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

VI-HPS Team

Score-P
Scalable performance measurement infrastructure for parallel codes

Score-P: Specialized Measurements and Analyses

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

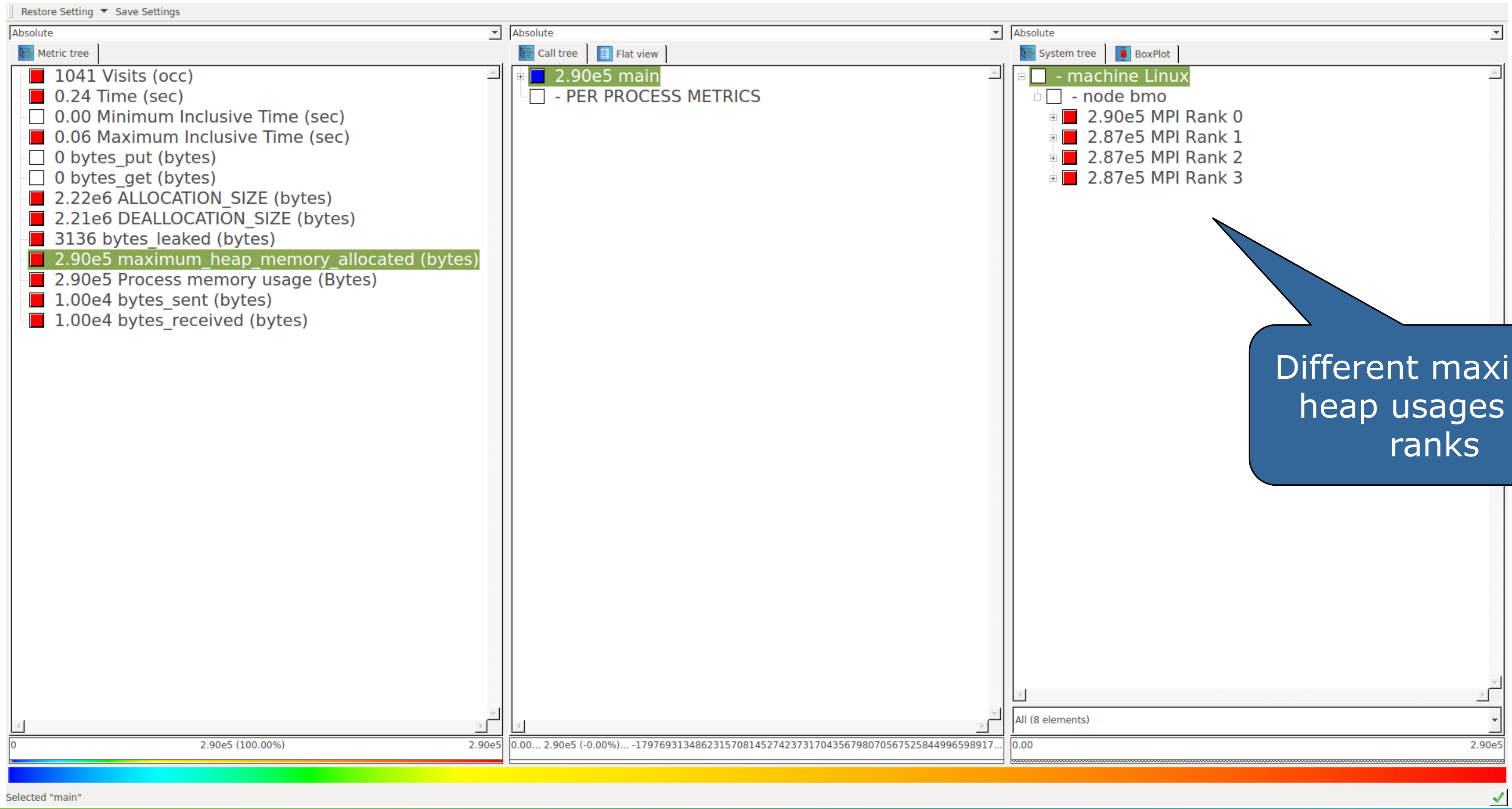
```
% export SCOREP_MEMORY_RECORDING=true
% export SCOREP_MPI_MEMORY_RECORDING=true

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

- Set new configuration variable to enable memory recording

- Available since v2.0

Mastering application memory usage



1041 Visits (occ)
0.24 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.06 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
2.22e6 ALLOCATION_SIZE (bytes)
