

Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir

VI-HPS Team



Score-P

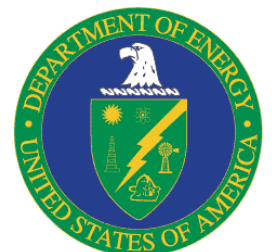


- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
 - Call-path profiling: CUBE4 data format used for data exchange
 - Event-based tracing: OTF2 data format used for data exchange
 - Online profiling: In conjunction with the Periscope Tuning Framework
- Supported parallel paradigms:
 - Multi-process: MPI, SHMEM
 - Thread-parallel: OpenMP, Pthreads
 - Accelerator-based: CUDA, OpenCL
- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Close collaboration with PRIMA project funded by DOE

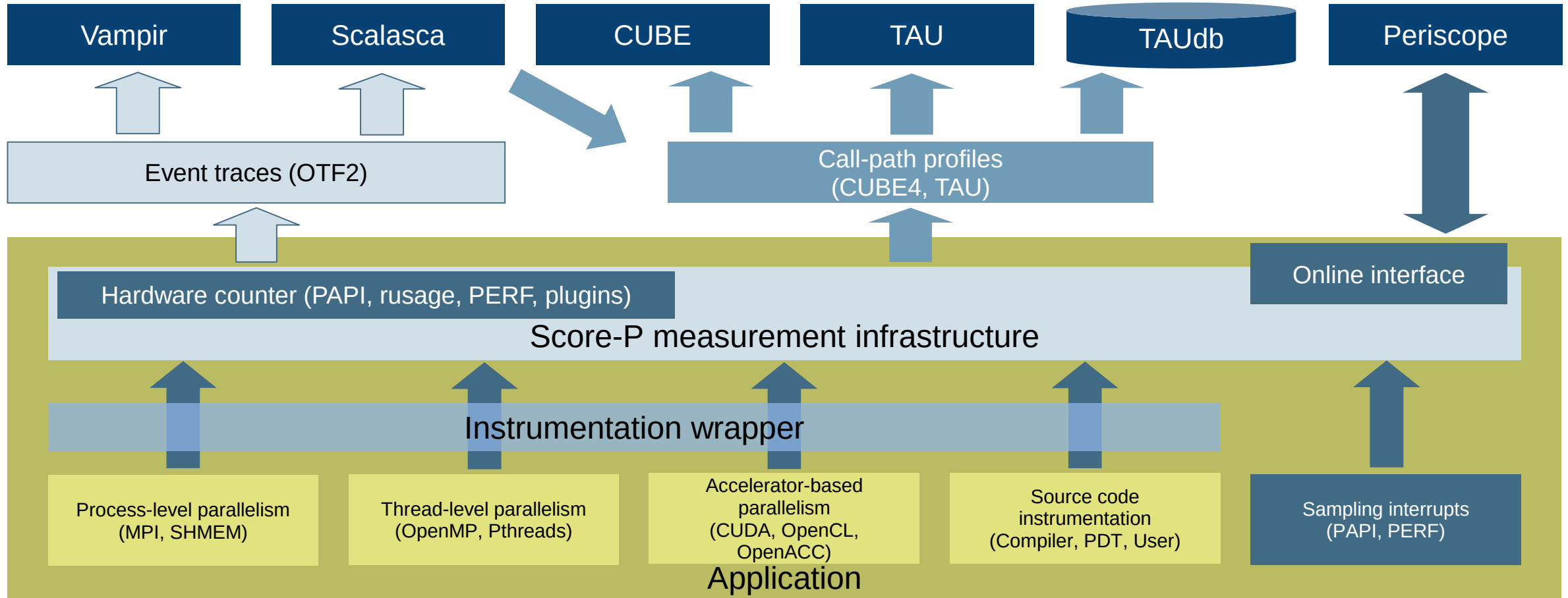
GEFÖRDERT VOM



Bundesministerium
für Bildung
und Forschung



Score-P overview



Partners

- Forschungszentrum Jülich, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Darmstadt, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA



Reference hands-on: TeaLeaf



Performance analysis steps

- Reference preparation for validation
- Program instrumentation
- Summary measurement collection
- Summary experiment scoring
- Trace measurement collection with filtering

TeaLeaf Reference Build

```
% cd reference/  
% make COMPILER=INTEL MPI_COMPILER=mpiifort C_MPI_COMPILER=mpiicc  
mpiicc -O3 -ipo -no-prec-div -restrict -fno-alias -qopenmp \  
-c ../TeaLeaf_ref/timer_c.c  
mpiifort -O3 -ipo -no-prec-div -qopenmp ../TeaLeaf_ref/data.f90 -c  
mpiifort -O3 -ipo -no-prec-div -qopenmp ../TeaLeaf_ref/definitions.f90 -c  
[...]  
mpiifort -O3 -ipo -no-prec-div -qopenmp ../TeaLeaf_ref/visit.f90 -c  
mpiifort -O3 -ipo -no-prec-div -qopenmp build_field.o calc_dt.o calc_dt_kernel.o data.o \  
definitions.o diffuse.o field_summary.o \  
[...]  
tea_leaf_ppcg.o tea_solve.o timer.o timestep.o update_halo.o update_halo_kernel.o visit.o \  
timer_c.o -o tea_leaf; echo
```

TeaLeaf reference execution

```
% sbatch --account=hpclab --reservation=aixcelerate tealeaf_ref.job
sbatch: [I] Partition set to 'cl8m' due to chosen account 'hpclab'.
sbatch: [I] Job set to exclusive, because usage of 'papi' has been requested.
Submitted batch job 11291873
% cat tealeaf_ref.11291873.out

Tea Version      1.000
  MPI Version
  OpenMP Version
Task Count       8
Thread Count:    12

Output file tea.out opened. All output will go there.
Test problem    5 is within    0.1804548E+02% of the expected solution
This test is considered PASSED
```

Save the benchmark run time
to be able to refer to it later.

