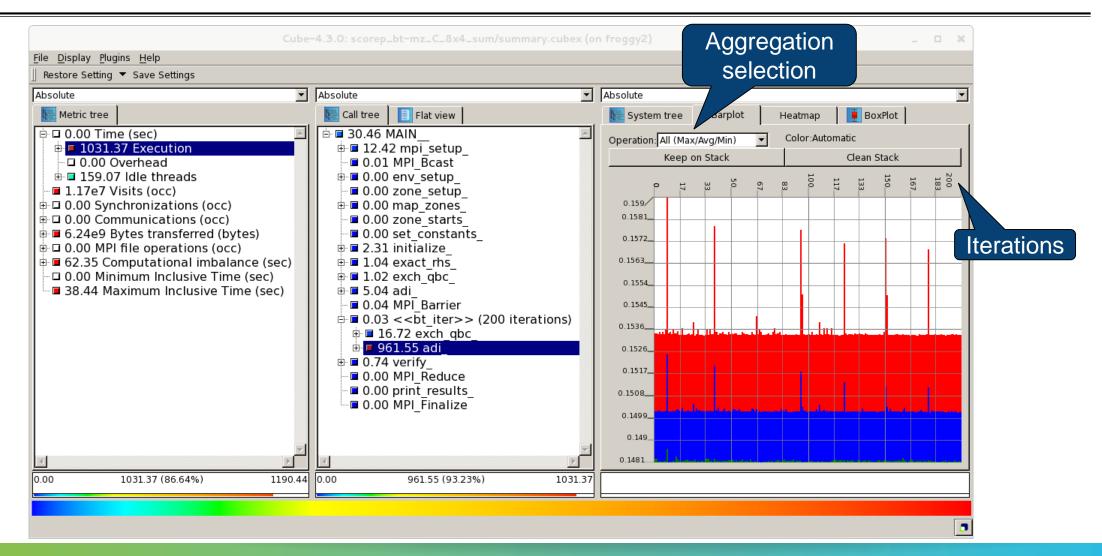
or

- 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

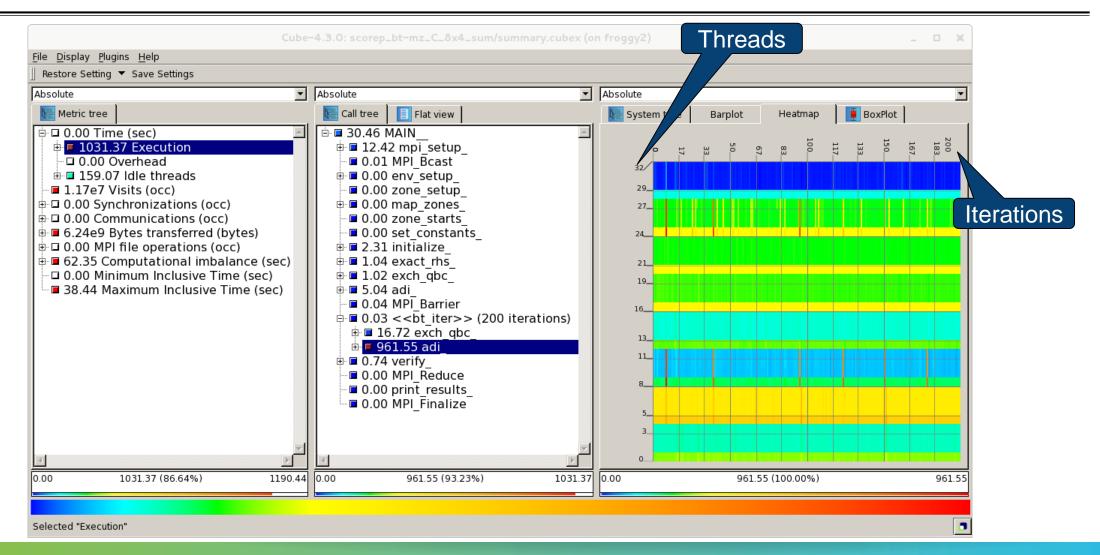




√i-HPS

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



Score-P/CUBE case study HemeLB





























HemeLB (SuperMUC-NG: no GPUs)