2.21e6 DEALLOCATION_SIZE (bytes)
3136 bytes_leaked (bytes)
2.90e5 maximum_heap_memory_allocated (bytes)
2.90e5 Process memory usage (Bytes)
1.00e4 bytes_sent (bytes)
1.00e4 bytes_received (bytes)

2.90e5 main
- PER PROCESS METRICS

- machine Linux
- node bmo
2.90e5 MPI Rank 0
2.87e5 MPI Rank 1
2.87e5 MPI Rank 2
2.87e5 MPI Rank 3

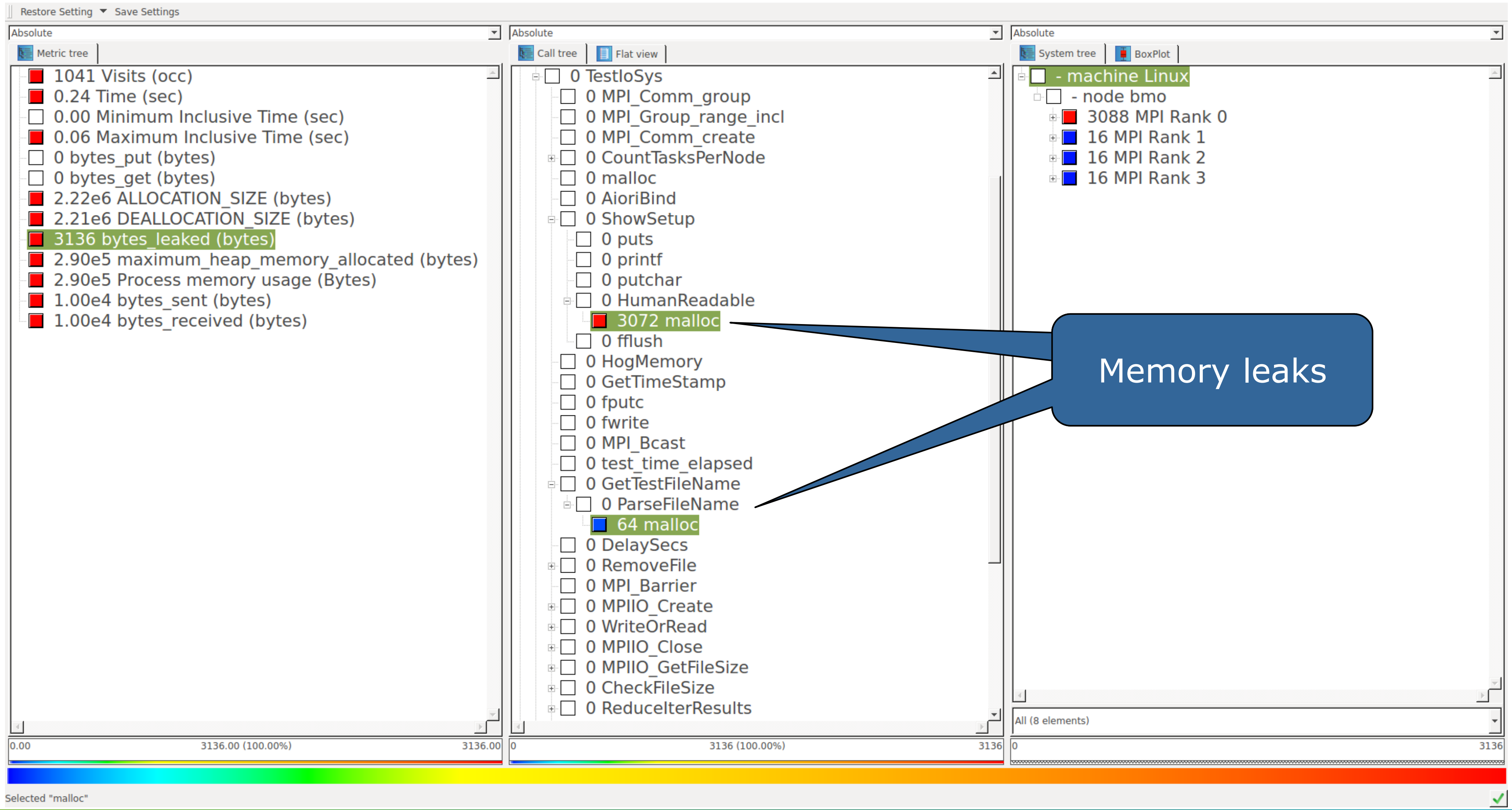
All (8 elements)

0 2.90e5 (100.00%) 2.90e5
0.00... 2.90e5 (-0.00%)... -17976931348623157081452742373170435679807056752584499659891 7...
0.00 2.90e5

Selected "main"

Different maximum heap usages per ranks

Mastering application memory usage



Restore Setting ▾ Save Settings

Absolute | Absolute | Absolute

Metric tree | Call tree | Flat view | System tree | BoxPlot