Performance analysis steps

- Reference preparation for validation
- Program instrumentation
- Summary measurement collection
- Summary experiment scoring
- Trace measurement collection with filtering

TeaLeaf Score-P instrumentation build

```
% make COMPILER=INTEL MPI_COMPILER="scorep mpiifort" \  
      C_MPI_COMPILER=mpiicc  
mpiicc -O3 -ipo -no-prec-div -restrict -fno-alias -qopenmp \  
  -c ../TeaLeaf_ref/timer_c.c  
scorep mpiifort -O3 -ipo -no-prec-div -qopenmp \  
  ../TeaLeaf_ref/data.f90 -c  
scorep mpiifort -O3 -ipo -no-prec-div -qopenmp \  
  ../TeaLeaf_ref/definitions.f90 -c  
[...]  
scorep mpiifort -O3 -ipo -no-prec-div -qopenmp \  
  ../TeaLeaf_ref/visit.f90 -c  
scorep mpiifort -O3 -ipo -no-prec-div -qopenmp \  
  build_field.o calc_dt.o calc_dt_kernel.o data.o \  
  definitions.o diffuse.o field_summary.o \  
[...]  
tea_leaf_ppcg.o tea_solve.o timer.o timestep.o \  
  update_halo.o update_halo_kernel.o visit.o timer_c.o \  
-o tea_leaf; echo
```

- Prefix compiler variable with 'scorep'
- Instruments compilation units with default settings.

TeaLeaf Score-P measurement execution

```
% sbatch --account=hpclab --reservation=aixcelerate tealeaf_ref.job
sbatch: [I] Partition set to 'cl8m' due to chosen account 'hpclab'.
sbatch: [I] Job set to exclusive, because usage of 'papi' has been requested.
Submitted batch job 11291873
% cat tealeaf_ref.11291873.out

Tea Version      1.000
  MPI Version
  OpenMP Version
Task Count       8
Thread Count:    12

Output file tea.out opened. All output will go there.
Test problem    5 is within    0.1804548E+02% of the expected solution
This test is considered PASSED
```

Save the benchmark run time
to be able to refer to it later.

Measurement configuration: scorep-info

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
  [...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
  [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
  [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
  [...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
  [...]
SCOREP_METRIC_PAPI
  Description: PAPI metric names to measure
  [...]
SCOREP_METRIC_RUSAGE
  Description: Resource usage metric names to measure
  [...] More configuration variables ...]
```

- Score-P measurements are configured via environmental variables

Congratulations!?

- If you made it this far, you successfully used Score-P to
 - instrument the application
 - record its execution with a summary measurement, and
 - [optional] examine it with one the interactive analysis report explorer GUIs
- ... revealing the call-path profile annotated with
 - the “Time” metric
 - Visit counts
 - MPI message statistics (bytes sent/received)
- ... but how **good** was the measurement?
 - The measured execution produced the desired valid result
 - however, the execution took rather longer than expected!
 - even when ignoring measurement start-up/completion, therefore
 - it was probably dilated by instrumentation/measurement overhead

Performance analysis steps

- Reference preparation for validation
- Program instrumentation
- Summary measurement collection
- Summary experiment scoring
- Trace measurement collection with filtering

TeaLeaf summary analysis result scoring

```
% scorep-score scorep-/profile.cubex
```

Estimated aggregate size of event trace:

Estimated requirements for largest trace buffer (max_buf):

Estimated memory requirements (SCOREP_TOTAL_MEMORY):

(hint: When tracing set SCOREP_TOTAL_MEMORY=687MB to avoid intermediate flushes or reduce requirements using USR regions filters.)

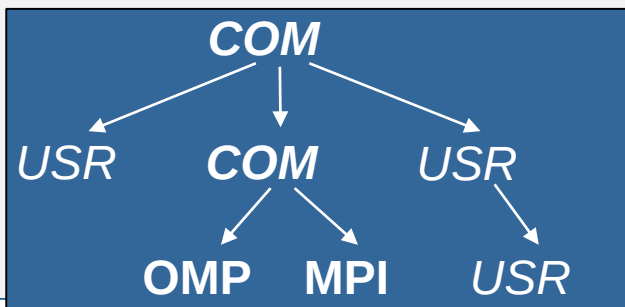
flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	694,583,945	117,797,019	8216.42	100.0	69.75	ALL
	OMP	657,597,924	110,518,776	8112.00	98.7	73.40	OMP
	COM	17,360,018	4,696,860	34.84	0.4	7.42	COM
	MPI	16,624,782	1,657,880	67.85	0.8	40.93	MPI
	USR	3,002,610	923,495	1.72	0.0	1.87	USR
	SCOREP	41	8	0.00	0.0	321.52	SCOREP

5GB
663MB
687MB

- Report scoring as textual output

5 GB total memory
690 MB per rank!

- Region/callpath classification
 - **MPI** pure MPI functions
 - **OMP** pure OpenMP regions
 - **USR** user-level computation
 - **COM** "combined" USR+OpenMP/MPI
 - **SCOREP** measurement internals
 - **ANY/ALL** aggregate of all region types



TeaLeaf summary analysis report breakdown

