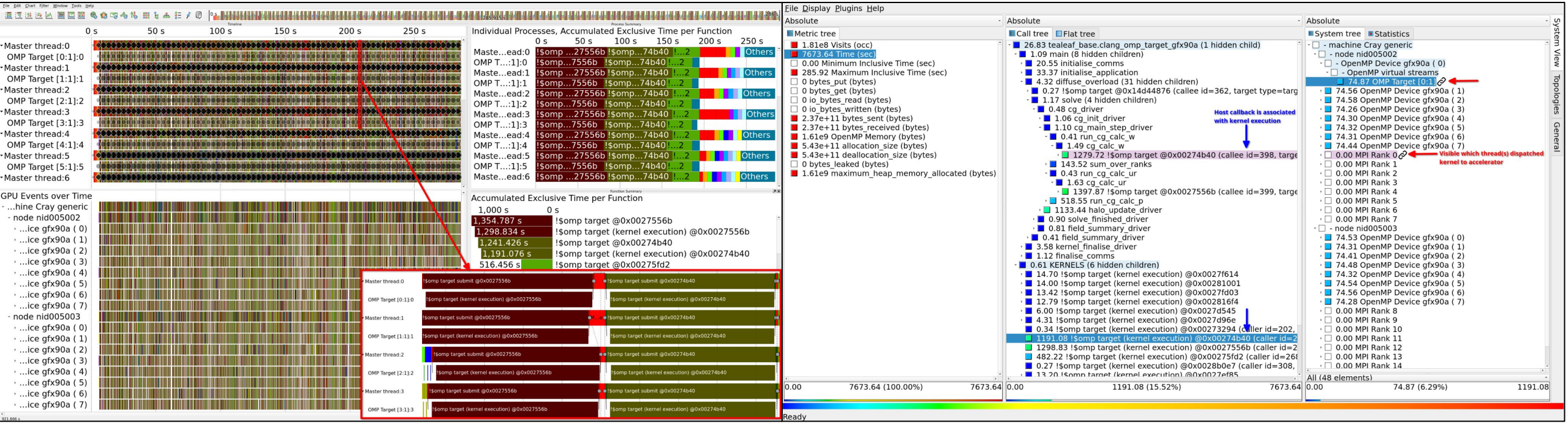


Talk @ TU Darmstadt

Overview of the Score-P & Scalasca measurement and analysis infrastructure

2026-05-27 I Jan André Reuter | j.reuter@fz-juelich.de

Score-P
Scalable performance measurement infrastructure for parallel codes

JÜLICH
Forschungszentrum

Motivation

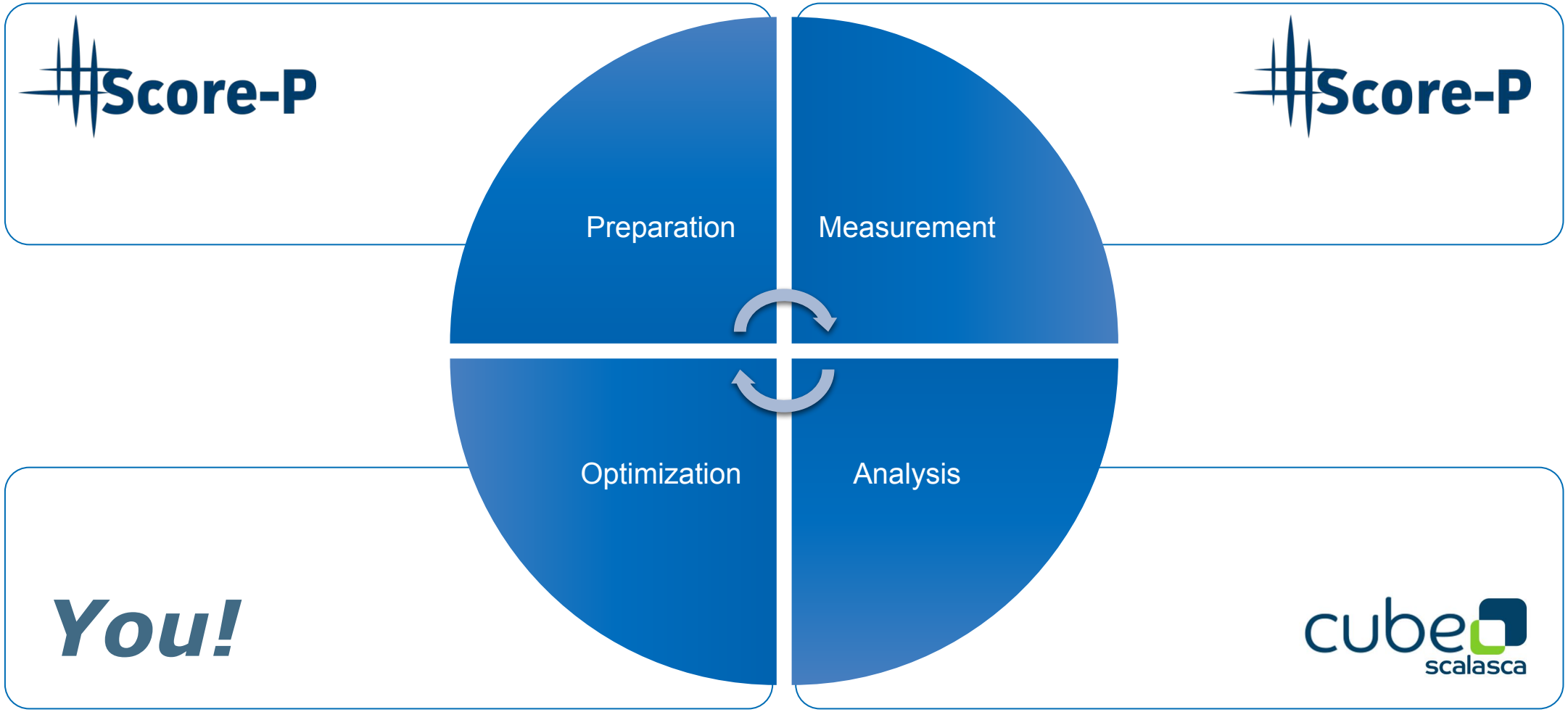
- Writing parallel code is hard
- Writing fast & efficient parallel code is even harder
- “Parallel” (multi-core / multi-node) performance factors
  - Partitioning / decomposition
  - Communication
  - Multithreading
  - Core binding
  - Synchronization
  - I/O
  - …

Tuning basics

- Carefully set various tuning parameters
  - The right (parallel) algorithms and libraries
  - Compiler flags and directives
  - Correct machine usage (mappings & bindings)

- Measurement is better than guessing
  - Determine performance bottlenecks
  - Test out alternatives
  - Validate tuning decisions and optimizations

Performance engineering workflow

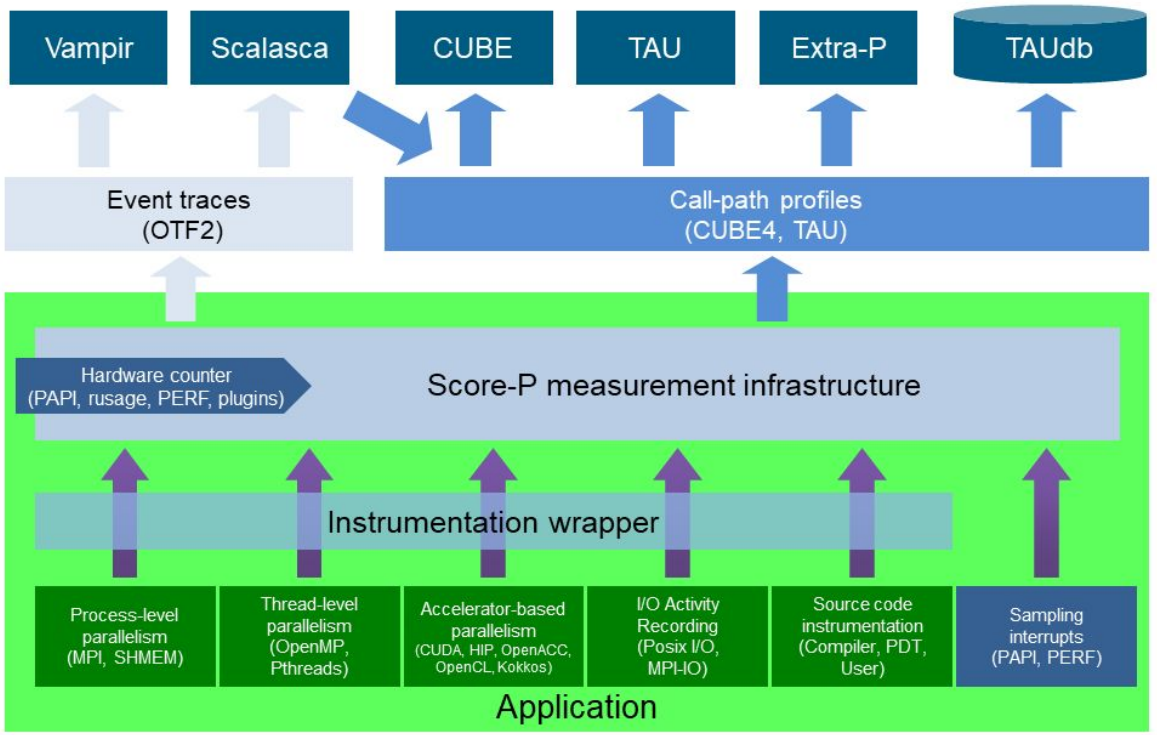


Score-P

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

- Score-P is a highly scalable performance measurement tool
- Support for multi-process, thread-parallel and accelerator-based paradigms
- Support for additional metrics (I/O, HW counters, …)
- Flexible measurement:
  - Profile generation
  - Event trace recording
- Support for C, C++, Fortran and Python
- Latest release: v9.4 (December 2025)



JÜLICH Forschungszentrum

Profiling & tracing definition

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

Profile / Runtime summarization

Summarization of events over execution interval

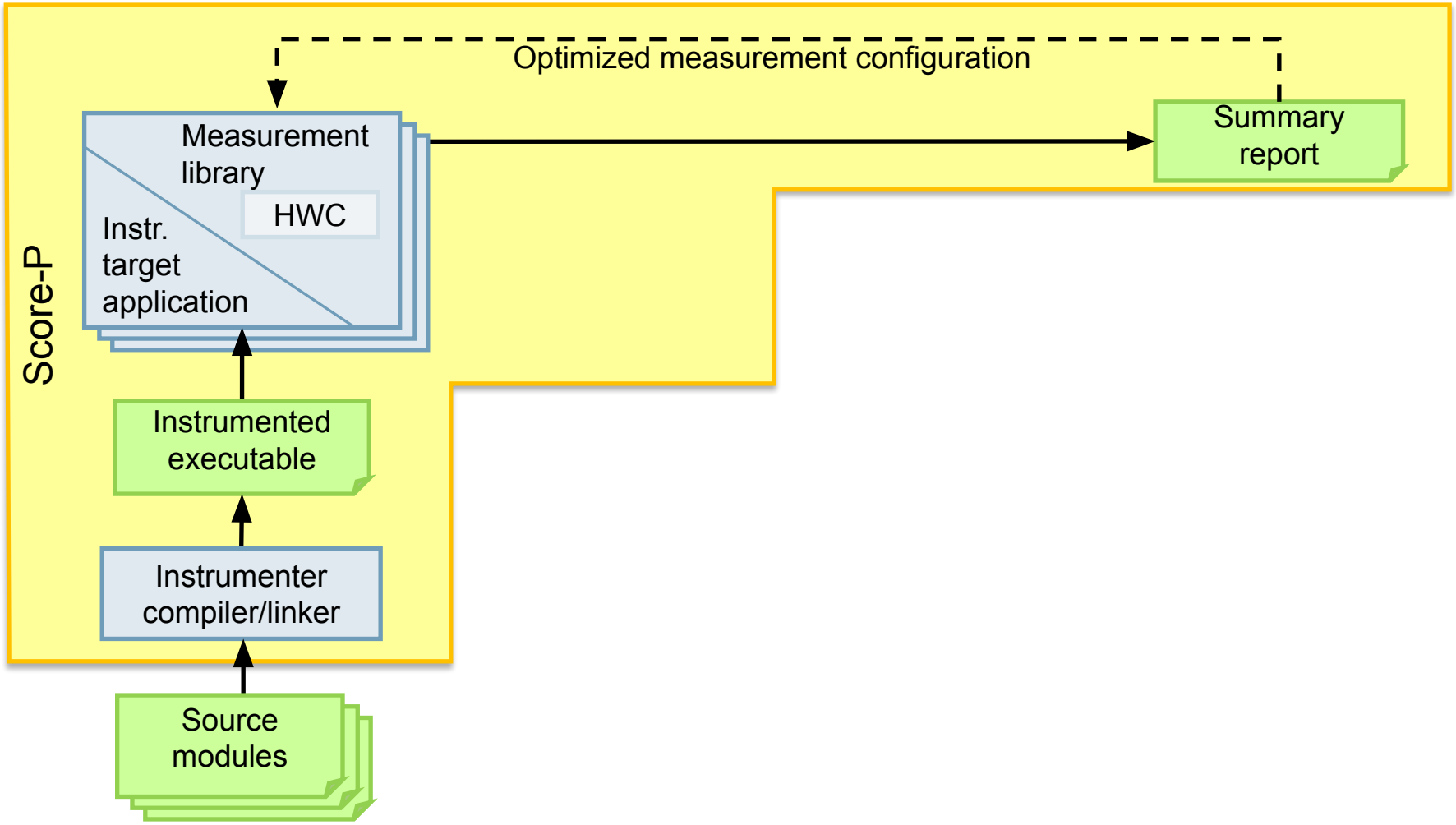
- Recording of aggregated information
  - Total, maximum, minimum, …
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
- Over program and system entities
  - Functions, call sites, basic blocks, loops, …
  - Processes, threads

Event trace

Chronologically ordered sequence of event records

- Recording detailed information about significant points during execution
  - Enter / leave of a region (function, loop, …)
  - Send / receive a message, …
- Save information in event record
  - Timestamp, location, event type
  - Plus event-specific information (e.g., communicator, sender / receiver, …)
- Abstract execution model on level of defined events

A typical instrumentation workflow



Instrumenting an application with Score-P

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

- Basic idea: Prepend compiler call with Score-P and options
  - Explicitly enable / disable paradigms
  - Most of the time detected automatically
- Build systems can be more challenging (e.g. CMake / Autotools)
  - Use wrapper scripts:

```
% SCOREP_WRAPPER=off \
  cmake -S <src-path> -B <build-path> \
        -DCMAKE_C_COMPILER=scorep-amdclang \
        -DCMAKE_CXX_COMPILER=scorep-amdclang++ \
        -DCMAKE_Fortran_COMPILER=scorep-amdflang
% SCOREP_WRAPPER_INSTRUMENTER_FLAGS="<scorep-options>" make
```

```
 LULESH_EXEC = lulesh2.0
-CXX = mpicxx
+CXX = scorep --user --thread=omp --mpp=mpi mpicxx
 CXXFLAGS = -g -O3 -fopenmp -I. -Wall
 LDFLAGS = -g -O3 -fopenmp

 .cc.o: lulesh.h
     $(CXX) -c $(CXXFLAGS) -o $@  $<

 all: $(LULESH_EXEC)

 $(LULESH_EXEC): $(OBJECTS2.0)
     @echo "Linking"
     $(CXX) $(OBJECTS2.0) $(LDFLAGS) -lm -o $@
```

Instrumenting an application with Score-P

What is happening behind the scenes?