```
1041 Visits (occ)
0.24 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.06 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
2.22e6 ALLOCATION_SIZE (bytes)
2.21e6 DEALLOCATION_SIZE (bytes)
3136 bytes_leaked (bytes)
2.90e5 maximum_heap_memory_allocated (bytes)
2.90e5 Process memory usage (Bytes)
1.00e4 bytes_sent (bytes)
1.00e4 bytes_received (bytes)
```

```
0 TestIoSys
  0 MPI_Comm_group
  0 MPI_Group_range_incl
  0 MPI_Comm_create
  0 CountTasksPerNode
  0 malloc
  0 AioriBind
  0 ShowSetup
    0 puts
    0 printf
    0 putchar
    0 HumanReadable
      3072 malloc
    0 fflush
  0 HogMemory
  0 GetTimeStamp
  0 fputc
  0 fwrite
  0 MPI_Bcast
  0 test_time_elapsed
  0 GetTestFileName
    0 ParseFileName
      64 malloc
  0 DelaySecs
  0 RemoveFile
  0 MPI_Barrier
  0 MPIIO_Create
  0 WriteOrRead
  0 MPIIO_Close
  0 MPIIO_GetFileSize
  0 CheckFileSize
  0 ReduceIterResults
```

```
- machine Linux
  - node bmo
    3088 MPI Rank 0
    16 MPI Rank 1
    16 MPI Rank 2
    16 MPI Rank 3
```

All (8 elements)

0.00 3136.00 (100.00%) 3136.00 | 0 3136 (100.00%) 3136 | 0 3136

Selected "malloc"

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

- Can be used to partition application into coarse grain phases
  - E.g., initialization, solver, & finalization

- Can be used to further subdivide functions
  - E.g., multiple loops inside a function

- 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++

Mastering heterogeneous applications

- Record CUDA applications and device activities

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

Idle is an artificial region defined as outside of kernel time

- Record OpenCL applications and device activities

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

- Record OpenACC applications