```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex
```

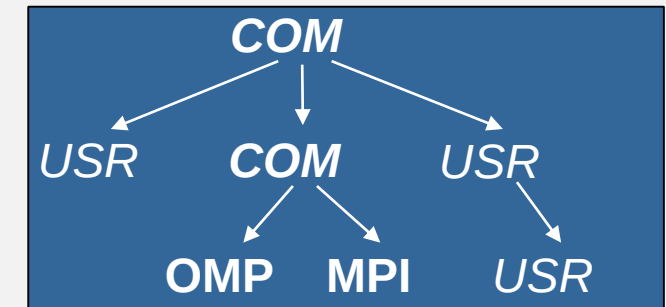
```
[...]
```

```
[...]
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	694,583,945	117,797,019	8216.42	100.0	69.75	ALL
	OMP	657,597,924	110,518,776	8112.00	98.7	73.40	OMP
	COM	17,360,018	4,696,860	34.84	0.4	7.42	COM
	MPI	16,624,782	1,657,880	67.85	0.8	40.93	MPI
	USR	3,002,610	923,495	1.72	0.0	1.87	USR
	SCOREP	41	8	0.00	0.0	321.52	SCOREP

OMP	48,094,992	3,316,896	9.91	0.1	2.99	!\$omp parallel @pack_kernel.f90:246
OMP	48,094,992	3,316,896	9.64	0.1	2.91	!\$omp parallel @pack_kernel.f90:384
OMP	48,094,992	3,316,896	9.83	0.1	2.96	!\$omp parallel @pack
OMP	48,070,980	2,210,160	6.67	0.1	3.02	!\$omp parallel @pac
OMP	48,070,980	2,210,160	7.02	0.1	3.18	!\$omp parallel @pac
OMP	48,070,980	2,210,160	6.97	0.1	3.15	!\$omp parallel @pack
OMP	48,070,980	3,315,240	9.88	0.1	2.98	!\$omp parallel @pack

```
[...]
```



First ~50 regions are
OMP and MPI.
(Blacklisting these will
loose context)

Score-P: Further information

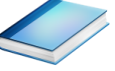
- Scalable Performance Measurement Infrastructure for Parallel Codes
 - Instrumenter, libraries, and tools to generate profile and trace measurements
 - Bundled with OTF2 (tracing), OPARI2 (OpenMP instrumentation), CubeWriter, and CubeLib (profiling)
- Available under 3-clause BSD open-source license
- Documentation & sources:
 - <http://www.score-p.org>
- User guide also part of installation:
 - `<prefix>/share/doc/scorep/pdf/scorep.pdf`
- Contact:
 - mailto: support@score-p.org



Score-P: Specialized Measurements and Analyses



Score-P filtering



```
% cat scorep.filt
SCOREP_REGION_NAMES_BEGIN
EXCLUDE
timer_*
SCOREP_REGION_NAMES_END
```

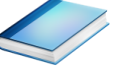
Region name
filter block
using wildcards

```
% export SCOREP_FILTERING_FILE=\
scorep.filt
```

Apply filter

- Filtering by source file name
 - All regions in files that are excluded by the filter are ignored
- Filtering by region name
 - All regions that are excluded by the filter are ignored
 - Overruled by source file filter for excluded files
- Apply filter by
 - exporting `SCOREP_FILTERING_FILE` environment variable
- Apply filter at
 - Run-time
 - Compile-time (GCC-plugin only)
 - Add cmd-line option `--instrument-filter`
 - No overhead for filtered regions but recompilation

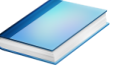
Source file name filter block



- Keywords
 - Case-sensitive
 - SCOREP_FILE_NAMES_BEGIN, SCOREP_FILE_NAMES_END
 - Define the source file name filter block
 - Block contains EXCLUDE, INCLUDE rules
 - EXCLUDE, INCLUDE rules
 - Followed by one or multiple white-space separated source file names
 - Names can contain bash-like wildcards *, ?, []
 - Unlike bash, * may match a string that contains slashes
 - EXCLUDE, INCLUDE rules are applied in sequential order
 - Regions in source files that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_FILE_NAMES_BEGIN
    # by default, everything is included
    EXCLUDE */foo/bar*
    INCLUDE */filter_test.c
SCOREP_FILE_NAMES_END
```

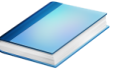
Region name filter block



- Keywords
 - Case-sensitive
 - SCOREP_REGION_NAMES_BEGIN,
SCOREP_REGION_NAMES_END
 - Define the region name filter block
 - Block contains EXCLUDE, INCLUDE rules
 - EXCLUDE, INCLUDE rules
 - Followed by one or multiple white-space separated region names
 - Names can contain bash-like wildcards *, ?, []
 - EXCLUDE, INCLUDE rules are applied in sequential order
 - Regions that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_REGION_NAMES_BEGIN
    # by default, everything is included
    EXCLUDE *
    INCLUDE bar foo
           baz
           main
SCOREP_REGION_NAMES_END
```

Region name filter block, mangling

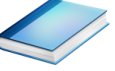


- Name mangling
 - Filtering based on names seen by the measurement system
 - Dependent on compiler
 - Actual name may be mangled
- scorep-score names as starting point
(e.g. `matvec_sub_`)
 - Use `*` for Fortran trailing underscore(s) for portability
 - Use `?` and `*` as needed for full signatures or overloading

```
void bar(int* a) {  
    *a++;  
}  
int main() {  
    int i = 42;  
    bar(&i);  
    return 0;  
}
```

```
# filter bar:  
# for gcc-plugin, scorep-score  
# displays 'void bar(int*)',  
# other compilers may differ  
  
SCOREP_REGION_NAMES_BEGIN  
    EXCLUDE void?bar(int?)  
SCOREP_REGION_NAMES_END
```