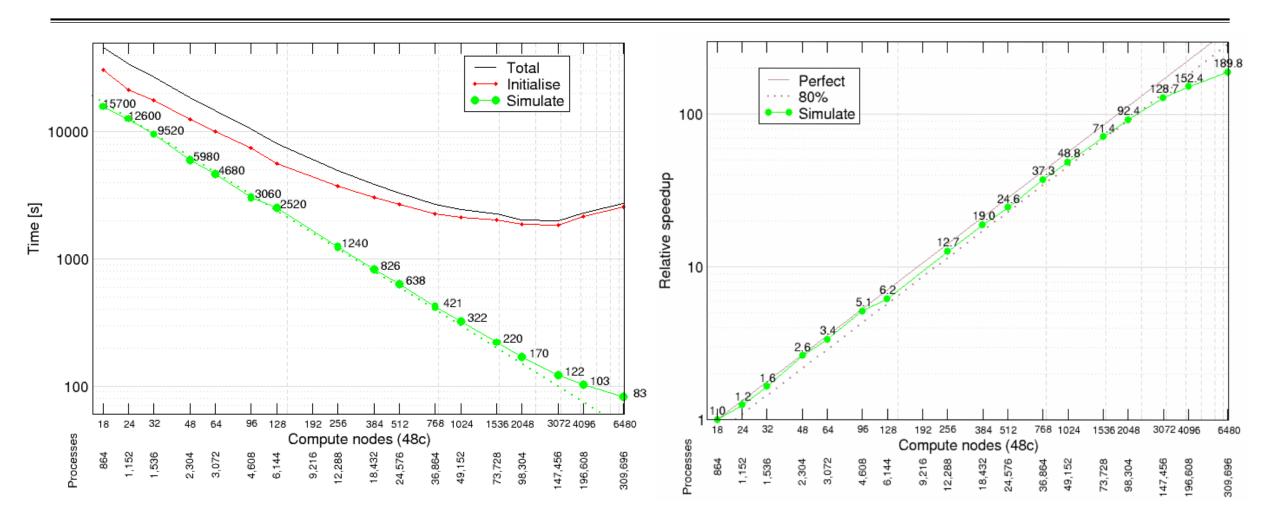


- 3D macroscopic blood flow in human arterial system developed by UC London (UK)
 - lattice-Boltzmann method tracking fluid particles on a lattice grid with complex boundary conditions
 - exascale flagship application of EU H2020 HPC Centre of Excellence for Computational Biomedicine
- Hemelb open-source code and test case: www.hemelb.org
 - 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 (500µs) simulation of cerebrovascular "circle of Willis" geometry
 - 6.4µm 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)



HemeLB@SNG strong scaling efficiency of FOA RunSimulation

Compute nodes	24	32	48	64	96	128	192	256	384	512	768	1024	1536	2048	3072	4096	6452
Processes	1152	1536	2304	3072	4608	6144	9216	12288	18432	24576	36864	49152	73728	98304	147456	196608	309696
Global scaling efficiency	0.79	0.79	0.84	0.80	0.82	0.75		0.73	0.72	0.73	0.74	0.68	0.68	0.65	0.62	0.57	0.45
- Parallel efficiency	0.79	0.80	0.87	0.83	0.86	0.80		0.75	0.74	0.74	0.77	0.71	0.72	0.70	0.72	0.70	0.73
Load balance efficiency	0.79	0.80	0.88	0.84	0.86	0.80		0.75	0.74	0.75	0.78	0.72	0.74	0.72	0.74	0.73	0.80
Communication efficiency	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	0.99	0.99	0.99	0.98	0.98	0.97	0.96	0.92
- Computation scaling	1.00	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.98	0.96	0.96	0.94	0.93	0.87	0.81	0.61
Instructions scaling	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	1.00	0.99	0.97	0.94	0.89	0.79	0.67	0.45
IPC scaling	1.00	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.99	0.98	0.99	1.00	1.04	1.11	1.21	1.36
IPC	1.411	1.395	1.353	1.355	1.342	1.316		1.377	1.387	1.396	1.383	1.390	1.417	1.473	1.566	1.704	1.919
											Key:	<0.65	<0.75	<0.85	<0.95	<1.00	>1.00