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

```
$ scorep --verbose gcc minimal.c
# Obtain CFLAGS
[...]/scorep-config <adapters> --cflags
# Create object file with instrumentation
gcc -fno-ipa-icf -fplugin=[...]/scorep_instrument_function_gcc.so <Score-P CFLAGS> -c minimal.c -o minimal_[...].o
# Create adapter initialization file and build it
[...]/scorep-config <adapters> --adapter-init > .scorep_init.c
gcc -c .scorep_init.c -o .scorep_init.o
# Link everything together
gcc .scorep_init.o [...]/scorep_compiler_plugin_begin.o minimal_[...].o [...]/scorep_compiler_plugin_end.o
`[...]/scorep-config <adapters> --constructor` `[...]/scorep-config <adapters> --ldflags`  -Wl,-start-group
`[...]/scorep-config <adapters> --event-libs`  `[...]/scorep-config <adapters> --mgmt-libs` -Wl,-end-group
[...]
# Cleanup
rm minimal_[...].o .scorep_init.c .scorep_init.o
```

Running an instrumented application

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

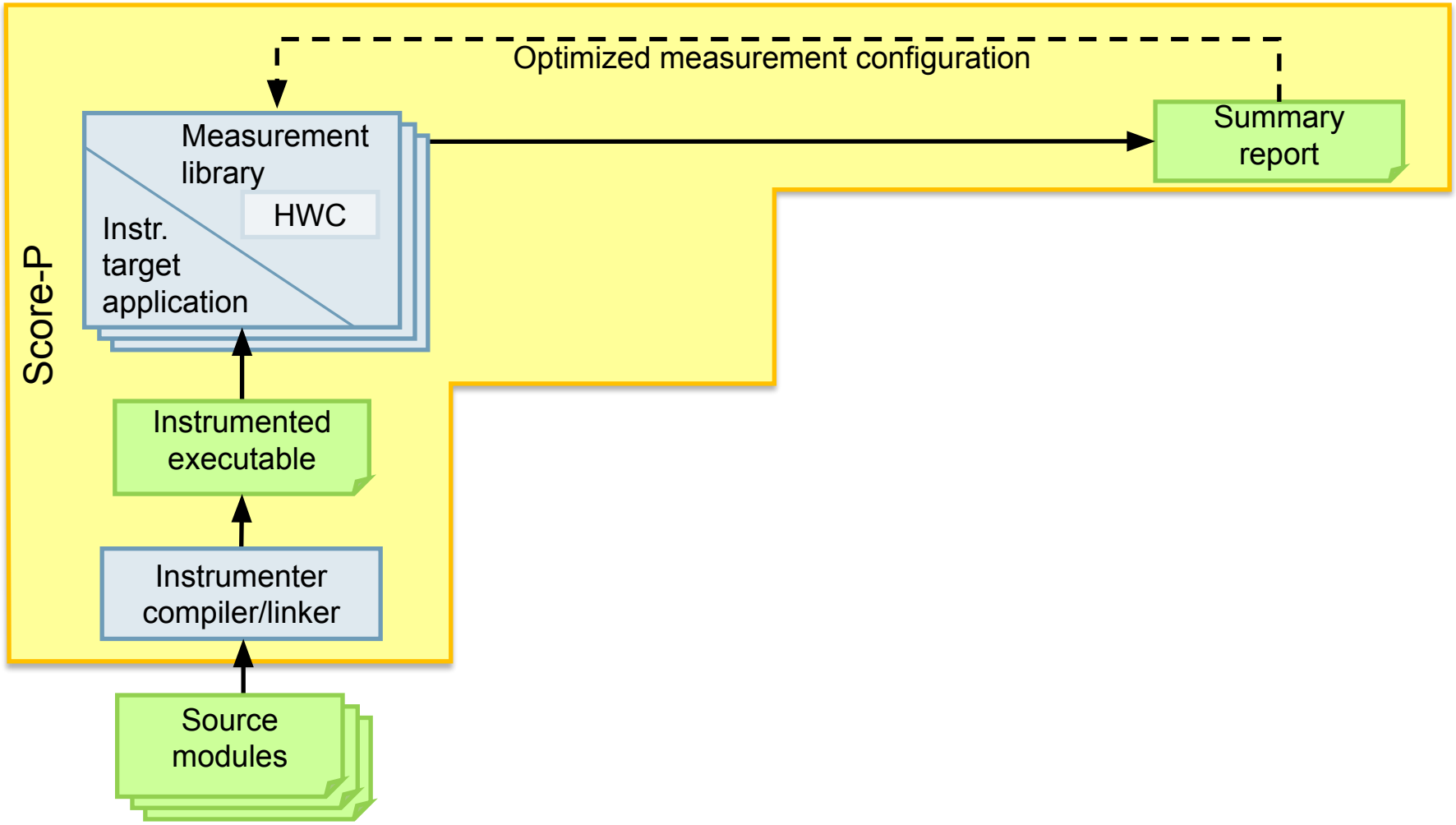
- Score-P measurements are configured via environment variables, e.g.

```
% scorep-info config-vars
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
[...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
[...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
[...]
SCOREP_HIP_ENABLE
  Description: HIP measurement features
         Type: Set
```

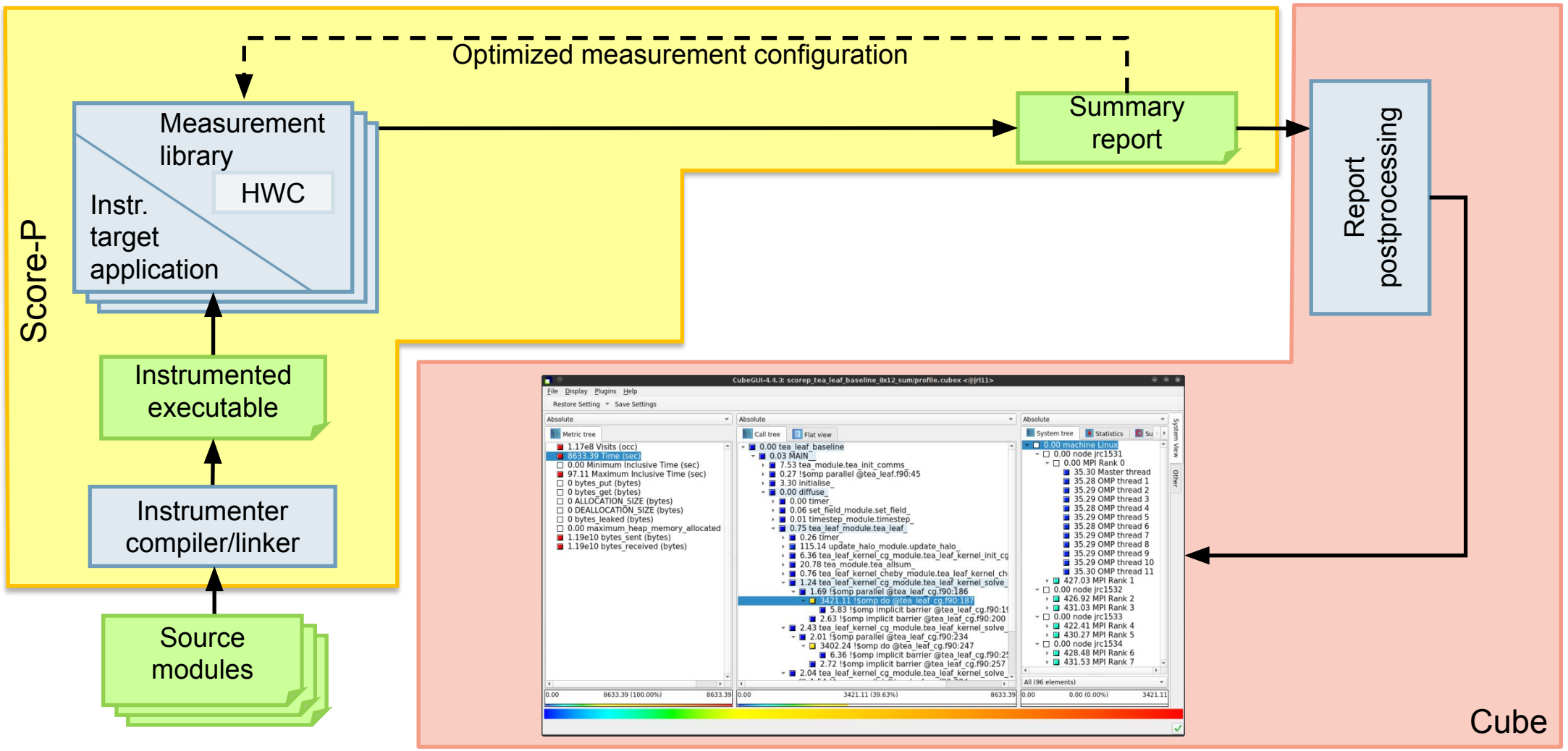
- Applications can be simply ran, creating an experiment directory on finished program execution

```
% SCOREP_[OPTION]=[VALUE] mpiexec -np 8 ./lulesh2.0
Running problem size 30^3 per domain until completion
Num processors: 8
Num threads: 2
Total number of elements: 216000
[...]
Run completed:
[...]
Elapsed time =   9.9 (s)
Grind time   =   17.130709 (per dom)
FOM          =   466.99759 (z/s)
% ls scorep-[YYYYMMDD]_[hhmm]_[suffix]
MANIFEST.md  profile.cubex  scorep.cfg
```

A typical instrumentation workflow



A typical instrumentation workflow

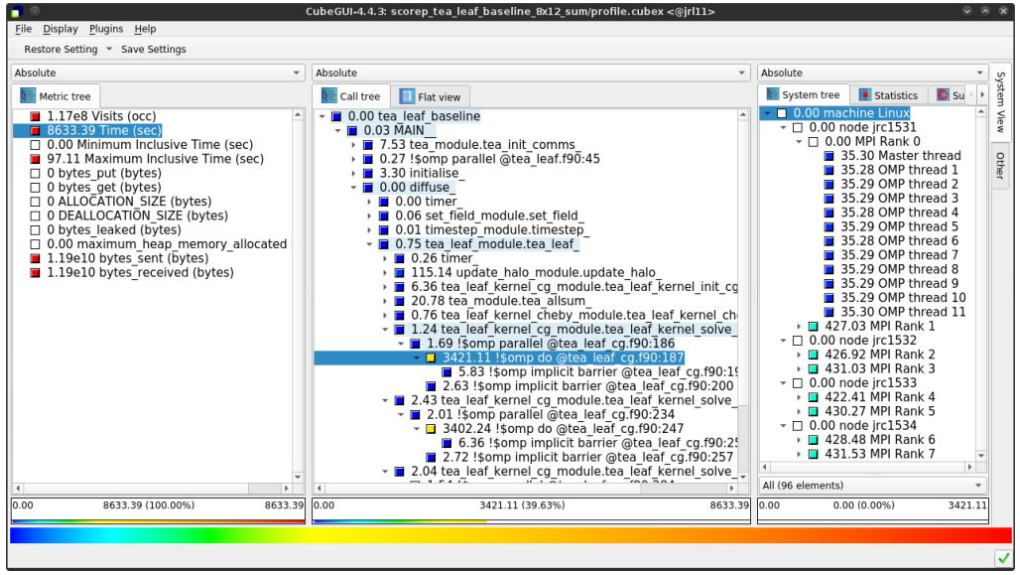


Cube

CubeLib DOI 10.5281/zenodo.1248078

CubeGUI DOI 10.5281/zenodo.1248087

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Originally developed as part of the Scalasca toolset