Mastering build systems



- Hooking up the Score-P instrumenter `scorep` into complex build environments like *Autotools* or *CMake* was always challenging
- Score-P provides convenience wrapper scripts to simplify this (since Score-P 2.0)
- *Autotools* and *CMake* need the used compiler already in the *configure step*, but instrumentation should not happen in this step, only in the *build step*

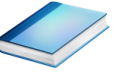
```
% SCOREP_WRAPPER=off \  
> cmake .. \  
> -DCMAKE_C_COMPILER=scorep-icc \  
> -DCMAKE_CXX_COMPILER=scorep-icpc
```

Disable instrumentation in the *configure step*

Specify the wrapper scripts as the compiler to use

- Allows to pass addition options to the Score-P instrumenter and the compiler via environment variables without modifying the *Makefiles*
- Run `scorep-wrapper --help` for a detailed description and the available wrapper scripts of the Score-P installation

Mastering C++ applications



- Automatic compiler instrumentation greatly disturbs C++ applications because of frequent/short function calls => Use sampling instead
- Novel combination of sampling events and instrumentation of MPI, OpenMP, ...
 - Sampling replaces compiler instrumentation (instrument with `--nocompiler` to further reduce overhead) => Filtering not needed anymore
 - Instrumentation is used to get accurate times for parallel activities to still be able to identify patterns of inefficiencies

- Supports profile and trace generation

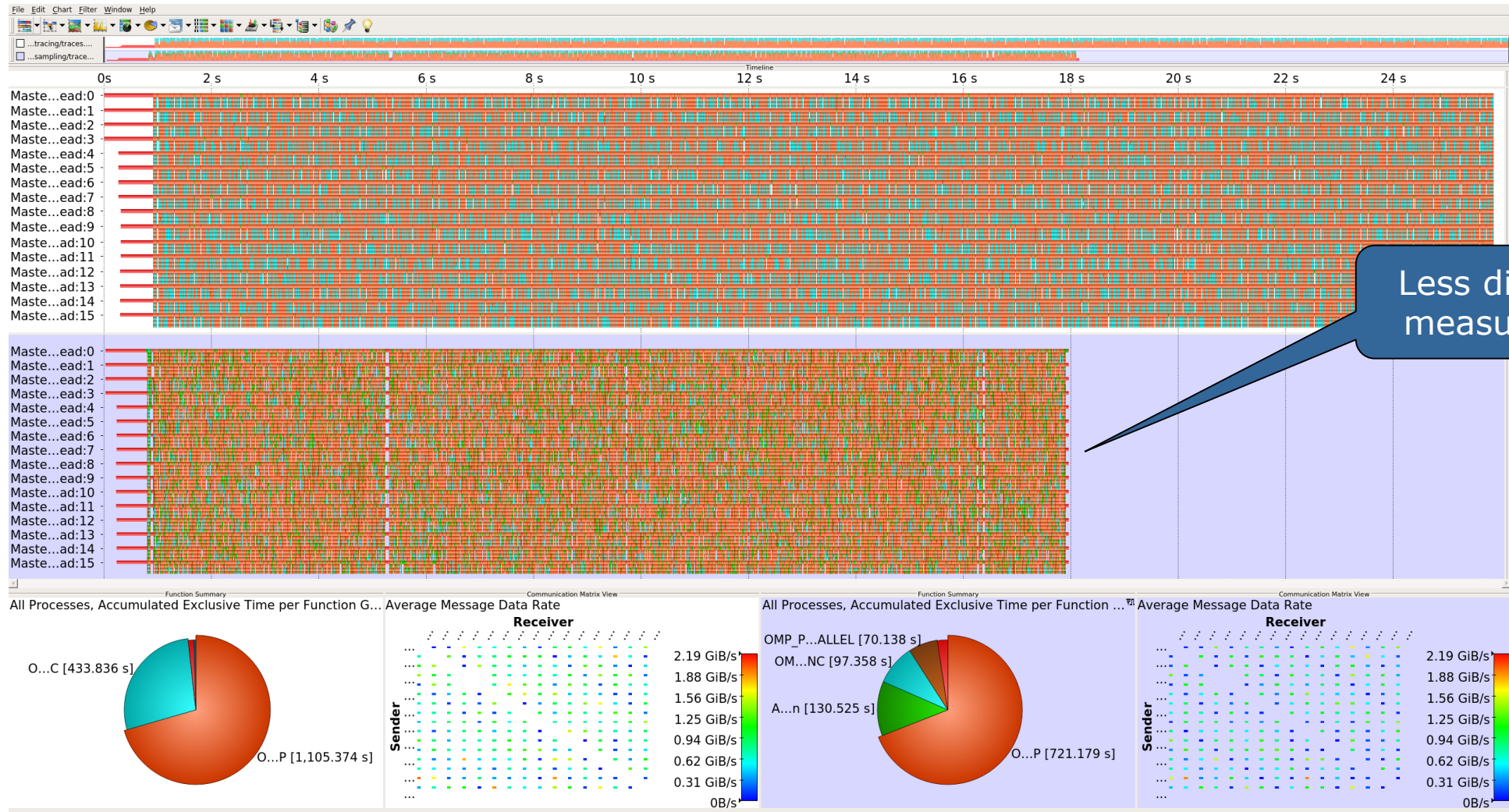
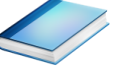
```
% export SCOREP_ENABLE_UNWINDING=true
% # use the default sampling frequency
% #export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000

% OMP_NUM_THREADS=4 mpiexec -np 4 ./bt-mz_W.4
```