Global scaling efficiency fairly good around 80%, before degrading at larger scales

- Parallel efficiency deteriorating following Load balance efficiency
 - Communication efficiency excellent throughout
- Computation scaling (relative to 1152 processes) very good except at largest scale
- Degradation of Instructions scaling partially compensated by improving IPC scaling
 [POP CoE scaling efficiency model: www.pop-coe.eu]



HemeLB@SNG instrumentation & measurement

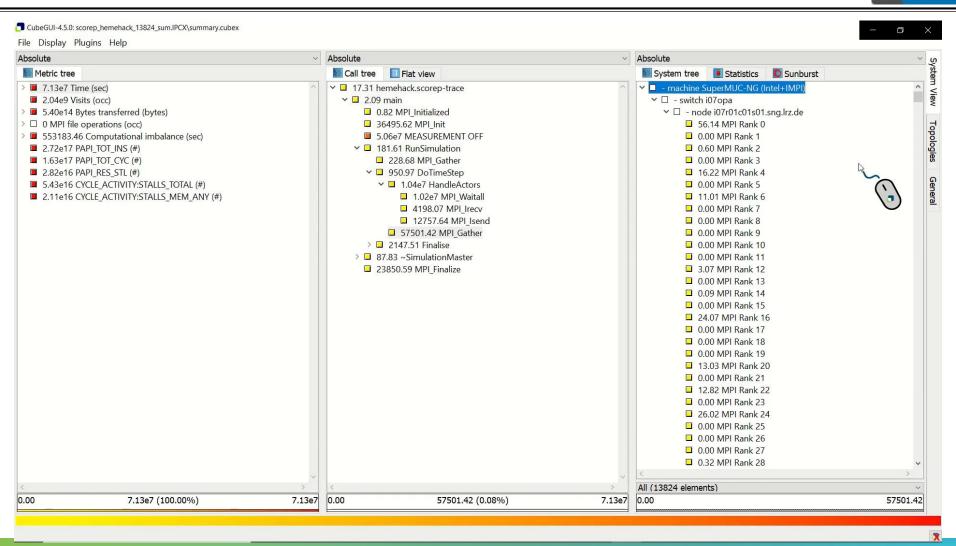


- Score-P/5.0 configured with MPIFLAGS=-DSCOREP_MPI_NO_MINI
 - don't generate wrappers for MPI_Comm_rank & MPI_Comm_size
- HemeLB instrumentation
 - application dependencies built without modification (or instrumentation)
 - src/main.cc annotated with SCOREP_RECORDING_OFF/SCOREP_RECORDING_ON macros
 - to pause measurement recording during initialization
 - configured using CXX=scorep-icpc with
 - SCOREP WRAPPER INSTRUMENTER FLAGS="--user --mpp=mpi --thread=none --nomemory"
 - SCOREP_WRAPPER_COMPILER_FLAGS="-tcollect-filter=<path_to_file>"
 - disable Intel compiler instrumentation apart from main routine and key classes such as SimulationMaster
- HemeLB execution configuration
 - SLURM --ear=off to enable 'performance' profile of processor and access to hardware counters
 - SCOREP_DEVELOPMENT_MEMORY_STATS=aggregated # report Score-P memory usage
 - SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_REF_CYC,PAPI_RES_STL
 - SCOREP_TIMER=gettimeofday # use globally synchronized clock
 - SCOREP_TOTAL_MEMORY=200MB # as determined from Score-P memory usage stats