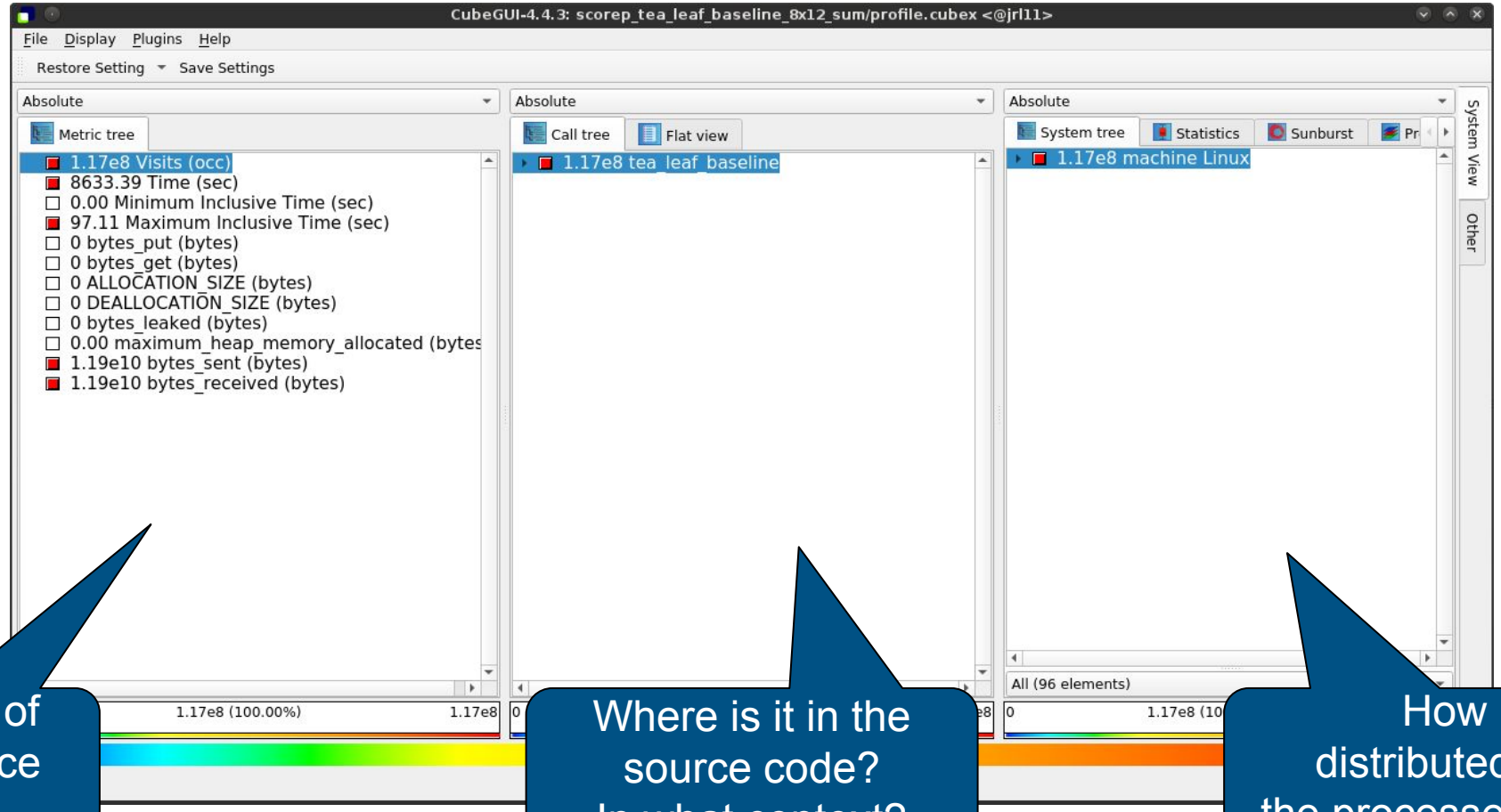
- Now available as a separate component
  - Can be installed independently of Score-P and Scalasca, e.g., on laptop or desktop
  - Latest release: Cube v4.9.1 (December 2025)

Note: source distribution tarballs for Linux, as well as binary packages provided for Windows & MacOS, from **www.scalasca.org** website in software/Cube-4x

JÜLICH Forschungszentrum

Cube

CubeLib DOI 10.5281/zenodo.1248078
CubeGUI DOI 10.5281/zenodo.1248087



Post-processing your measurement

CubeLib DOI 10.5281/zenodo.1248078
CubeGUI DOI 10.5281/zenodo.1248087

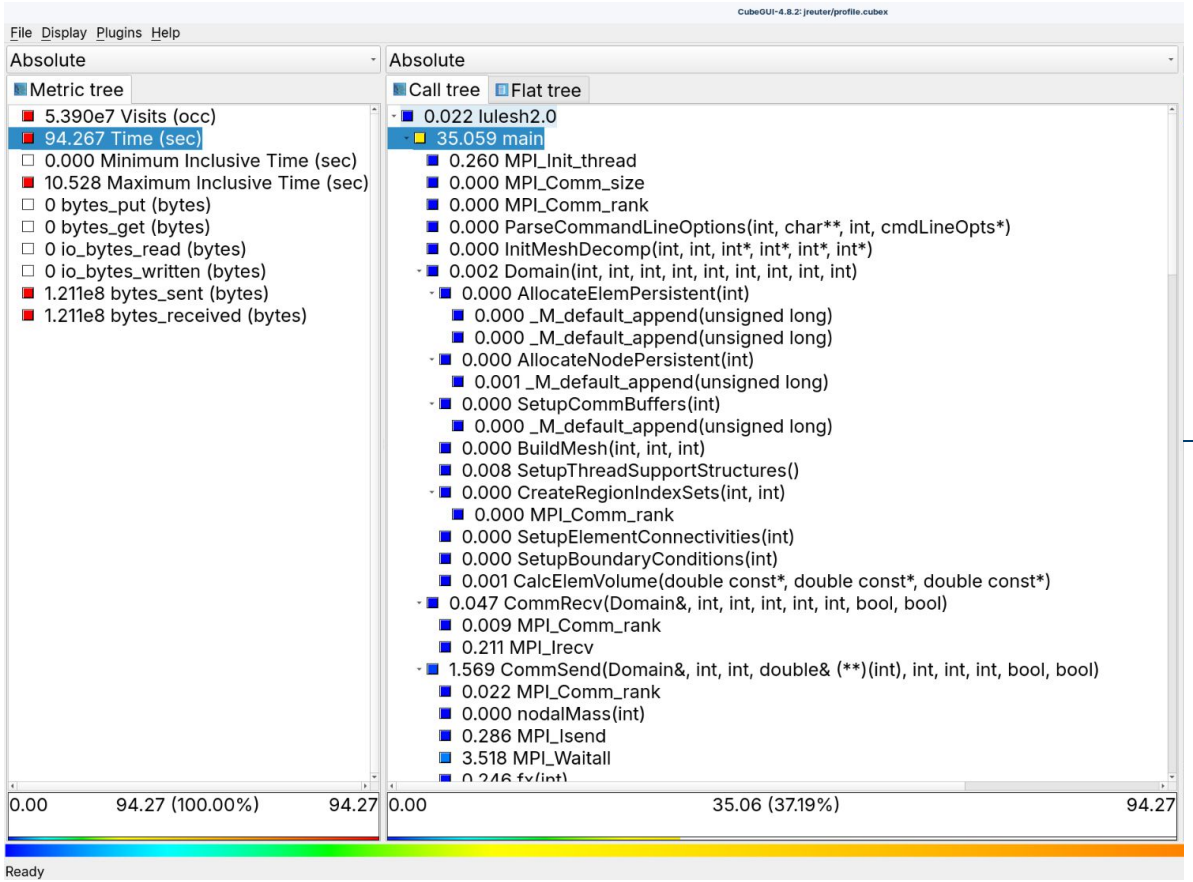
- Measurements do not contain many metrics
- Enrich measurement by post-processing events into categories, like Computation and Kernel Launches
- **`cube_remap2`** uses CubePL specification file embedded into measurements
- Metrics defined by Score-P and updated by developers

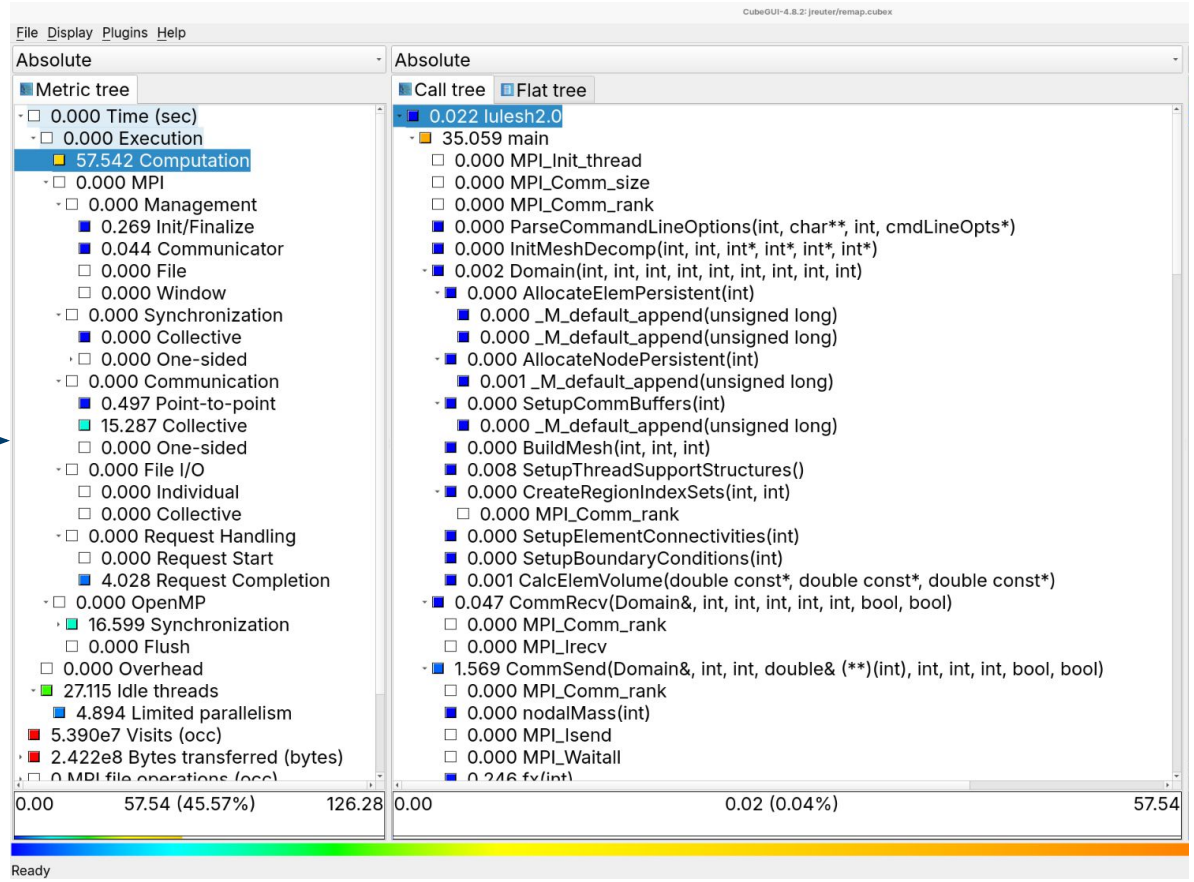
```
Usage: cube_remap2 -r <remap specification file> [-o output] [-d] [-s] [-h] <cube experiment>
  -r     Name of the remapping specification file. By omitting this option the specification file from the
        cube experiment is taken if present.
  -c     Create output file with the same structure as an input file. It overrides option "-r"
  -o     Name of the output file (default: remap)
  -d     Convert all prederived metrics into usual metrics, calculate and store their values as a data.
  -s     Add hardcoded Scalasca metrics "Idle threads" and "Limited parallelizm"
  -h     Help; Output a brief help message.

Report bugs to <scalasca@fz-juelich.de>

jreuter@ZAM054 ~ cube_remap2 -s -d profile.cubex
++++++++++++ Remapping operation begins +++++++++++++++++++++++++
Reading profile.cubex ...  done.
Found remapping specification file inside of cube. Use it.
Create cube according the remapping specification...
 Add scalasca specific metrics ...
 Add metrics "idle threads" and "limited parallelism"...done.
 Add metrics "load imbalance"...
      Determine numbers of void processes and threads to ignore...done.
      Calculate average exclusive execution time...done.
      Calculate difference to average value...done.
 Add metrics "imbalance due to singularities and imperfect parallelization"...done.
Copy values of derived metric into target cube...
 Metric min_time with type MINDOUBLE cannot be declared as an inclusive metric
 Metric max_time with type MAXDOUBLE cannot be declared as an inclusive metric
++++++++++++ Remapping operation ends successfully ++++++++++++++++
Writing remap ... done.
```