- Set new configuration variable to enable sampling

- Available since Score-P 2.0, only x86-64 supported currently

Mastering C++ applications



Wrapping calls to 3rd party libraries



- Enables users to install library wrappers for any C/C++ library
- Intercept calls to a library API
 - no need to either build the library with Score-P or add manual instrumentation to the application using the library
 - no need to access the source code of the library, header and library files suffice
- Score-P needs to be executed with `--libwrap=...`
- Execute `scorep-libwrap-init` for directions:

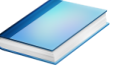
Step 1: Initialize the working directory
Step 2: Add library headers
Step 3: Create a simple example application
Step 4: Further configure the build parameters
Step 5: Build the wrapper
Step 6: Verify the wrapper
Step 7: Install the wrapper
Step 8: Verify the installed wrapper

Only once

Often

Step 9: Use the wrapper

Wrapping calls to 3rd party libraries



- Generate your own library wrappers by telling `scorep-libwrap-init` how you would compile and link an application, e.g. using FFTW

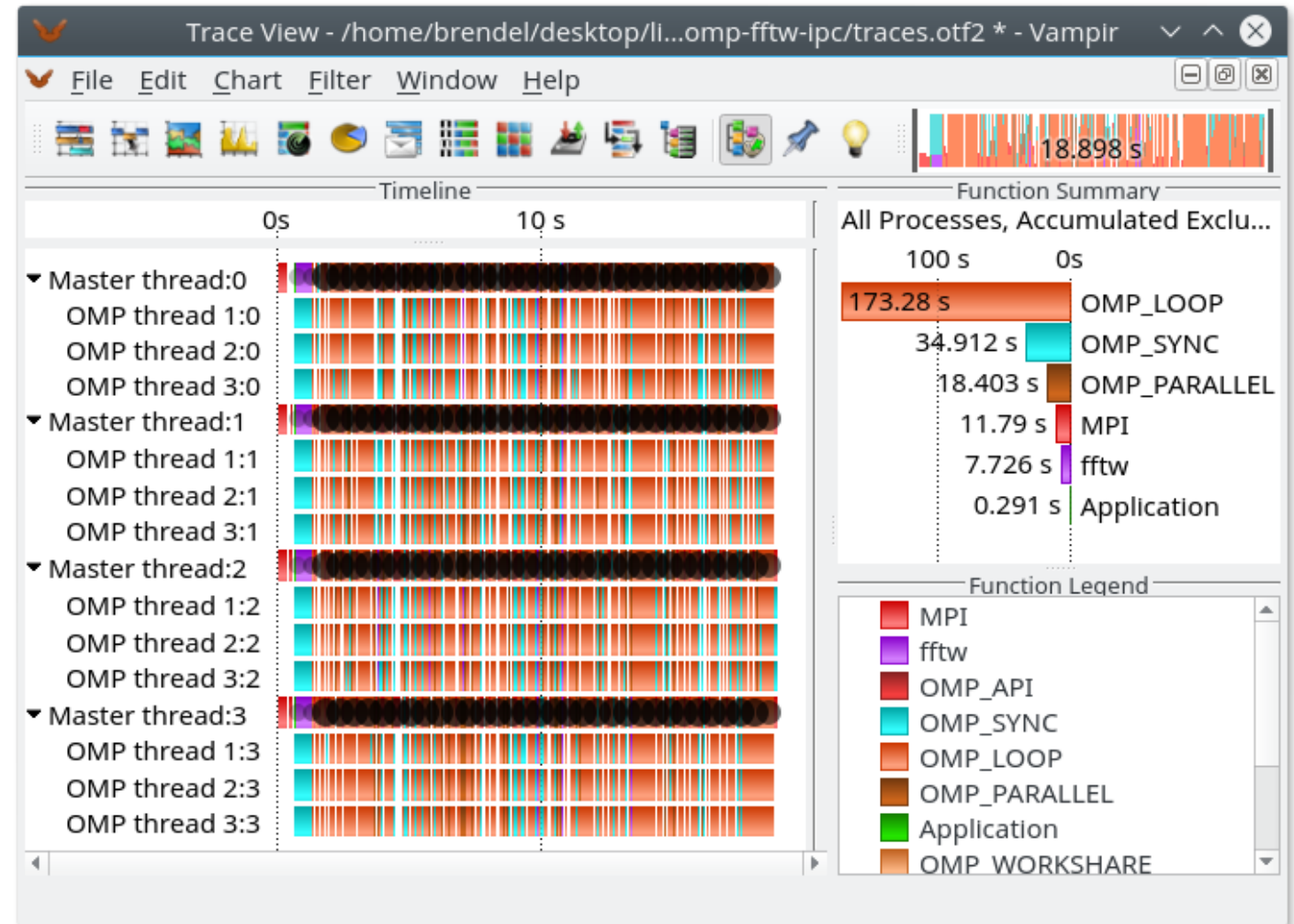
```
% scorep-libwrap-init      \  
>  --name=fftw             \  
>  --prefix=$PREFIX        \  
>  -x c                    \  
>  --cppflags="-O3 -DNDEBUG -openmp -I$FFTW_INC" \  
>  --ldflags="-L$FFTW_LIB"  \  
>  --libs="-lfftw3f -lfftw3" \  
>  working_directory
```