Initial tree presentation: Time of MPI_Gather per MPI process



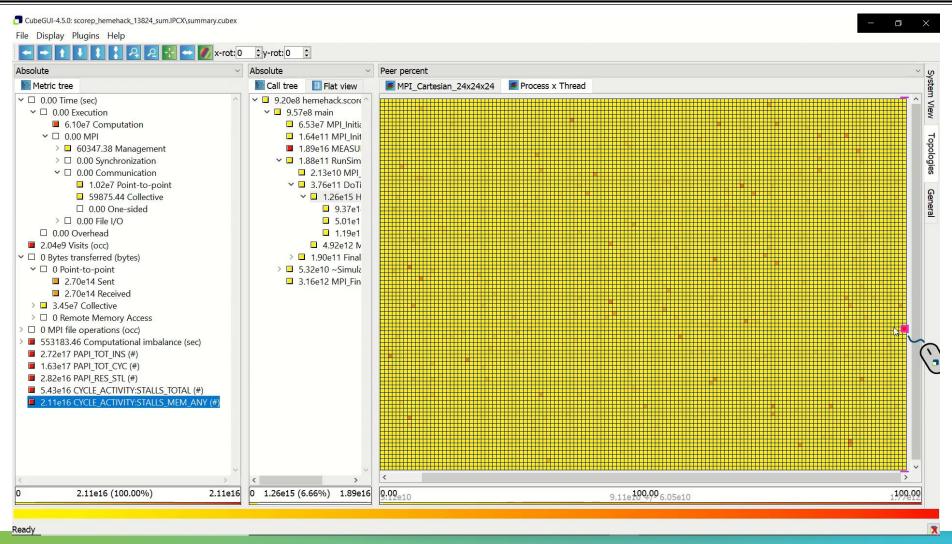
DOI 10.5281/zenodo.4080701





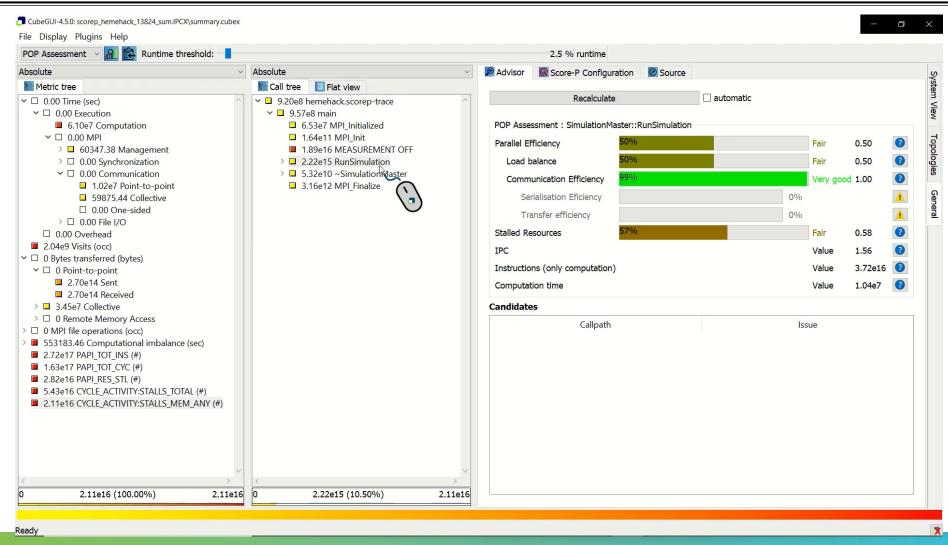
Topological presentation: STALLS_MEM_ANY for HandleActors





Advisor: POP efficiency assessment for RunSimulation





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
 - 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