Looking at post-processed measurements

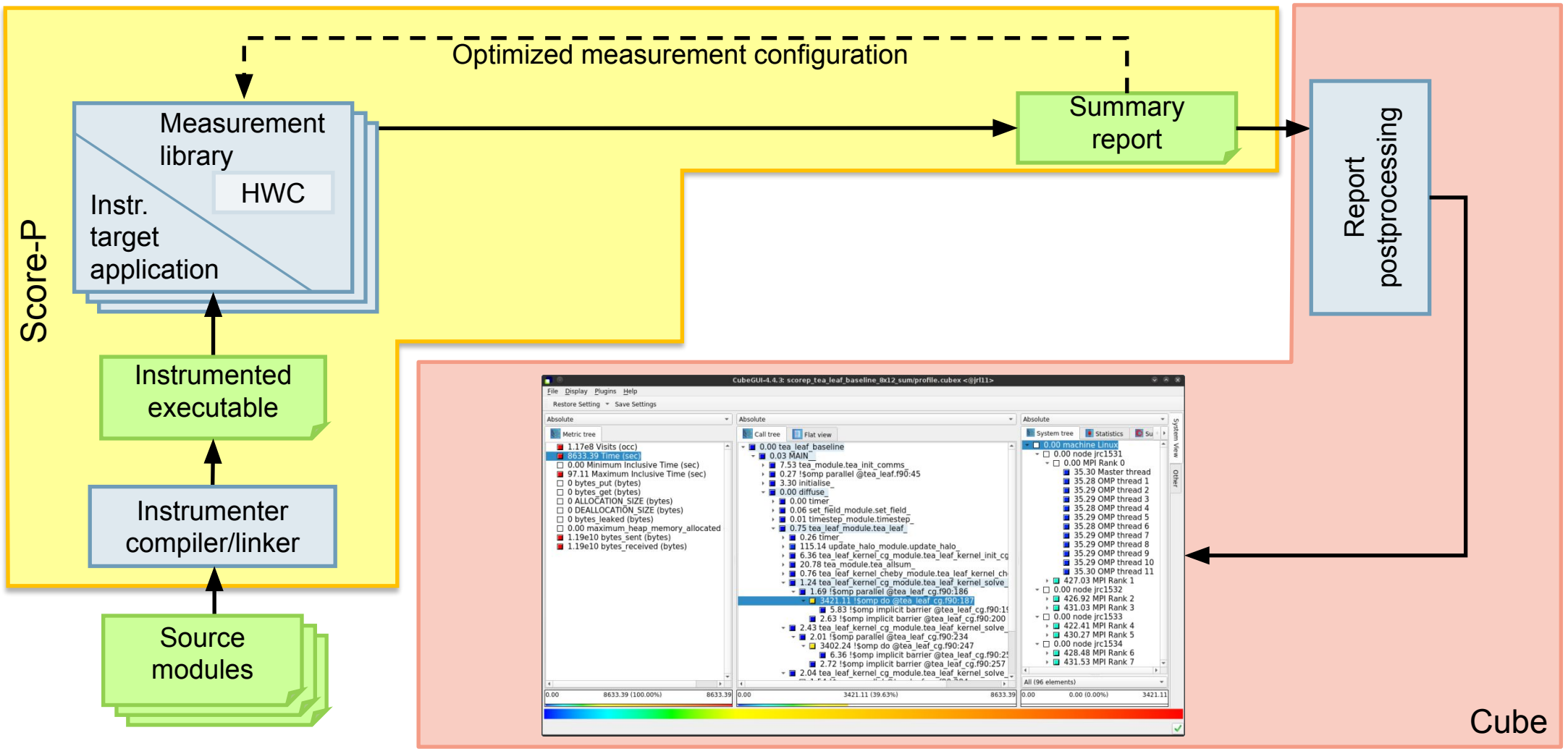
CubeLib DOI 10.5281/zenodo.1248078

CubeGUI DOI 10.5281/zenodo.1248087

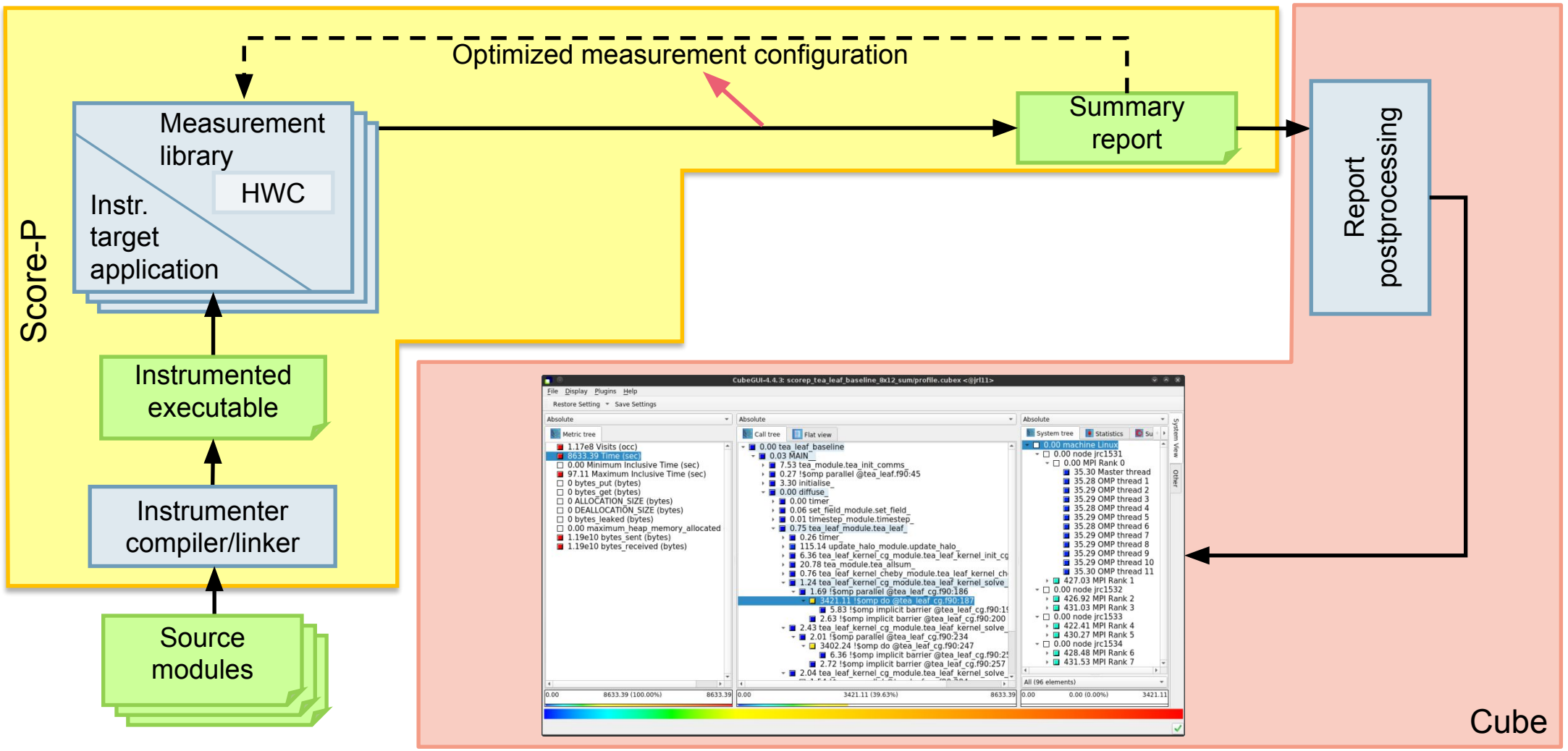




A typical instrumentation workflow



A typical instrumentation workflow

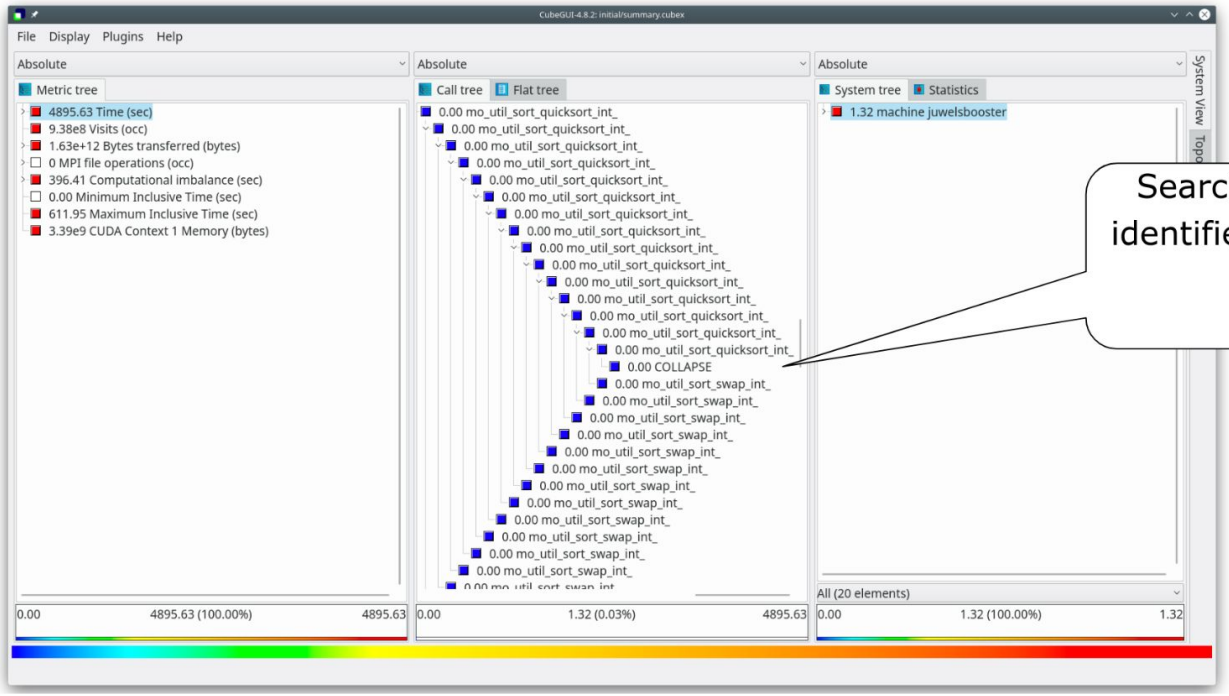


Optimizing the first measurement for analysis

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

- Measurements might have a high overhead or are too unwieldy to analyze



Search for "COLLAPSE" callpath identified a small number of highly recursive functions
⇨ Filter!

- **Solution:** Filter your measurements
  - User adapter to select relevant regions
  - Filter file to select recorded functions
  - Adapter settings to enable / disable features

JÜLICH Forschungszentrum

Using filters to optimize measurements

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

- **Idea:** Filter functions with low impact but high overhead
- Automatic filtering via `scorep-score -g`
- Filter can be applied in two ways:
  - Runtime via `SCOREP_FILTERING_FILE=<file>`
  - Compile-time via `scorep --instrument-filter=<file>`

```
jreuter@ZAM054 ~ scorep-score -r profile.cubex

Estimated aggregate size of event trace:                   1600MB
Estimated requirements for largest trace buffer (max_buf): 231MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       235MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=235MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type  max_buf[B]     visits time[s] time[%] time/visit[us]  region
         ALL 242,073,659 53,895,971   94.27   100.0          1.75  ALL
         USR 143,044,642 44,013,717    5.15     5.5          0.12  USR
         OMP  96,709,250  9,493,250   29.11    30.9          3.07  OMP
         MPI   1,856,105    246,348   20.13    21.3         81.70  MPI
         COM     463,606    142,648   39.85    42.3        279.39  COM
     SCOREP          56          8    0.02     0.0       2700.37  SCOREP

         OMP  46,923,450  3,218,850    3.66     3.9          1.14  !$omp parallel @lulesh.cc:2748
         USR  29,900,000  9,200,000    1.26     1.3          0.14  CalcElemShapeFunctionDerivatives(double const*, double cons
t*, double const*, double (*) [8], double*)
         USR  14,976,000  4,608,000    0.61     0.6          0.13  CalcElemVolume(double const*, double const*, double const*)
         OMP  14,023,100  2,147,050   13.79    14.6          6.42  !$omp implicit barrier @lulesh.cc:2748
         USR  11,870,300  3,652,400    0.43     0.5          0.12  Domain::fx(int)
         USR  11,870,300  3,652,400    0.42     0.4          0.11  Domain::fy(int)
         USR  11,870,300  3,652,400    0.40     0.4          0.11  Domain::fz(int)
         USR   8,970,000  2,760,000    0.28     0.3          0.10  Domain::delv_xi(int)
         USR   8,970,000  2,760,000    0.27     0.3          0.10  Domain::delv_eta(int)
         USR   8,970,000  2,760,000    0.26     0.3          0.09  Domain::delv_zeta(int)
         USR   5,935,150  1,826,200    0.22     0.2          0.12  Domain::x(int)
         USR   5,935,150  1,826,200    0.21     0.2          0.12  Domain::y(int)
```

```
lulesh.h:266:   Real_t& x(Index_t idx){ return m_x[idx] ; }
```

Verifying filters

Score-P
Scalable performance measurement infrastructure for parallel codes

DOI 10.5281/zenodo.1240731

- Example filter:

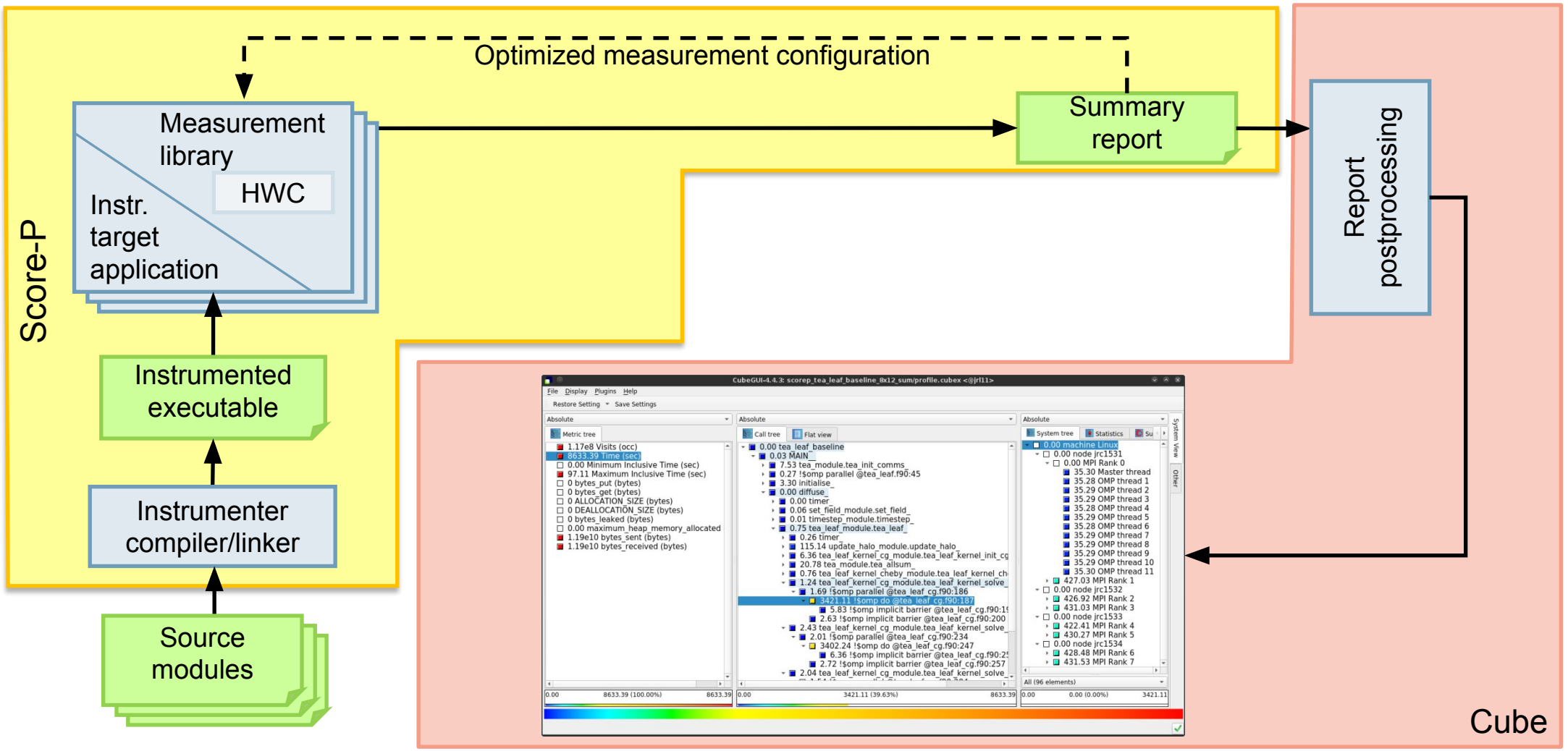
```
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    MANGLED _Z14CalcElemVolumePKdS0_S0_
    MANGLED _ZN6Domain2fxEi
    [...]
SCOREP_REGION_NAMES_END
```