- Generate and build wrapper

```
% cd working_directory  
% ls                # (Check README.md for instructions)  
% make              # Generate and build wrapper  
% make check        # See if header analysis matches symbols  
% make install      #  
% make installcheck # More checks: Linking etc.
```

Wrapping calls to 3rd party libraries

- MPI + OpenMP
- Calls to FFTW library



Mastering application memory usage



- Determine the maximum heap usage per process
- Find high frequent small allocation patterns
- Find memory leaks
- Support for:
 - C, C++, MPI, and SHMEM (Fortran only for GNU Compilers)
 - Profile and trace generation (profile recommended)
 - Memory leaks are recorded only in the profile
 - Resulting traces are not supported by Scalasca yet

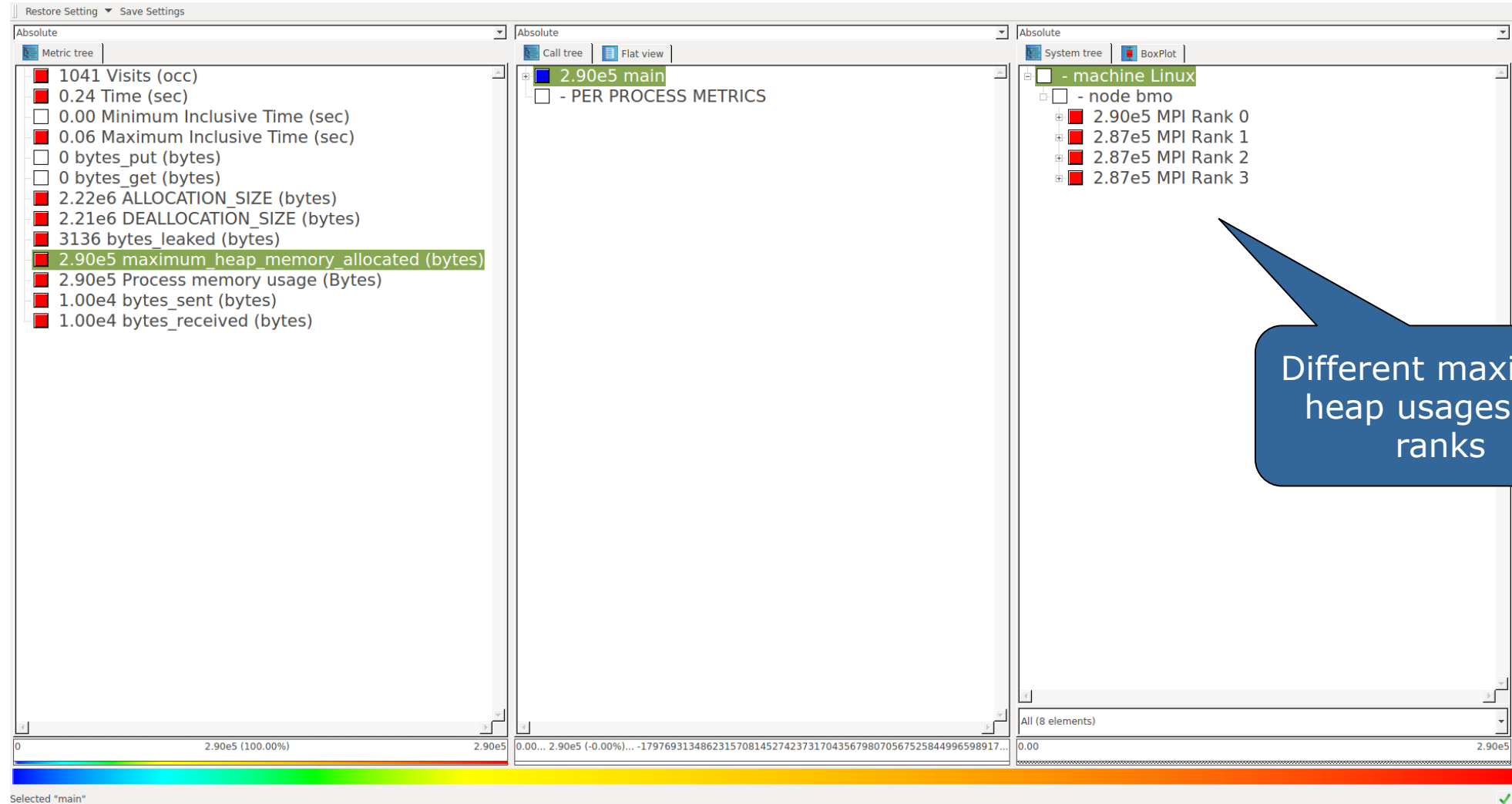
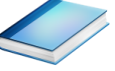
```
% export SCOREP_MEMORY_RECORDING=true
% export SCOREP_MPI_MEMORY_RECORDING=true

% OMP_NUM_THREADS=4 mpiexec -np 4 ./bt-mz_W.4
```