```
% export SCOREP_OPENACC_ENABLE=yes
```

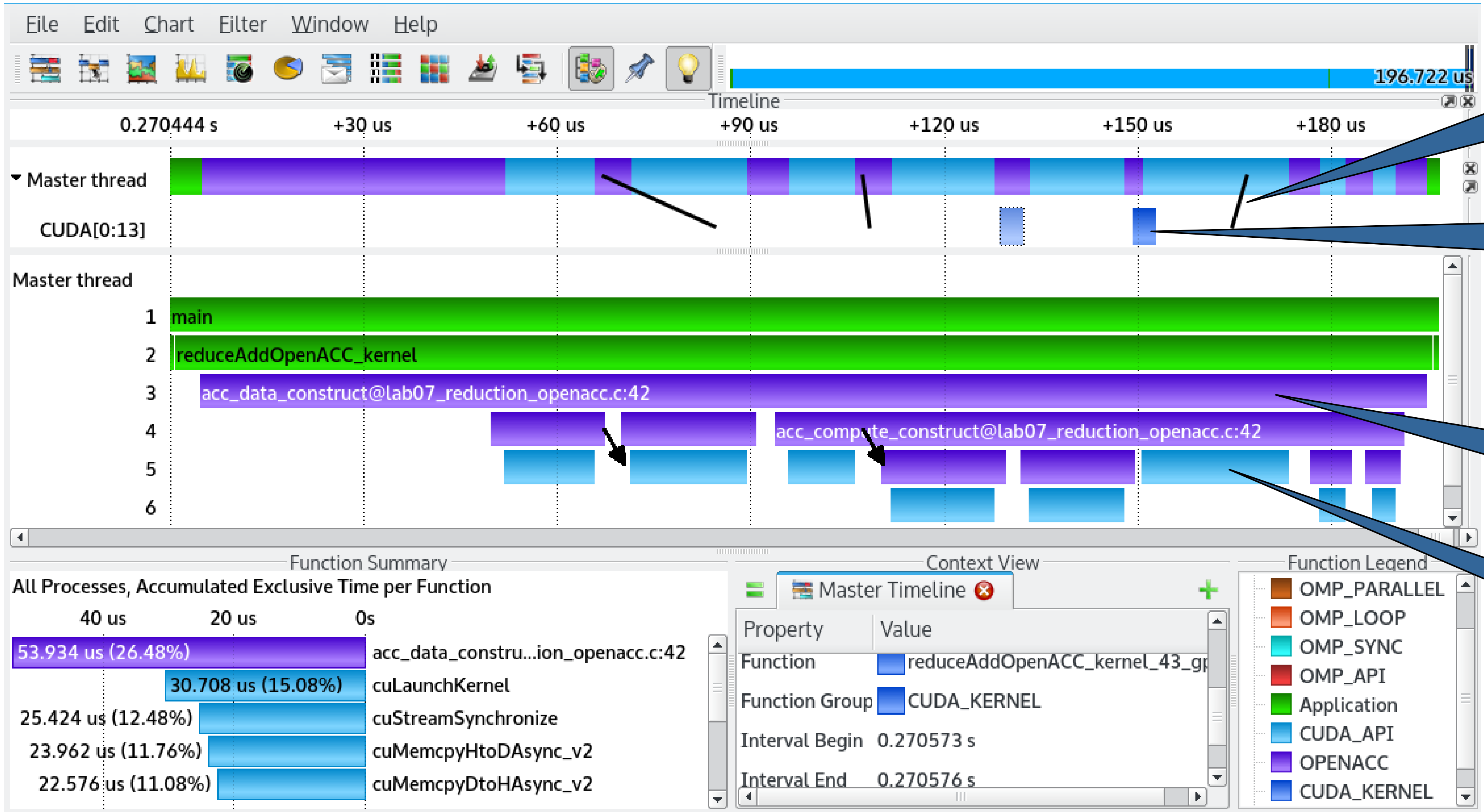
  - Can be combined with CUDA if it is a NVIDIA device

