

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

---

VI-HPS Team



# Congratulations!?

---

- If you made it this far, you successfully used Score-P to
  - instrument the application
  - analyze its execution with a summary measurement, and
  - 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

---

- 0.0 Reference preparation for validation
  
- 1.0 Program instrumentation
  - 1.1 Summary measurement collection
  - 1.2 Summary analysis report examination
  
- 2.0 Summary experiment scoring
  - 2.1 Summary measurement collection with filtering
  - 2.2 Filtered summary analysis report examination
  
- 3.0 Event trace collection
  - 3.1 Event trace examination & analysis

## BT-MZ summary analysis result scoring

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

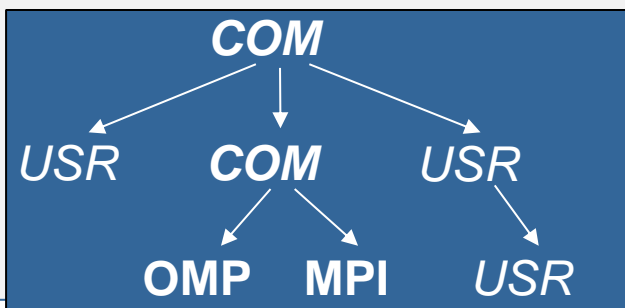
Estimated aggregate size of event trace:

Estimated requirements for largest trace buffer (max\_buf):

Estimated memory requirements (SCOREP\_TOTAL\_MEMORY):

(hint: When tracing set SCOREP\_TOTAL\_MEMORY=2373MB 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	2,479,514,724	1,634,202,275	11031.37	100.0	6.75	ALL
	USR	2,477,923,488	1,631,143,401	4383.44	39.7	2.69	USR
	OMP	4,129,716	2,743,808	4895.09	44.4	1784.05	OMP
	MPI	372,431	128,436	1738.54	15.8	13536.22	MPI
	COM	225,290	186,630	14.30	0.1	76.61	COM



- Report scoring as textual output

40 GB total memory  
2.3 GB per rank!

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

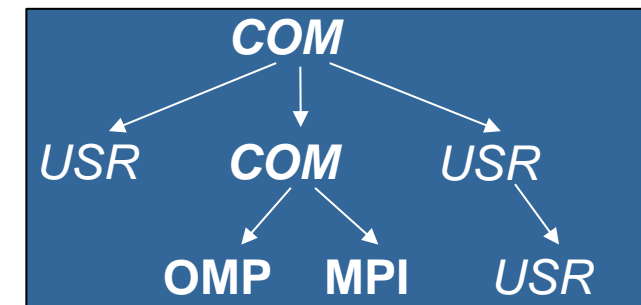
# BT-MZ summary analysis report breakdown

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

```
[...]
[...]
```

flt type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
ALL	2,479,514,724	1,634,202,275	9402.51	100.0	5.75	ALL
USR	2,477,923,448	1,631,143,401	3694.84	39.3	2.27	USR
OMP	4,129,716	2,743,808	4200.62	44.7	1530.94	OMP
MPI	372,430	128,436	1494.89	15.9	11639.21	MPI
COM	225,290	186,630	12.16	0.1	65.15	COM

USR	800,074,470	522,844,416	924.44	9.8	1.77	matvec_sub_
USR	800,074,470	522,844,416	1593.32	16.9	3.05	binvrhs_
USR	800,074,470	522,844,416	1030.22	11.0	1.97	matmul_sub_
USR	26,365,170	22,692,096	60.65	0.6	2.67	lhsinit_
USR	26,365,170	22,692,096	55.60	0.6	2.45	binvrhs_
USR	24,964,368	17,219,840	30.58	0.3	1.78	exact_solution_



More than  
2.2 GB just for these 6  
regions

## BT-MZ summary analysis score

---

- Summary measurement analysis score reveals
  - Total size of event trace would be ~40 GB
  - Maximum trace buffer size would be ~2.3 GB per rank
    - smaller buffer would require (unsynchronized) flushes to disk during measurement resulting in substantial perturbation
  - 99.8% of the trace requirements are for USR regions
    - purely computational routines never found on COM call-paths common to communication routines or OpenMP parallel regions
  - These USR regions contribute around 39% of total time
    - however, much of that is very likely to be measurement overhead for frequently-executed small routines
- Advisable to tune measurement configuration
  - Specify an adequate trace buffer size
  - Specify a filter file listing (USR) regions not to be measured