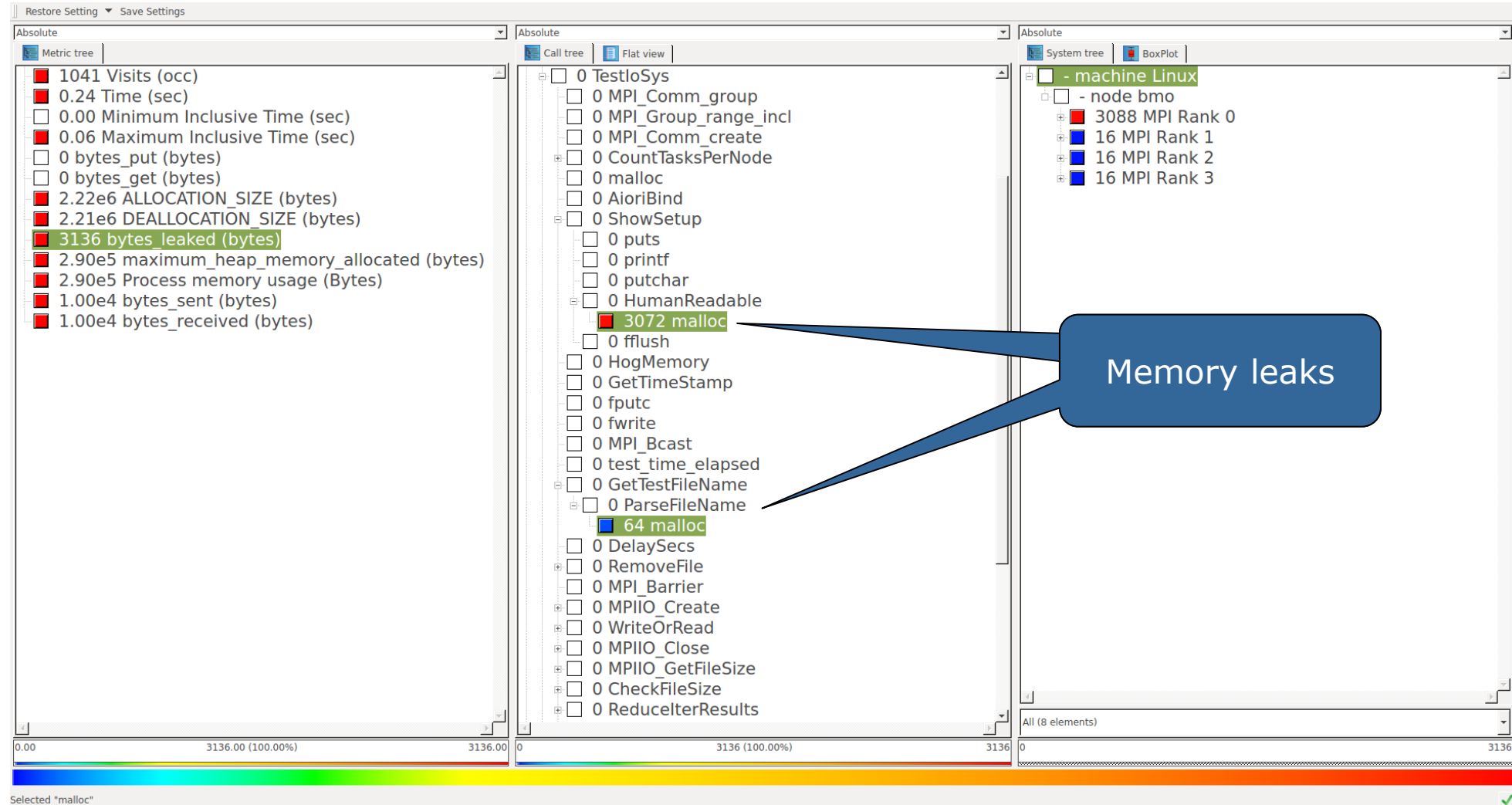
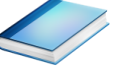
- Set new configuration variable to enable memory recording

- Available since Score-P 2.0

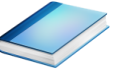
Mastering application memory usage



Mastering application memory usage



Mastering heterogeneous applications



- Record CUDA applications and device activities

```
% export SCOREP_CUDA_ENABLE=gpu,kernel,idle
```

- Record OpenCL applications and device activities

```
% export SCOREP_OPENCL_ENABLE=api,kernel
```