- Result can be examined with

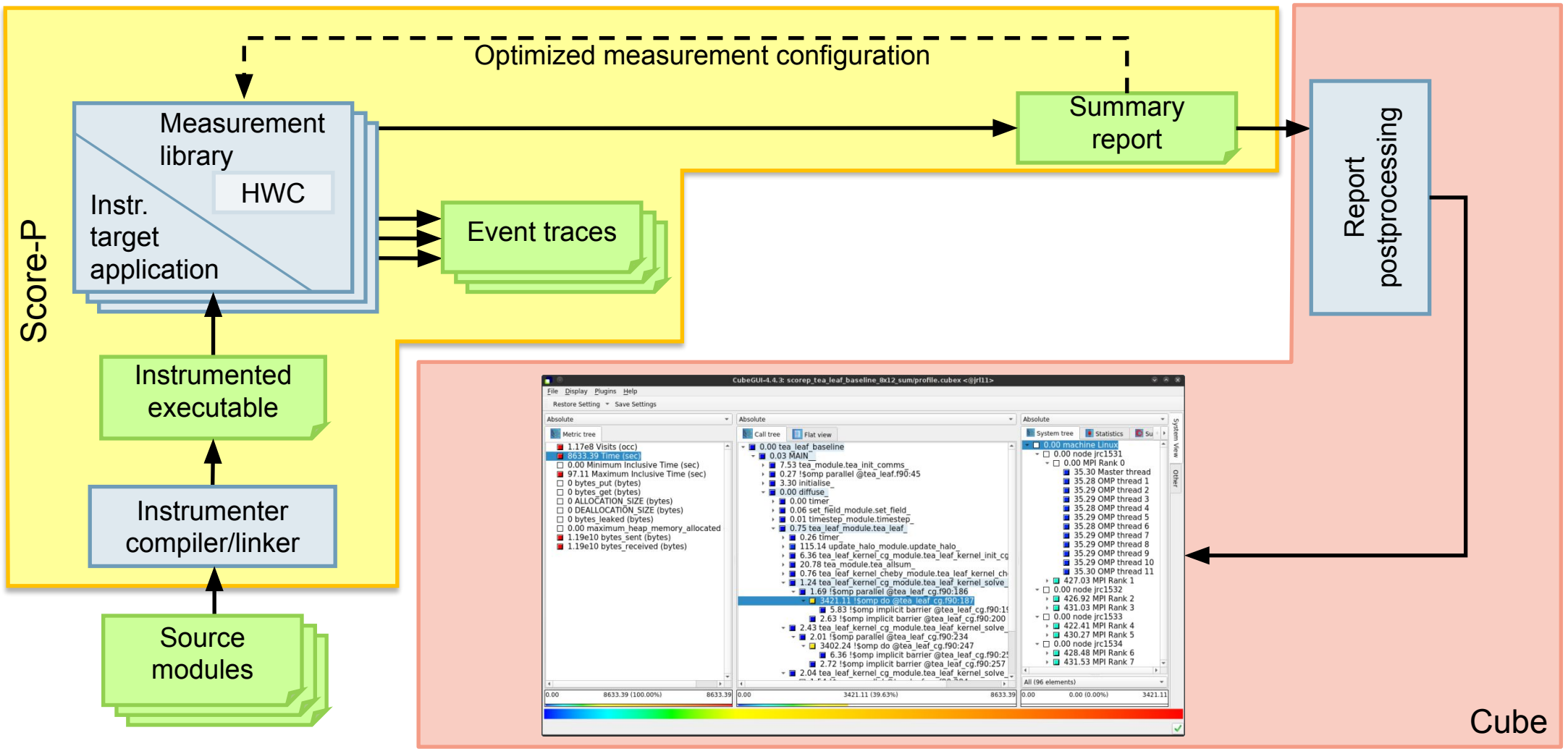
  `scorep-score -f <filter-file>`

```
jreuter@ZAM054 ~/Projects/Benchmarks/LULESH/scorep-20241128_1420_706694849837105 master ± scorep-score -r -f initial_s
corep.filter profile.cubex

Estimated aggregate size of event trace:                   509MB
Estimated requirements for largest trace buffer (max_buf): 95MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       99MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=99MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type  max_buf[B]     visits time[s] time[%] time/visit[us]  region
 -       ALL 242,073,659 53,895,971   94.27   100.0           1.75  ALL
 -       USR 143,044,642 44,013,717    5.15     5.5           0.12  USR
 -       OMP  96,709,250  9,493,250   29.11    30.9           3.07  OMP
 -       MPI   1,856,105    246,348   20.13    21.3          81.70  MPI
 -       COM     463,606    142,648   39.85    42.3         279.39  COM
 -    SCOREP          56          8    0.02     0.0        2700.37  SCOREP

 *       ALL  99,065,859  9,893,571   89.13    94.6           9.01  ALL-FLT
 +       FLT 143,007,800 44,002,400    5.13     5.4           0.12  FLT
 -       OMP  96,709,250  9,493,250   29.11    30.9           3.07  OMP-FLT
 -       MPI   1,856,105    246,348   20.13    21.3          81.70  MPI-FLT
 *       COM     463,606    142,648   39.85    42.3         279.39  COM-FLT
 *       USR      36,842     11,317    0.02     0.0           1.51  USR-FLT
 -    SCOREP          56          8    0.02     0.0        2700.37  SCOREP-FLT

 -       OMP  46,923,450  3,218,850    3.66     3.9           1.14  !$omp parallel @lulesh.cc:2748
 +       USR  29,900,000  9,200,000    1.26     1.3           0.14  CalcElemShapeFunctionDerivatives(double const*, double cons
t*, double const*, double (*) [8], double*)
 +       USR  14,976,000  4,608,000    0.61     0.6           0.13  CalcElemVolume(double const*, double const*, double const*)
 -       OMP  14,023,100  2,147,050   13.79    14.6           6.42  !$omp implicit barrier @lulesh.cc:2748
 +       USR  11.870.300  3.652.400    0.43     0.5           0.12  Domain::fx(int)
```

A typical instrumentation workflow



A typical instrumentation workflow



Generating event traces

- Event trace collection not enabled by default
- To enable, set **SCOREP_ENABLE_TRACING=true**

```
% scorep-info config-vars
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
[...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
[...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
[...]
SCOREP_HIP_ENABLE
  Description: HIP measurement features
         Type: Set
```

- Application run will create trace files in addition to profile

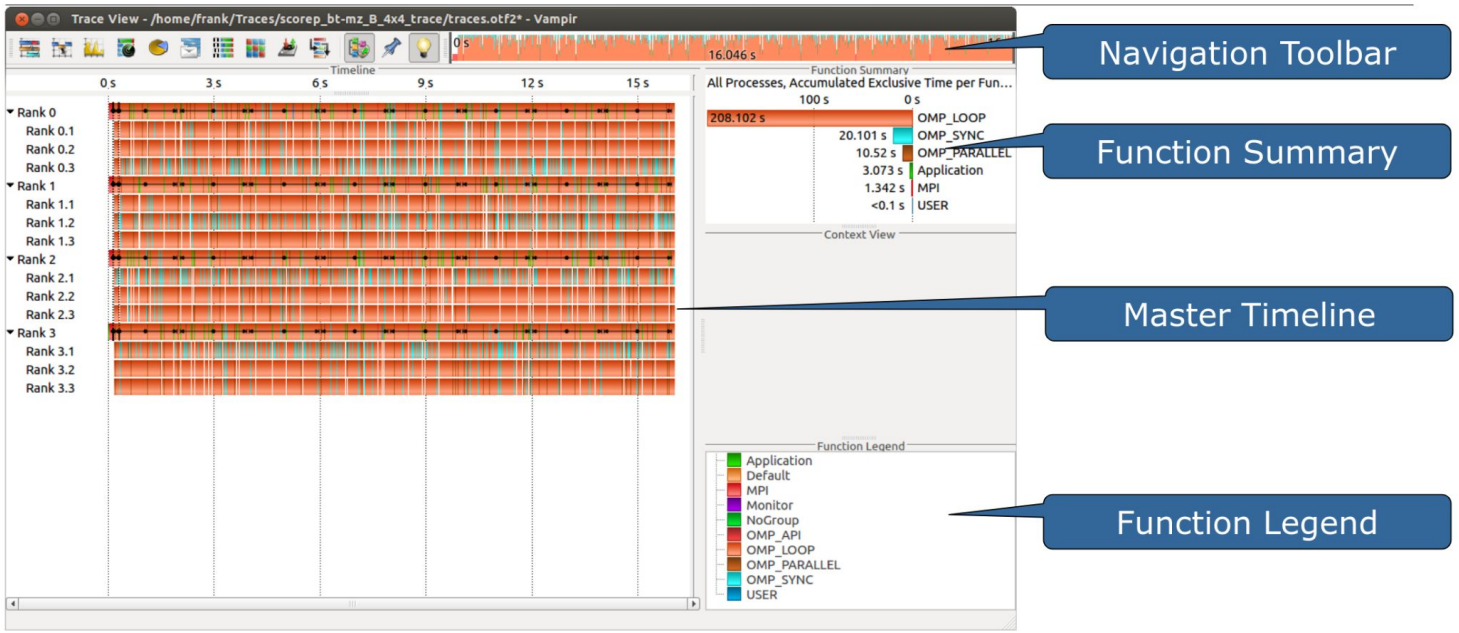
```
% SCOREP_ENABLE_TRACING=true mpiexec -np 8 ./lulesh2.0
Running problem size 30^3 per domain until completion
Num processors: 8
Num threads: 2
Total number of elements: 216000
[...]
Run completed:
[...]
Elapsed time =    9.9 (s)
Grind time   =    17.130709 (per dom)
FOM          =    466.99759 (z/s)
% ls scorep-[YYYYMMDD]_[hhmm]_[suffix]
MANIFEST.md  profile.cubex  scorep.cfg
traces       traces.def     traces.otf2
```

Vampir

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically at any level of detail
- Provide statistics and performance metrics

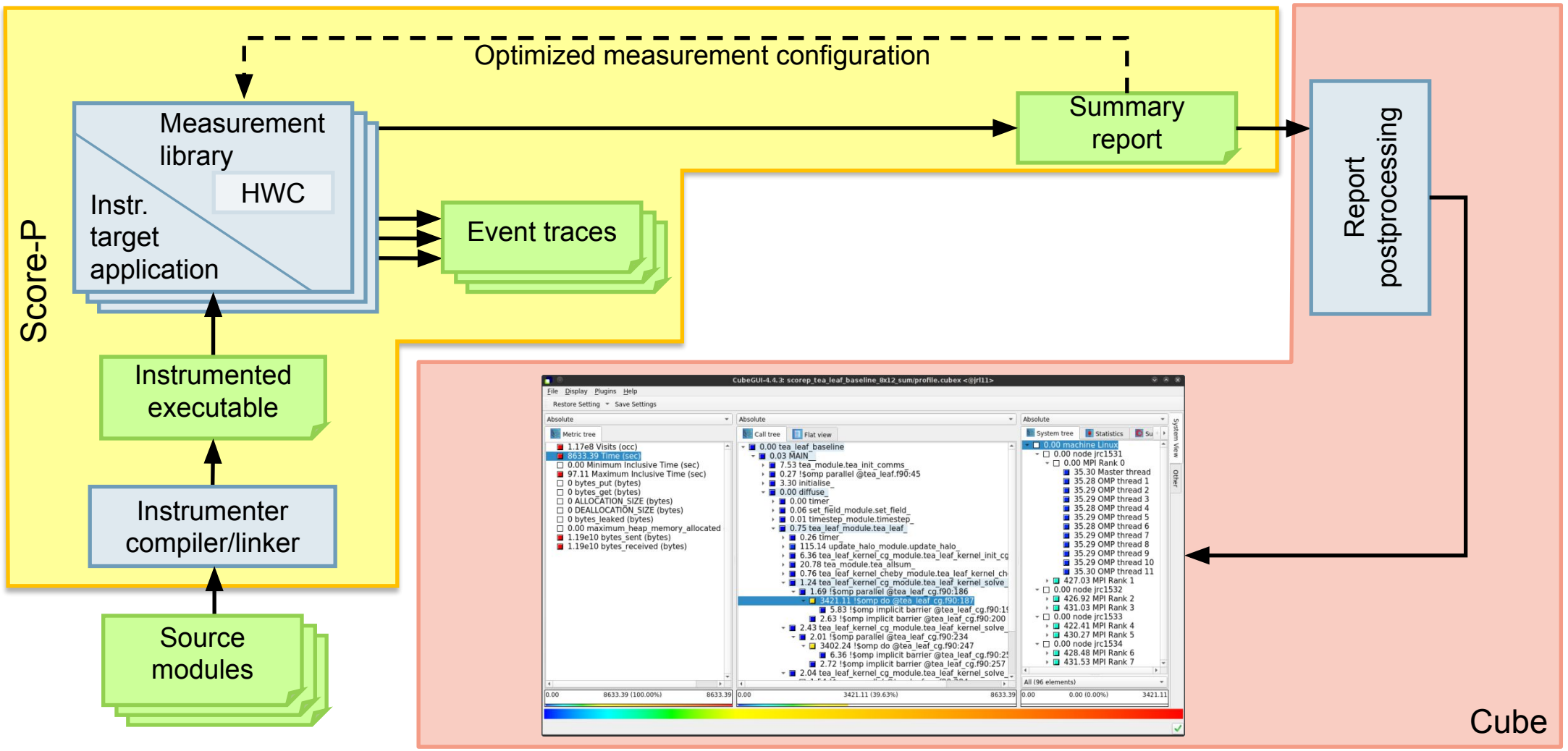
- **Timeline charts**
  - Show application activities and communication along a time axis