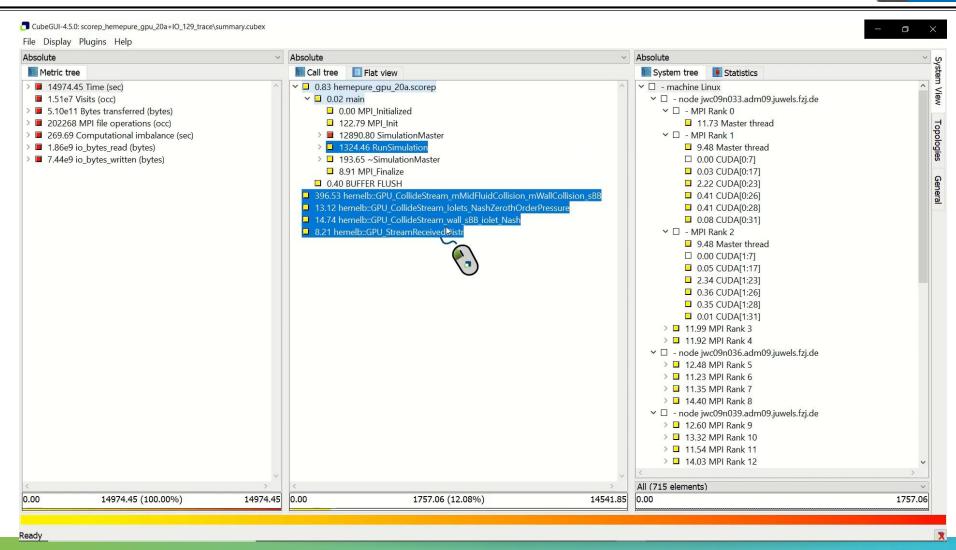


- Focus of analysis 2,000 time-step (each 100µs) simulation of CBM2019_Arteries_patched geometry
 - 1.78 GiB: 66,401,494 lattice sites, 1+38 iolets
- Executed on JUWELS-Volta (@JSC):
 - 2x 20-core Intel Xeon Platinum 8168 ('Skylake') CPUs + 4 Nvidia V100 'Volta' GPUs
 - 4* MPI processes/node (one per GPU), 32 (of 56) compute nodes: 129 MPI processes
- ⇒ Identification & quantification of impact of load balance and its variation