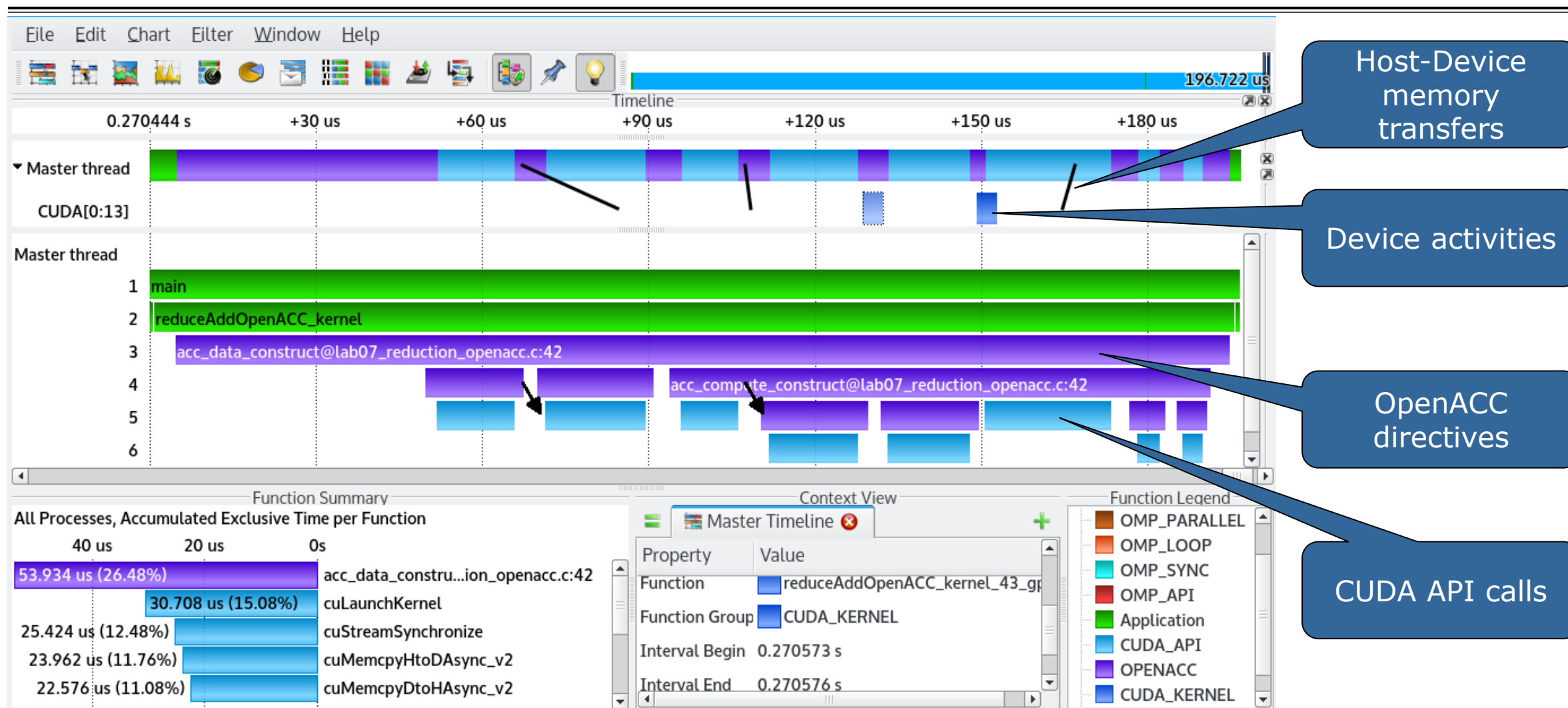
- Record OpenACC applications

```
% export SCOREP_OPENACC_ENABLE=yes
```

- Can be combined with CUDA if it is a NVIDIA device

```
% export SCOREP_CUDA_ENABLE=kernel
```

Mastering heterogeneous applications



Enriching measurements with performance counters



- Record metrics from PAPI:

```
% export SCOREP_METRIC_PAPI=PAPI_TOT_CYC
% export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_TCM
```

- Use PAPI tools to get available metrics and valid combinations:

```
% papi_avail
% papi_native_avail
```

- Record metrics from Linux perf:

```
% export SCOREP_METRIC_PERF=cpu-cycles
% export SCOREP_METRIC_PERF_PER_PROCESS=LLC-load-misses
```

- Use the `perf` tool to get available metrics and valid combinations:

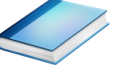
```
% perf list
```

- Write your own metric plugin

- Repository of available plugins: <https://github.com/score-p>

Only the master thread records the metric (assuming all threads of the process access the same L3 cache)

Score-P user instrumentation API



- No replacement for automatic compiler instrumentation
- Can be used to further subdivide functions
 - E.g., multiple loops inside a function
- Can be used to partition application into coarse grain phases
 - E.g., initialization, solver, & finalization
- Enabled with `--user` flag to Score-P instrumenter
- Available for Fortran / C / C++

Score-P user instrumentation API (Fortran)



```
#include "scorep/SCOREP_User.inc"

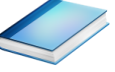
subroutine foo(...)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! Some code...
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                           SCOREP_USER_REGION_TYPE_LOOP )

  do i=1,100
    [...]
  end do
  SCOREP_USER_REGION_END( solve )
  ! Some more code...
end subroutine
```

- Requires processing by the C preprocessor
 - For most compilers, this can be automatically achieved by having an uppercase file extension, e.g., `main.F` or `main.F90`

Score-P user instrumentation API (C/C++)

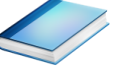


```
#include "scorep/SCOREP_User.h"

void foo()
{
    /* Declarations */
    SCOREP_USER_REGION_DEFINE( solve )

    /* Some code... */
    SCOREP_USER_REGION_BEGIN( solve, "<solver>",
                             SCOREP_USER_REGION_TYPE_LOOP )
    for (i = 0; i < 100; i++)
    {
        [...]
    }
    SCOREP_USER_REGION_END( solve )
    /* Some more code... */
}
```

Score-P user instrumentation API (C++)

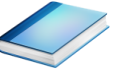


```
#include "scorep/SCOREP_User.h"

void foo()
{
    // Declarations

    // Some code...
    {
        SCOREP_USER_REGION( "<solver>",
                           SCOREP_USER_REGION_TYPE_LOOP )
        for (i = 0; i < 100; i++)
        {
            [...]
        }
    }
    // Some more code...
}
```

Score-P measurement control API



- Can be used to temporarily disable measurement for certain intervals
 - Annotation macros ignored by default
 - Enabled with `--user` flag

```
#include "scorep/SCOREP_User.inc"

subroutine foo(...)
  ! Some code...
  SCOREP_RECORDING_OFF()
  ! Loop will not be measured
  do i=1,100
    [...]
  end do
  SCOREP_RECORDING_ON()
  ! Some more code...
end subroutine
```

Fortran (requires C preprocessor)

```
#include "scorep/SCOREP_User.h"

void foo(...) {
  /* Some code... */
  SCOREP_RECORDING_OFF()
  /* Loop will not be measured */
  for (i = 0; i < 100; i++) {
    [...]
  }
  SCOREP_RECORDING_ON()
  /* Some more code... */
}
```

C / C++