- **Summary charts**
  - Provide quantitative results for the currently selected time interval

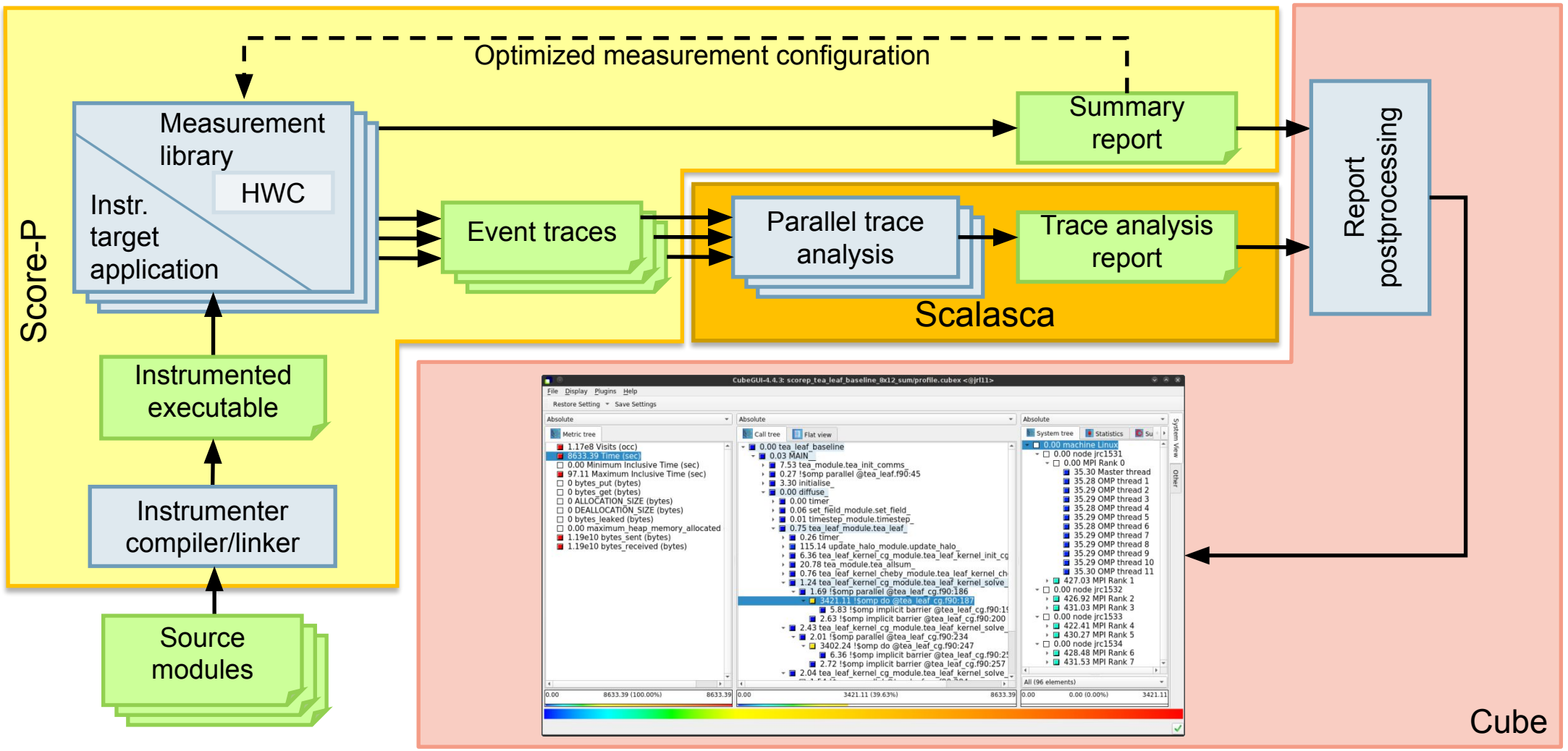


Inspecting your traces

A typical instrumentation workflow



A typical instrumentation workflow



JÜLICH Forschungszentrum

Scalasca Trace Tools

DOI 10.5281/zenodo.7440854

- **Scalable trace-based** performance analysis toolset for popular parallel programming paradigms
  - Main idea: Automatic trace analysis
  - Based on **Score-P** instrumentation and measurement system & **Cube** analysis report browser
    - Reads event traces in OTF2 format, generated by Score-P
    - Writes analysis reports in CUBE format, enriched with higher-level metrics
  - Current focus: MPI, OpenMP (host-side), and (to a limited extend) POSIX threads

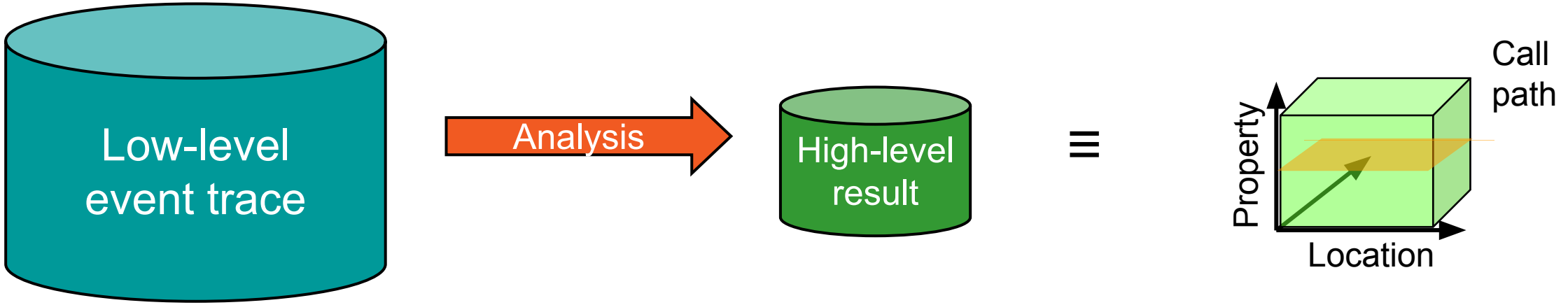
- Specifically targeting large-scale parallel applications
  - Demonstrated scalability up to 1.8 million parallel threads
  - Of course also works at small/medium scale

Automatic trace analysis

scalasca

DOI 10.5281/zenodo.7440854

- Automatic search for patterns of inefficient behavior (e.g., wait states)
- Classification of behavior & quantification of significance
- Identification of delays as root causes of inefficiencies



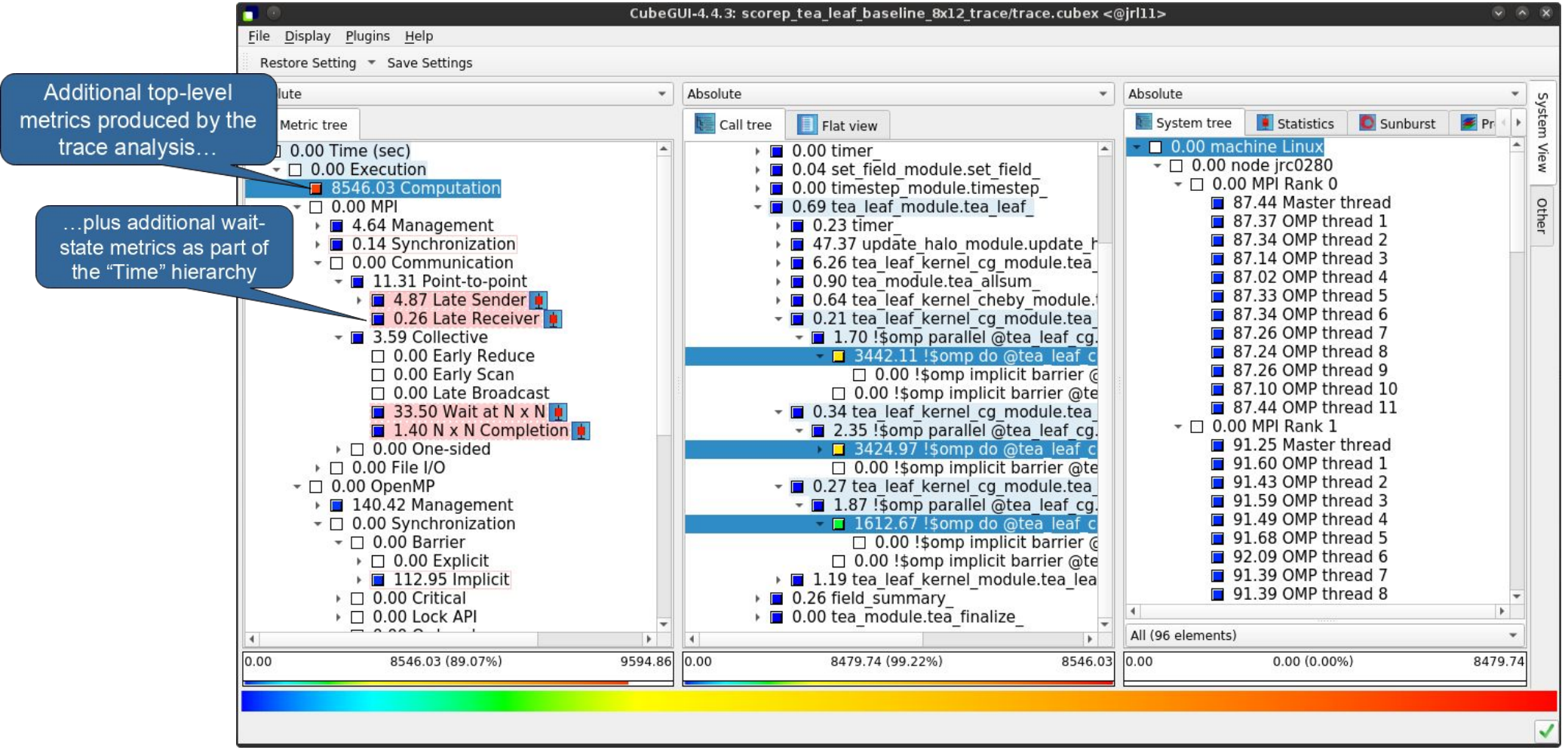
- Guaranteed to cover the entire event trace
- Quicker than a manual/visual trace analysis
  - Not necessarily a replacement, but rather a complement to assist in focusing a manual/visual analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

How to use Scalasca?

DOI 10.5281/zenodo.7440854

- Run your measurement with it:
  `SCOREP_TOTAL_MEMORY=<memory> scan -s -t mpiexec -np 8 ./lulesh2.0`
- Post-measurement analysis:
  `mpiexec -np 8 scout.hyb traces.otf2`
  - Different variants depending on your application:
    - MPI: `scout.mpi`
    - OpenMP: `scout.omp`
    - Hybrid: `scout.hyb`
    - Requires same number of threads / ranks as your application

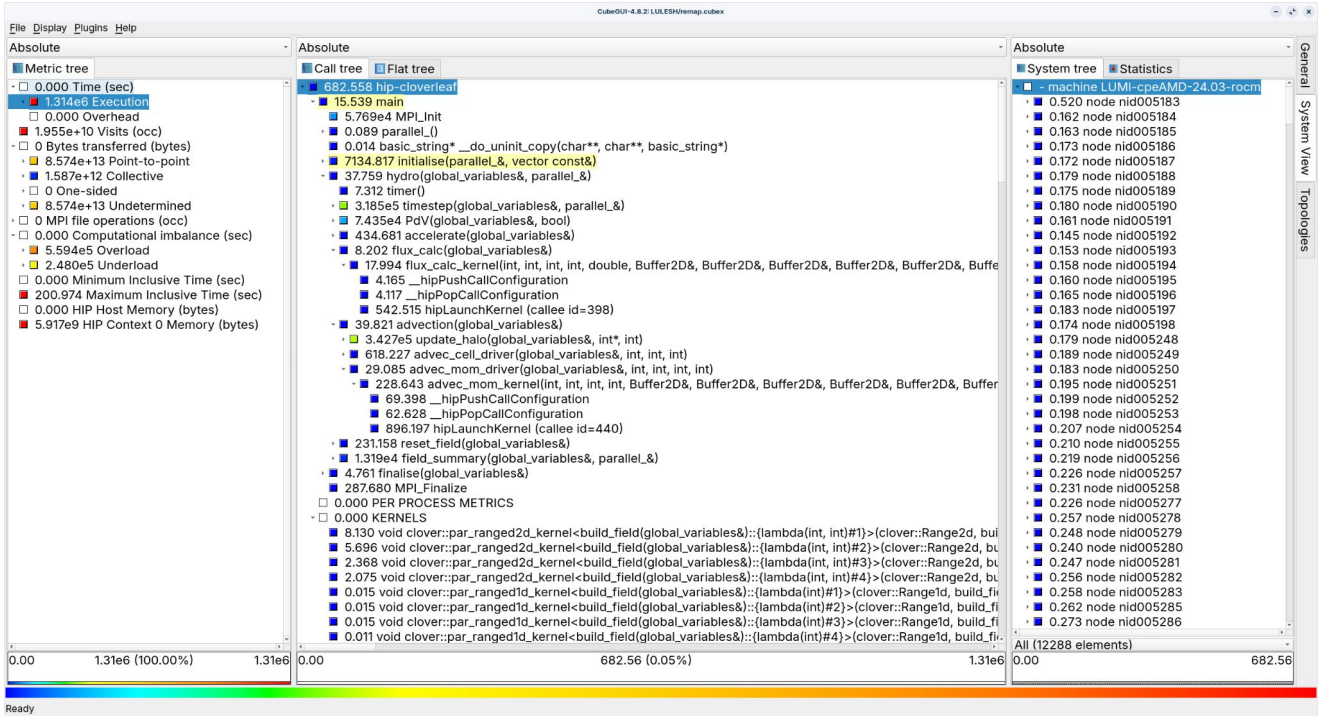
Looking at enhanced profiles



Measuring GPU applications

- HIP & ROCm adapter for AMD GPUs
- CUDA adapter for NVIDIA GPUs
- API-specific adapters:

  OpenMP, OpenACC, Kokkos, OpenCL, …




```
$ scorep-hipcc
    --offload-arch=gfx90a
    my-hip-code.hip
$ ./a.out
$ # Remap for additional metrics
$ cube_remap2 -s -d
  scorep-<suffix>/profile.cubex
$ cube remap.cubex
```

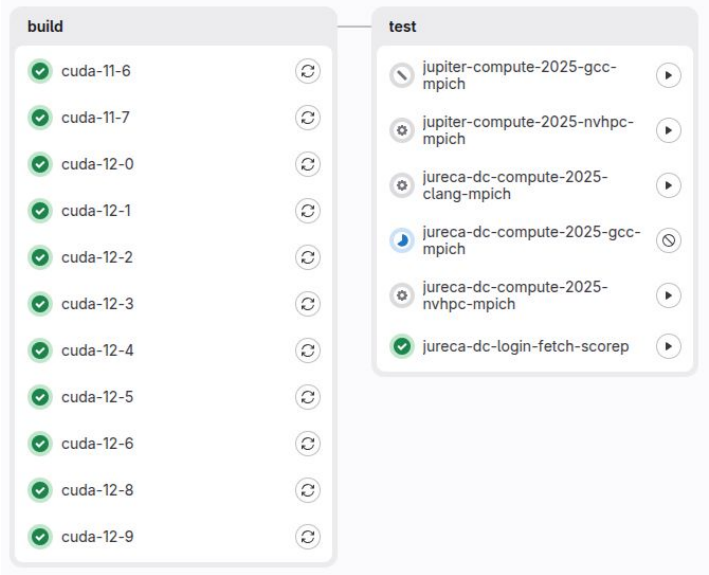
Version 10
What will be coming?

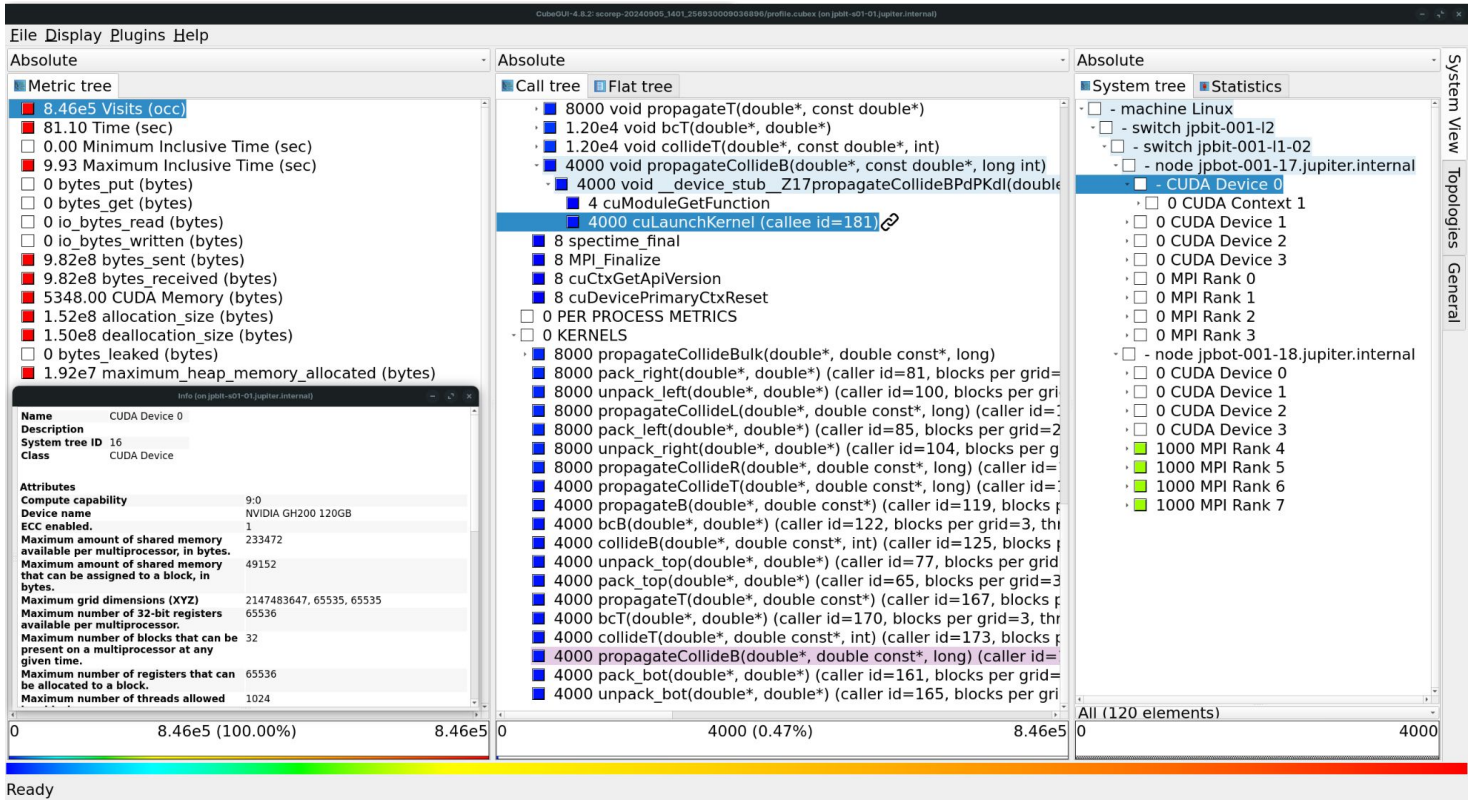
Fully rewritten CUDA adapter

- Replaces old CUDA adapter, based on CUDA 4.0
- Based on CUDA 11.6, ready for new features

Benefits

- Reduction in overhead
- Removal of pitfalls due to old CUDA versions



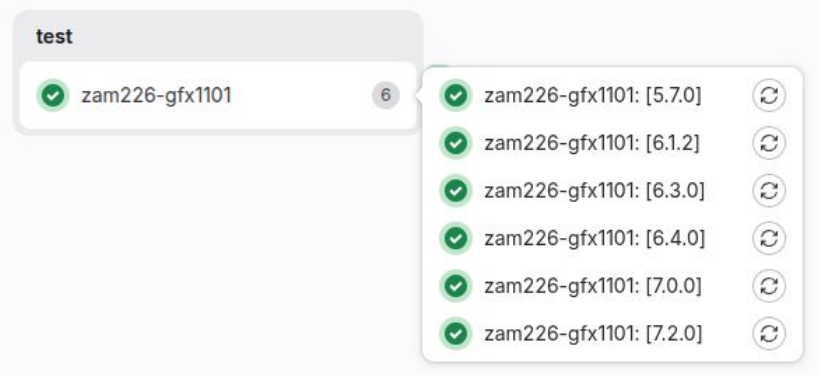


Fully rewritten ROCm adapter

- Eventually replaces HIP adapter, available since Score-P v8.0
- Based on ROCprofiler-SDK v1.0+

Benefits

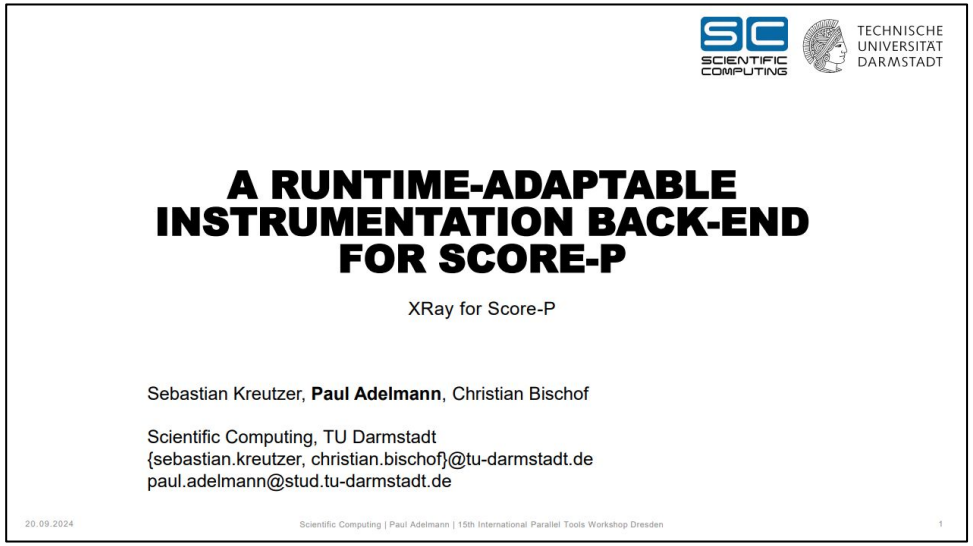
- Streamlining of GPU adapters
- Switch to new interface, allowing to add new features in v11.x



XRay compiler instrumentation

- Thanks to work / research by Sebastian Kreuzer & Paul Adelmann @ TUDa
- Yet another compiler instrumentation option, but very powerful
- Unfortunately no Fortran support…

```
$ scorep-clang minimal.c
$ objdump -h -j xray_instr_map ./a.out

./a.out:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 10 xray_instr_map 000000c0  000000000000efd0  000000000000efd0  0000efd0  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
$ ./a.out
$ cube_calltree scorep-20260421_1518_276293092750324/profile.cubex
Reading scorep-20260421_1518_276293092750324/profile.cubex... done.
a.out
  + main
  |   + foo
  |   + bar
  |   |   + foo
```