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

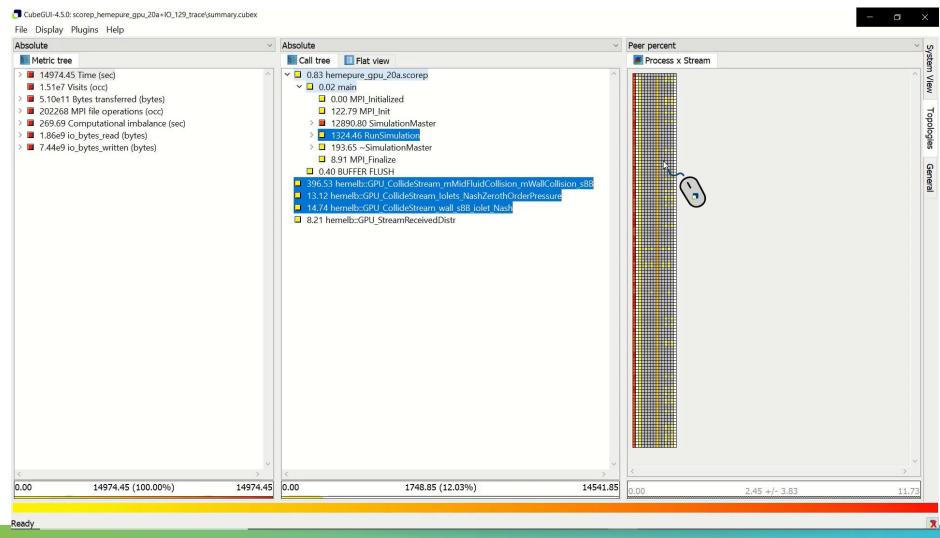


DOI 10.5281/zenodo.4081080



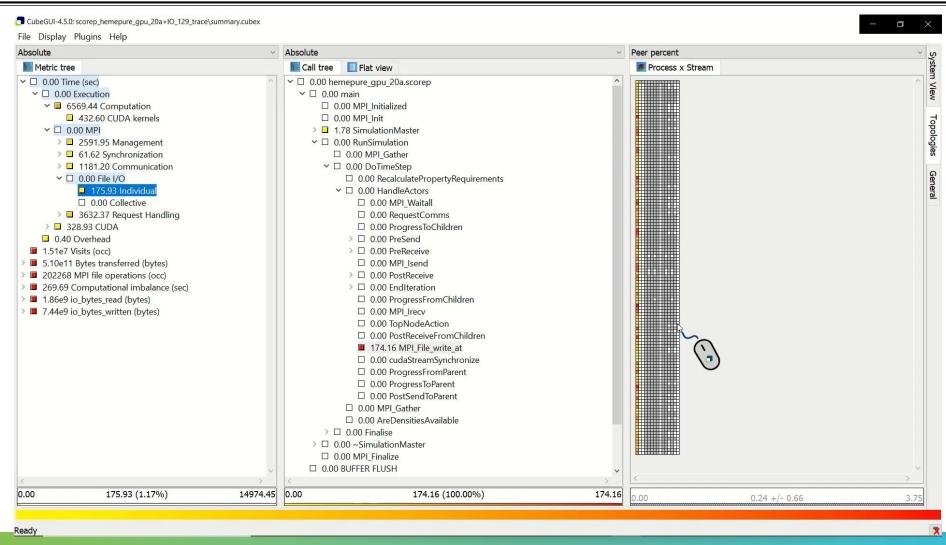
Topo: Time for asynch. CUDA kernels on separate CUDA streams





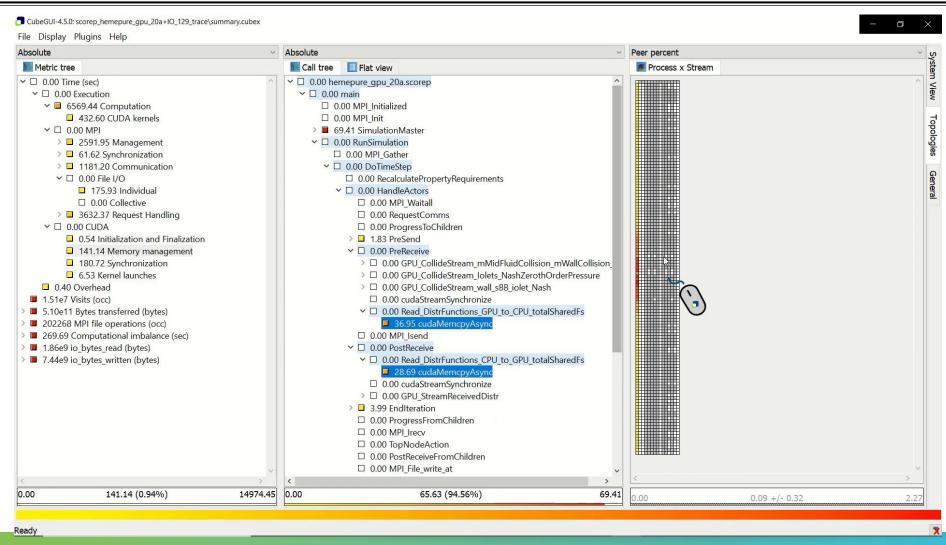
Topo: Time for MPI file writing on CPU varies per MPI process