## BT-MZ summary analysis report filtering

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
binvrhs*
matmul_sub*
matvec_sub*
exact_solution*
binvrhs*
lhs*init*
timer_*

% scorep-score -f ../config/scorep.filt -c 2 \
  scorep_bt-mz_sum/profile.cubex

Estimated aggregate size of event trace: 242 MB
Estimated requirements for largest trace buffer (max_buf): 12 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 20 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=20MB to avoid \
>intermediate flushes
or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

242 MB of memory in total,  
20 MB per rank!

(Including 2 metric values)

# BT-MZ summary analysis report filtering

```
% scorep-score -r -f ../config/scorep.filt scorep_bt-mz_sum/profile.cubex
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/ visit[us]	region
-	ALL	2,479,514,724	1,634,202,275	9402.51	100.0	5.75	ALL
-	USR	2,477,923,448	1,631,143,401	3694.84	39.3	2.27	USR
-	OMP	4,129,716	2,743,808	4200.62	44.7	1530.94	OMP
-	MPI	372,430	128,436	1494.89	15.9	11639.21	MPI
-	COM	225,290	186,630	12.16	0.1	65.15	COM
*	ALL	4,732,090	3,064,245	5707.70	60.7	1862.68	ALL-FLT
+	FLT	2,477,918,768	1,631,138,030	3694.81	39.3	2.27	FLT
-	OMP	4,129,716	2,743,808	4200.62	44.7	1530.94	OMP-FLT
-	MPI	372,430	128,436	1494.89	15.9	11639.21	MPI-FLT
*	COM	225,290	186,630	12.16	0.1	65.15	COM-FLT
*	USR	4,680	5,371	0.03	0.0	5.59	USR-FLT
+	USR	800,074,470	522,844,416	924.44	9.8	1.77	matvec_sub_
+	USR	800,074,470	522,844,416	1593.32	16.9	3.05	binvrhs_
+	USR	800,074,470	522,844,416	1030.22	11.0	1.97	matmul_sub_