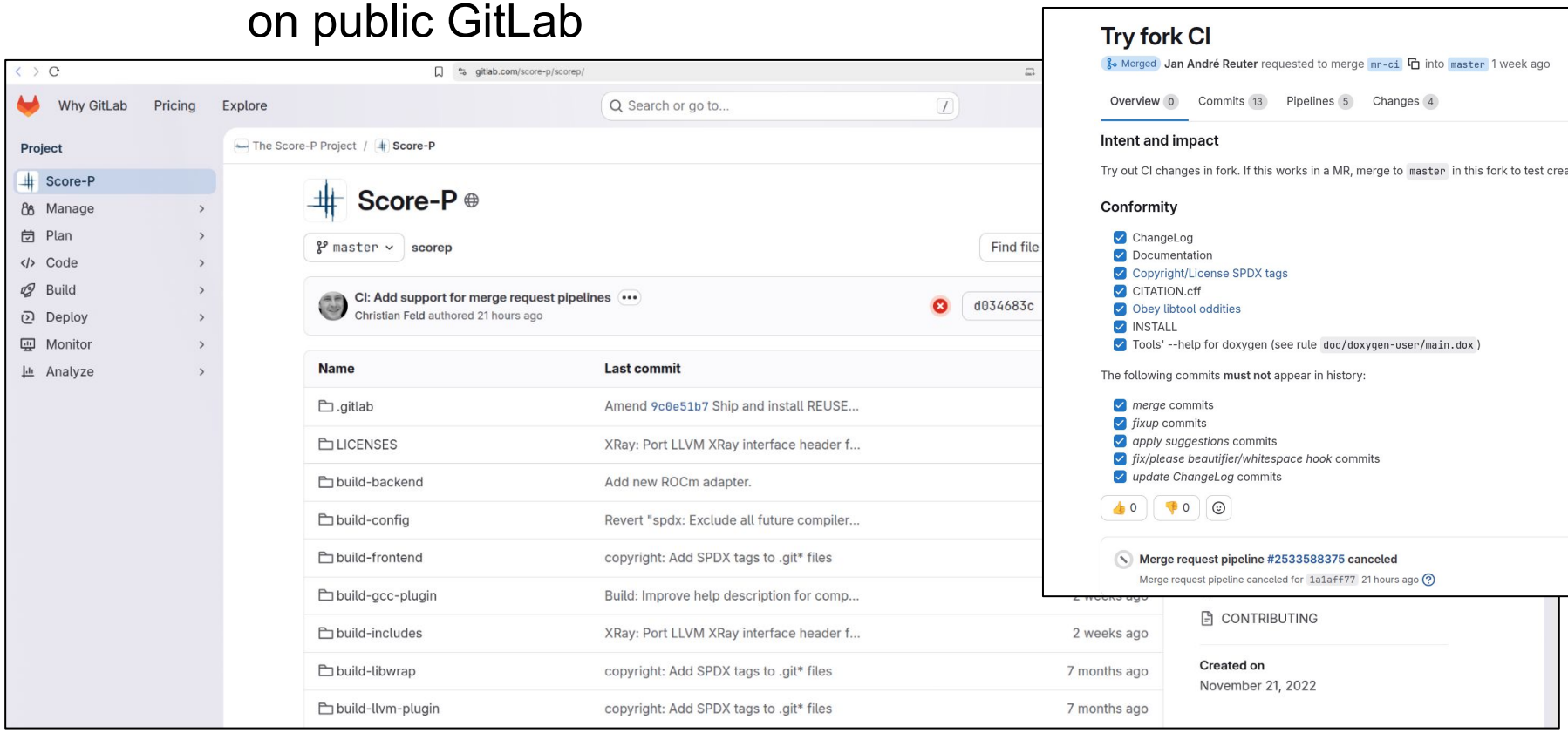


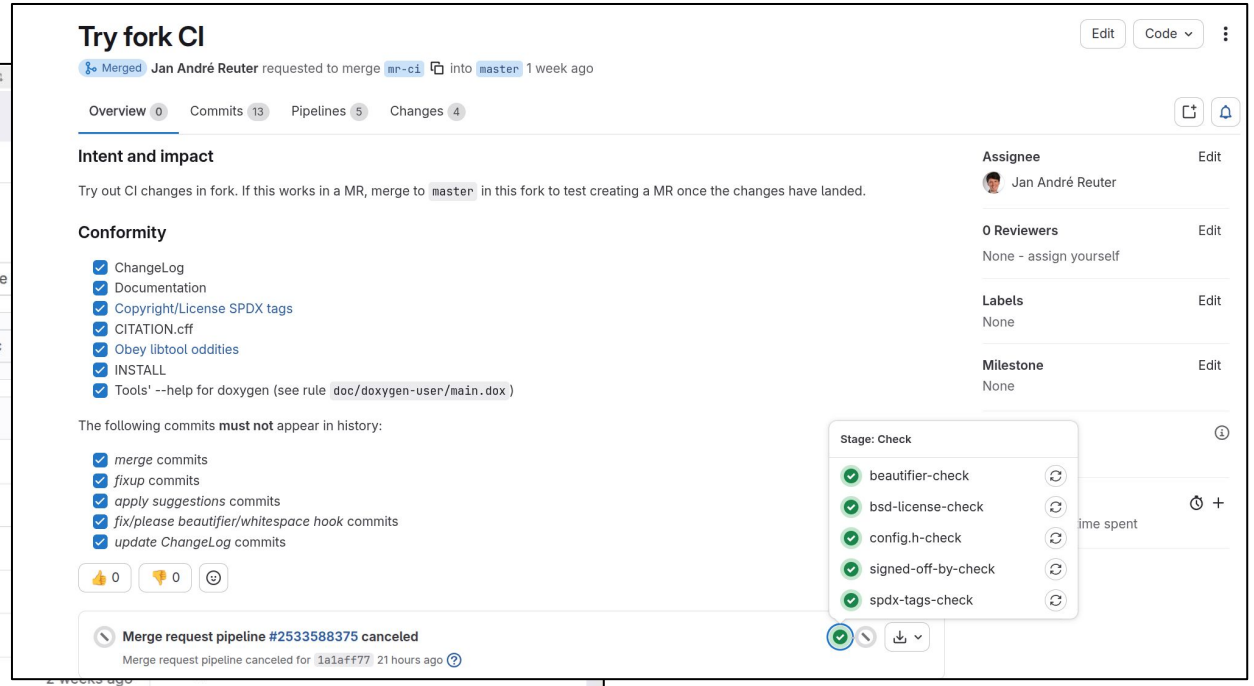
Benefits

- Enables compiler instrumentation on CCE 20.0+
- Significantly lower overhead than `__cyg_profile_func` when filtering
- Widely available (in LLVM-based compilers)

External contributions via GitLab

- November 2022: Score-P mirror published on public GitLab
  - Only mirroring `master` and release branches
- Soon: Acceptance of external contributions via merge requests on public GitLab





What am I working on?

My work topics

- Improve CUDA support
- Add & investigate support for features
  - CUDA Graphs, Sync, Unified Memory, …
  - More efficient event collection
- Interaction with OpenMP, OpenACC, Kokkos, …

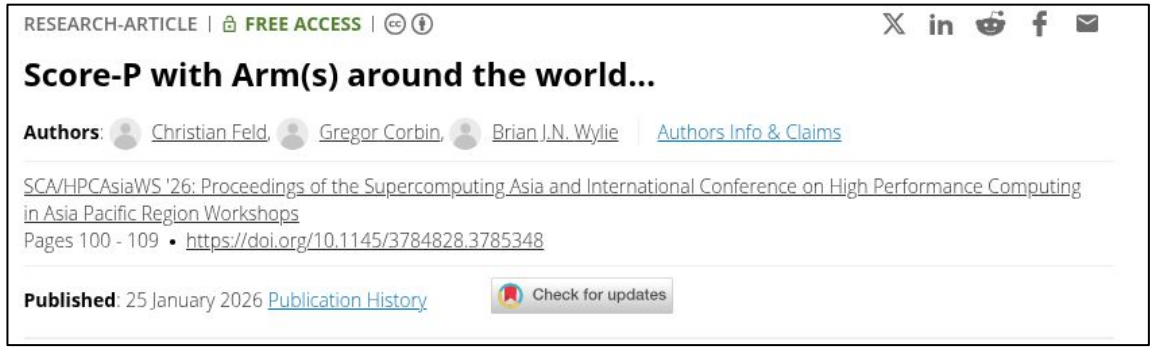
- Support for OpenMP Tools Interface
  - “Finish” host-side callbacks
  - Support accelerator events
- Join OpenMP Tools calls
- Investigate compiler support & report / fix issues

Compilers and Platforms

- Compiler instrumentation
- Investigate compiler (IBM, CCE, …) feature support
- Maintenance of existing compiler instrumentation (e.g. LLVM IR Plug-In)

More info about Score-P

Score-P
Scalable performance measurement infrastructure for parallel codes



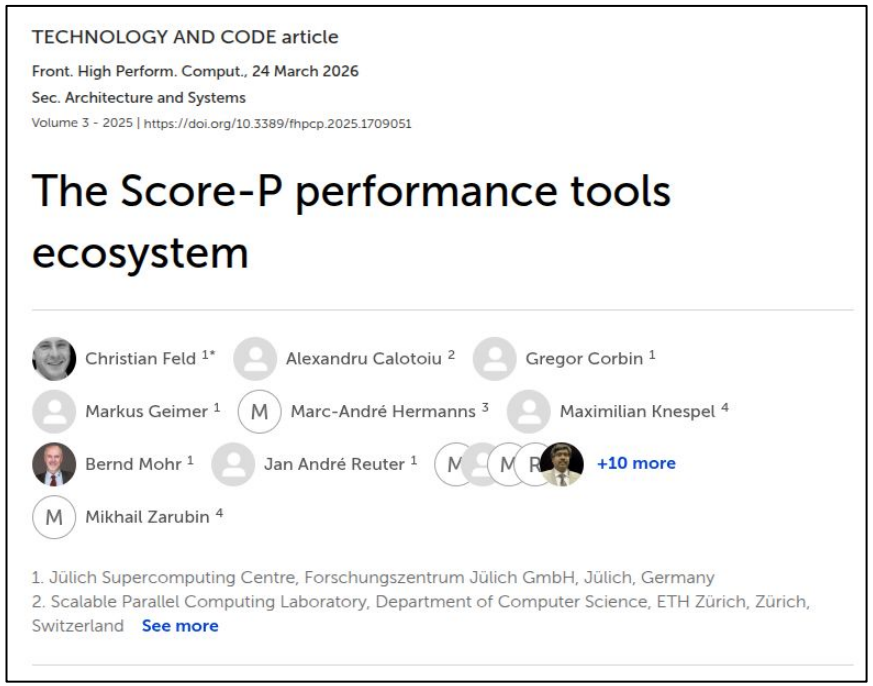
RESEARCH-ARTICLE | FREE ACCESS |

Score-P with Arm(s) around the world...

Authors: Christian Feld, Gregor Corbin, Brian J.N. Wylie | Authors Info & Claims

SCA/HPCAsiaWS '26: Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops
Pages 100 - 109 • https://doi.org/10.1145/3784828.3785348

Published: 25 January 2026 Publication History | Check for updates



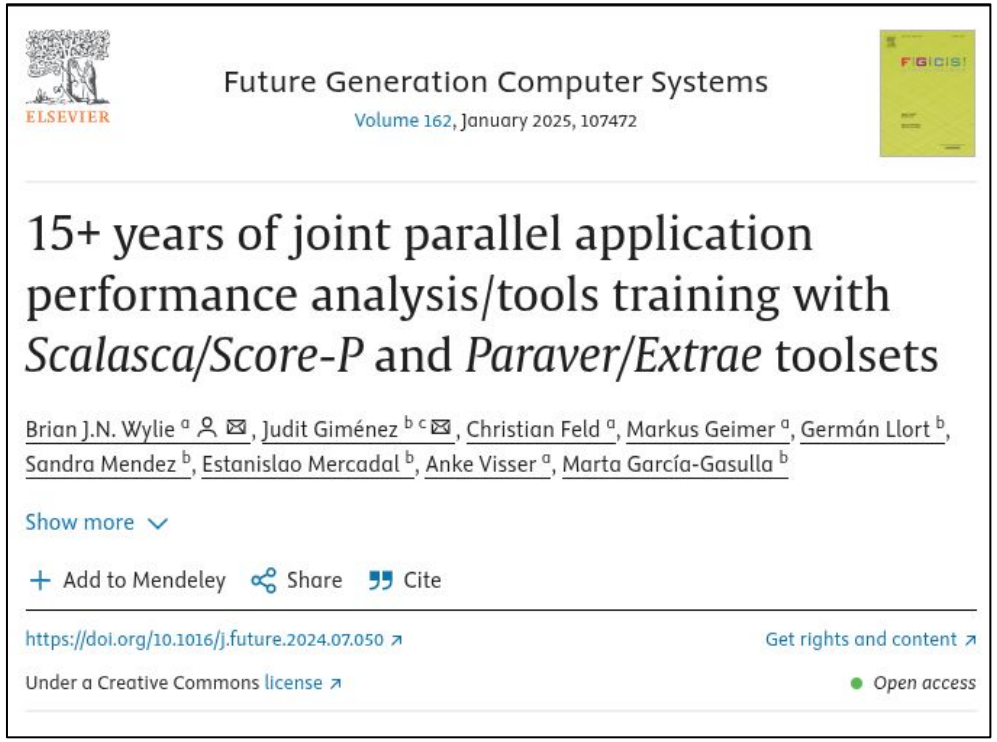
TECHNOLOGY AND CODE article

Front. High Perform. Comput., 24 March 2026
Sec. Architecture and Systems
Volume 3 - 2025 | https://doi.org/10.3389/fhpcp.2025.1709051

The Score-P performance tools ecosystem

Christian Feld [1*] Alexandru Calotoiu [2] Gregor Corbin [1]
Markus Geimer [1] Marc-André Hermanns [3] Maximilian Knespel [4]
Bernd Mohr [1] Jan André Reuter [1] +10 more
Mikhail Zarubin [4]

1. Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany
2. Scalable Parallel Computing Laboratory, Department of Computer Science, ETH Zürich, Zürich, Switzerland See more



ELSEVIER

Future Generation Computer Systems
Volume 162, January 2025, 107472

15+ years of joint parallel application performance analysis/tools training with *Scalasca/Score-P* and *Paraver/Extrae* toolsets

Brian J.N. Wylie [a], Judit Giménez [b c], Christian Feld [a], Markus Geimer [a], Germán Llort [b], Sandra Mendez [b], Estanislao Mercadal [b], Anke Visser [a], Marta García-Gasulla [b]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/j.future.2024.07.050 Get rights and content

Under a Creative Commons license Open access

JÜLICH Forschungszentrum

Obtain our software and get in contact

- Visit our web page:

  score-p.org | scalasca.org
- Check out our public Score-P GitLab mirror:

  gitlab.com/score-p/scorep
- Join our Matrix User Group:

  matrix.to/#/#score-p-user-group:hpc.rwth-aachen.de
- Available on several different platforms:

- Support:

  support@score-p.org

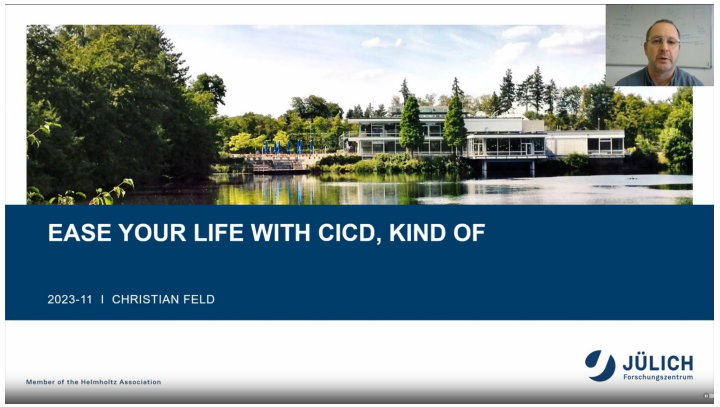
  scalasca@fz-juelich.de

THANKS FOR YOUR ATTENTION!
QUESTIONS?

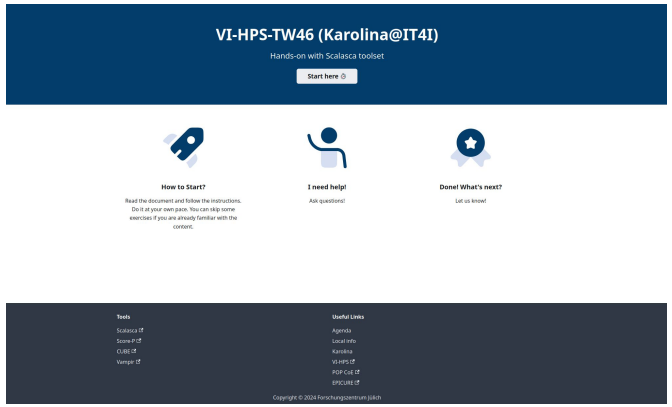
More info about our tools & development

Score-P
Scalable performance measurement infrastructure for parallel codes

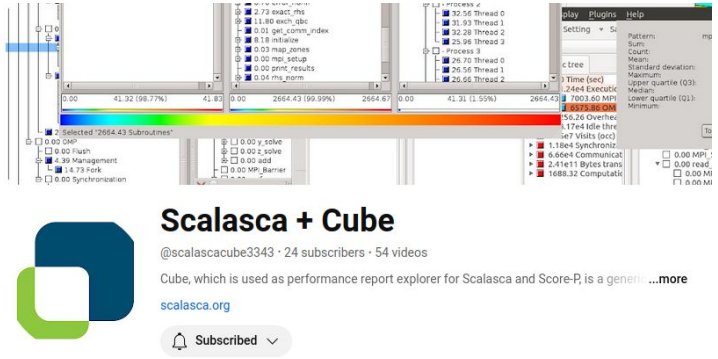
CI infrastructure



Workshop Hands-On



Cube YouTube channel



Score-P on Helmholtz.Software (incl. references)

Score-P

Score-P provides insight into massively parallel HPC applications, their communication, synchronization, I/O, and scaling behaviour to pinpoint performance bottlenecks and their causes.