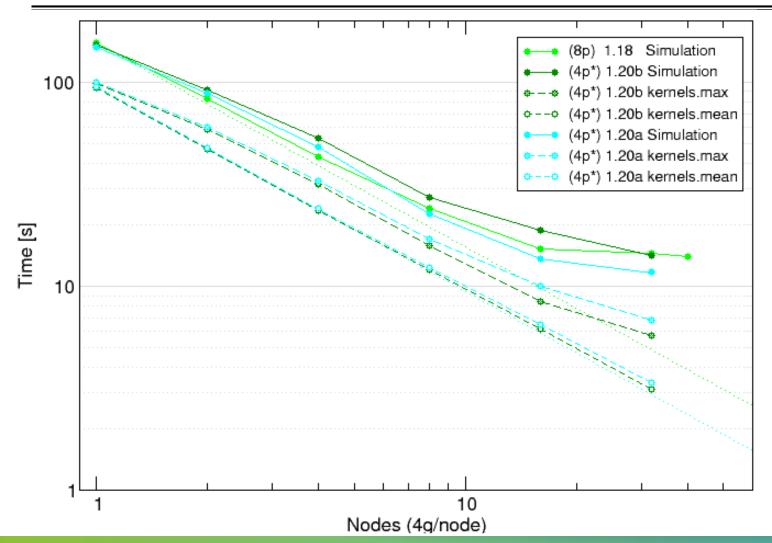
Topo: Time for CUDA asynchronous memory copies is imbalanced







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

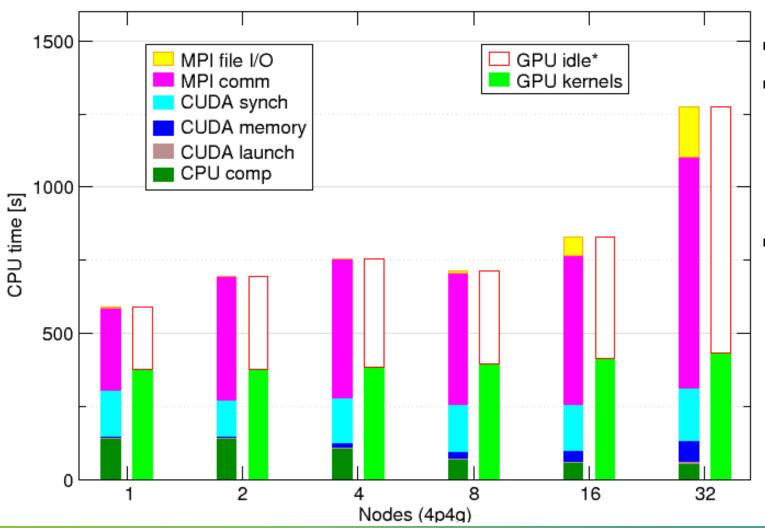
Simulation time [s]	1n 5p 147.87	2n 9p 88.38	4n 17p 48.13	8n 33p 22.66	16n 65p 13.68	32n 129p 11.67
Global scaling efficiency	0.64	0.53	0.49	0.52	0.43	0.25
– Parallel efficiency	0.64	0.53	0.50	0.54	0.47	0.29
– – Load balance efficiency (GPU)	0.95	0.78	0.73	0.73	0.65	0.50
– Communication efficiency (GPU)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

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]



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



HemeLB@JUWELS-Volta instrumentation & measurement



- Score-P/6.0 instrumentation
 - CMake: CUDACXX=scorep-nvcc (no instrumentation of CXX)
 - SCOREP_WRAPPER_COMPILER_FLAGS="--relocatable-device=true -c"
 - SCOREP_WRAPPER_INSTRUMENTER_FLAGS=
 "--cuda --mpp=mpi --thread=none --instrument-filter=hemelb.filt"
 - hemelb.filt specifying patterns to exclude most source modules and include only routines of particular interest
- Scalasca/2.5 runtime measurement configuration
 - SCAN_TRACE_ANALYZER=none # CUDA streams not supported
 - SCOREP_CUDA_ENABLE=runtime,memcpy,kernel,sync,flushatexit
 - SCOREP_CUDA_BUFFER=10MB
 - SCOREP_MPI_ENABLE_GROUPS=coll,env,io,p2p,rma,topo,xnonblock # no `cg' group
 - SCOREP_TIMER=gettimeofday # use globally synchronized clock
 - SCOREP_TOTAL_MEMORY=64MB # as determined from Score-P memory usage stats