```
% export SCOREP_CUDA_ENABLE=kernel
```

  - Check **`scorep-info config-vars –full`** for a wide range of further options and default values

Adding options will increase overhead to a varying degree

Mastering heterogeneous applications


File Edit Chart Filter Window Help
196.722 us
Timeline
0.270444 s
+30 us
+60 us
+90 us
+120 us
+150 us
+180 us
Master thread
CUDA[0:13]
Master thread
1 main
2 reduceAddOpenACC_kernel
3 acc_data_construct@lab07_reduction_openacc.c:42
4 acc_compute_construct@lab07_reduction_openacc.c:42
5
6
Function Summary
Context View
Function Legend

All Processes, Accumulated Exclusive Time per Function

40 us 20 us 0s

53.934 us (26.48%)	acc_data_constru...ion_openacc.c:42
30.708 us (15.08%)	cuLaunchKernel
25.424 us (12.48%)	cuStreamSynchronize
23.962 us (11.76%)	cuMemcpyHtoDAsync_v2
22.576 us (11.08%)	cuMemcpyDtoHAsync_v2

Master Timeline

Property	Value
Function	reduceAddOpenACC_kernel_43_gp
Function Group	CUDA_KERNEL
Interval Begin	0.270573 s
Interval End	0.270576 s

🟫	OMP_PARALLEL
🟧	OMP_LOOP
🟦	OMP_SYNC
🟥	OMP_API
🟩	Application
🟦	CUDA_API
🟪	OPENACC
🟦	CUDA_KERNEL

Host-Device memory transfers

Device activities

OpenACC directives

CUDA API calls

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 identifies patterns of inefficiencies
- Supports profile and trace generation

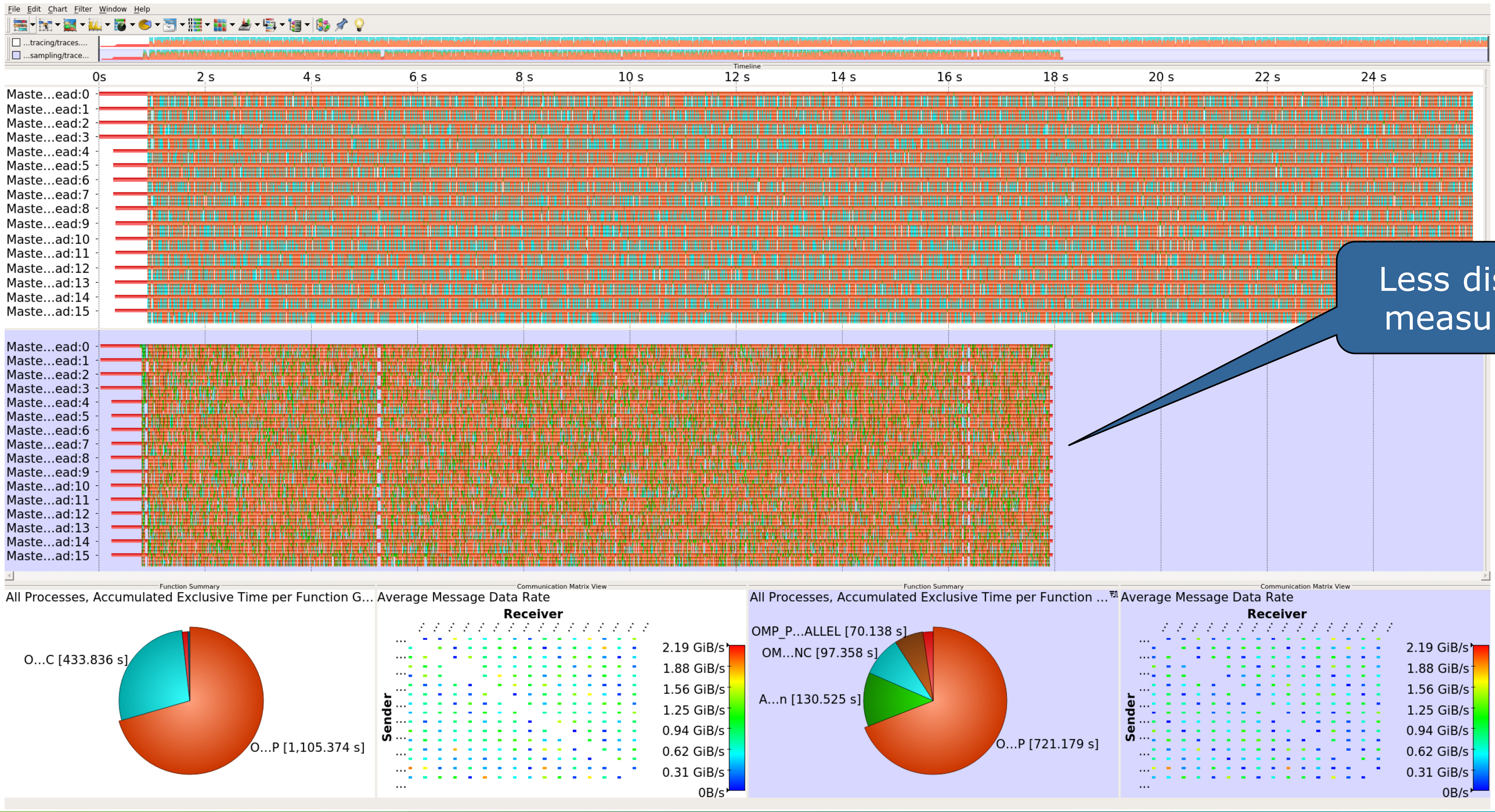
```
% export SCOREP_ENABLE_UNWINDING=true
% # use the default sampling frequency
% #export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000

% 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



Less disturbed measurement

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:

Only once

```
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
```

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

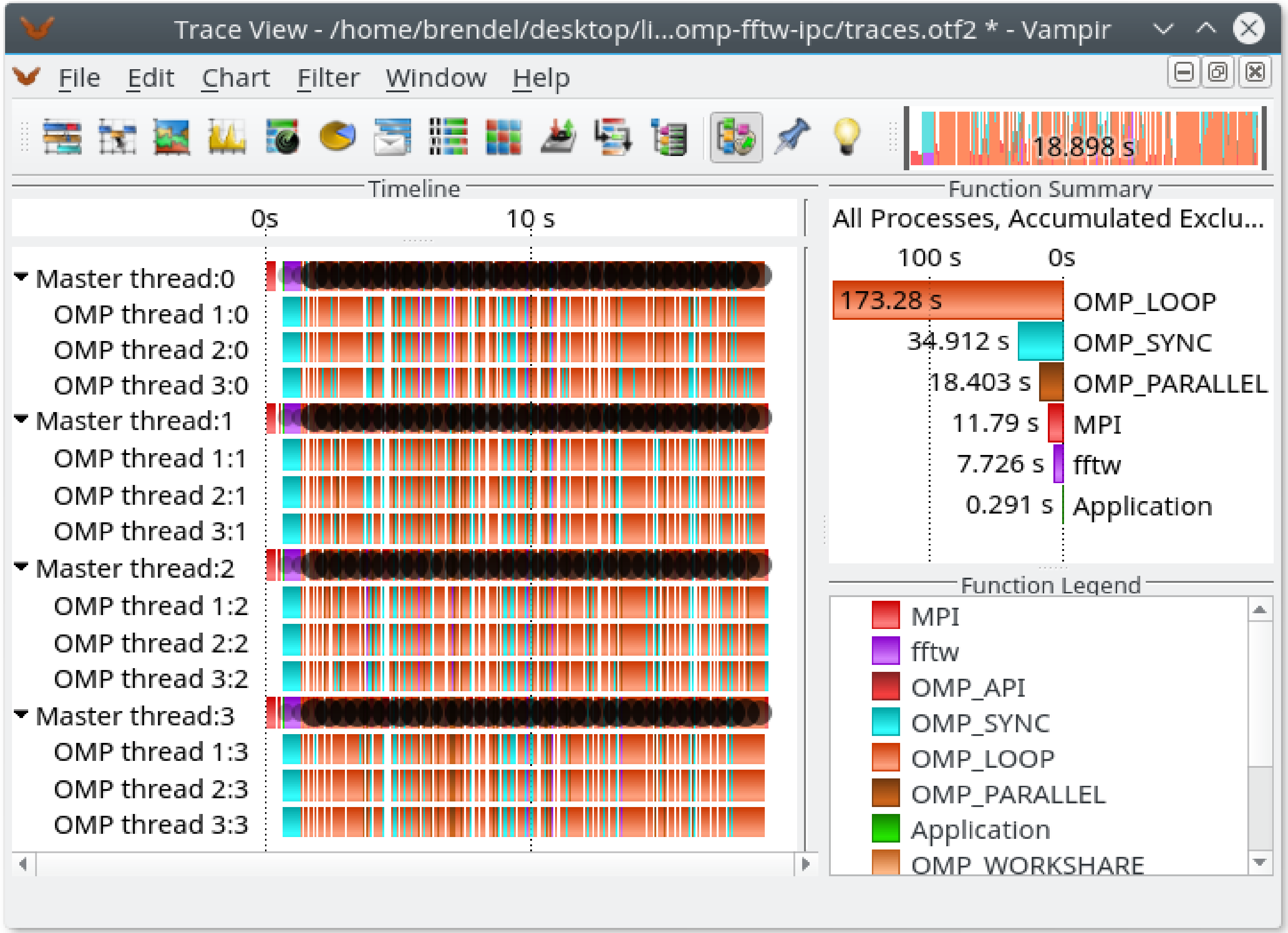
```
% scorep-libwrap-init         \
>   --name=fftw               \
>   --prefix=$PREFIX          \
>   -x c                      \
>   --cppflags="-O3 -DNDEBUG -openmp -I$FFTW_INC" \
>   --ldflags="-L$FFTW_LIB"   \
>   --libs="-lfftw3f -lfftw3" \
>   working_directory
```

- 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



Trace View - /home/brendel/desktop/li...omp-fftw-ipc/traces.otf2 * - Vampir

File Edit Chart Filter Window Help

Function Summary

All Processes, Accumulated Exclu...

Time	Function
173.28 s	OMP_LOOP
34.912 s	OMP_SYNC
18.403 s	OMP_PARALLEL
11.79 s	MPI
7.726 s	fftw
0.291 s	Application

Function Legend

- 🟥 MPI
- 🟪 fftw
- 🟥 OMP_API
- 🟦 OMP_SYNC
- 🟧 OMP_LOOP
- 🟫 OMP_PARALLEL
- 🟩 Application
- 🟧 OMP_WORKSHARE

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
- Available under 3-clause BSD open-source license
- Download sources, subscribe to news mailing list:
  - http://www.score-p.org
- User guide part of installation or available online:
  - <prefix>/share/doc/scorep/{pdf,html}/
  - Online HTML / Online PDF
- Support and feedback: support@score-p.org