+	USR	26,365,170	22,692,096	60.65	0.6	2.67	lhsinit_
+	USR	26,365,170	22,692,096	55.60	0.6	2.45	binvrhs_
+	USR	24,964,368	17,219,840	30.58	0.3	1.78	exact_solution_

- Score report breakdown by region

Filtered routines marked with '+'





# Score-P filtering

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN
EXCLUDE
  binvcrhs*
  matmul_sub*
  matvec_sub*
  exact_solution*
  binvrhs*
  lhs*init*
  timer_*
SCOREP_REGION_NAMES_END

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

Region name  
filter block  
using wildcards

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

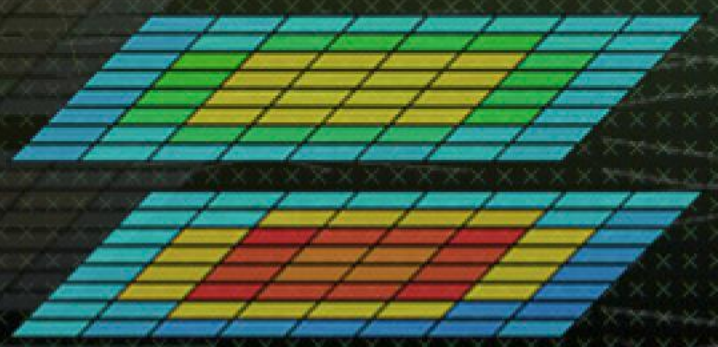


- Name mangling
  - Filtering based on names seen by the measurement system
    - Dependent on compiler
    - Actual name may be mangled
- `scorep-score` names as starting point  
(e.g. `matvec_sub_`)
  - Use `*` for Fortran trailing underscore(s) for portability
  - Use `?` and `*` as needed for full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



## Score-P: Advanced Measurement Configuration



# Mastering build systems



- Hooking up the Score-P instrumenter `scorep` into complex build environments like *Autotools* or *CMake* was always challenging
- Score-P provides new 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 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

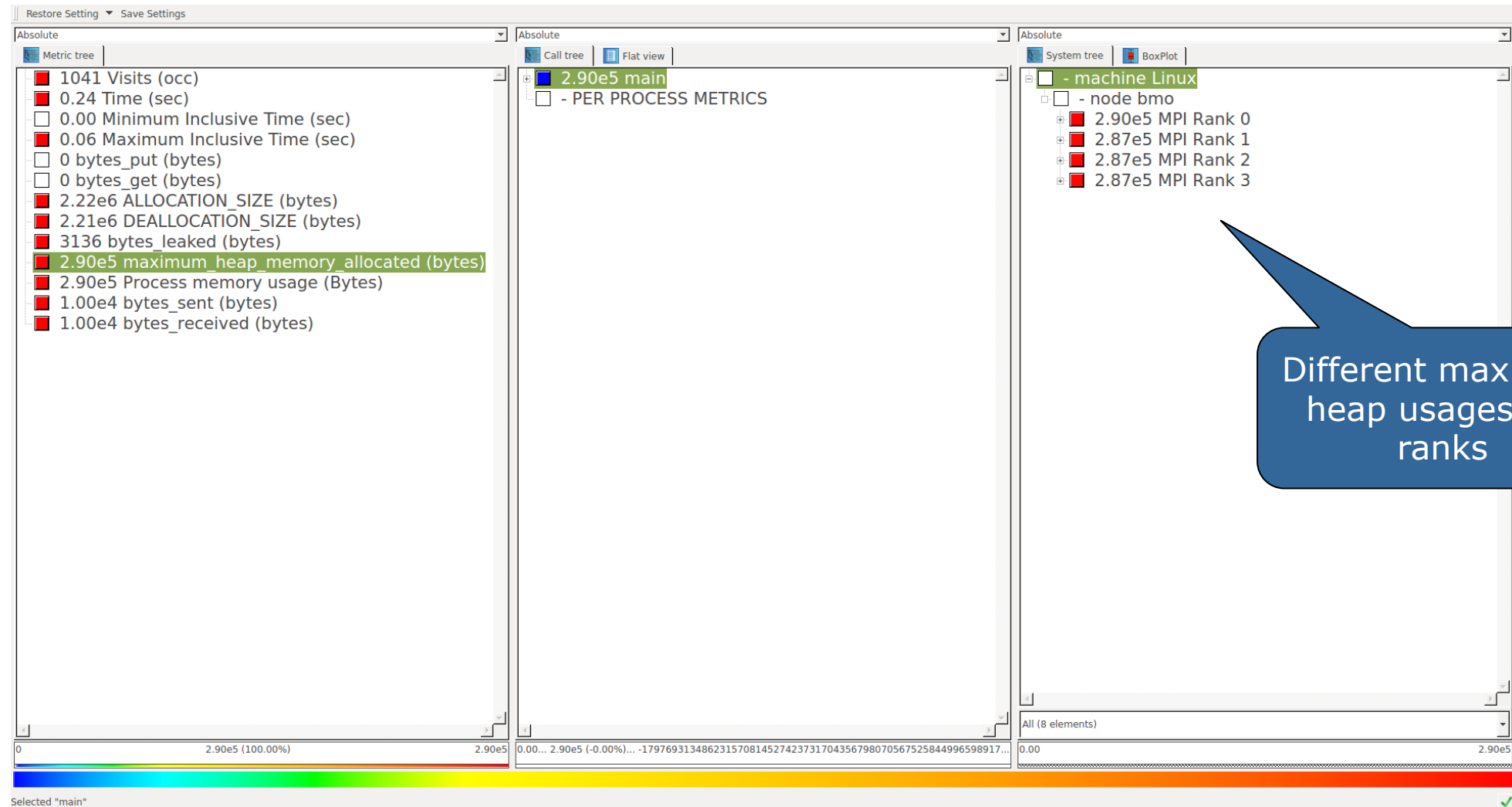
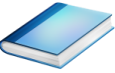
```
% export SCOREP_MEMORY_RECORDING=true  
% export SCOREP_MPI_MEMORY_RECORDING=true  
  
% OMP_NUM_THREADS=4 mpiexec -np 4 ./bt-mz_W.4
```

- Set new configuration variable to enable memory recording

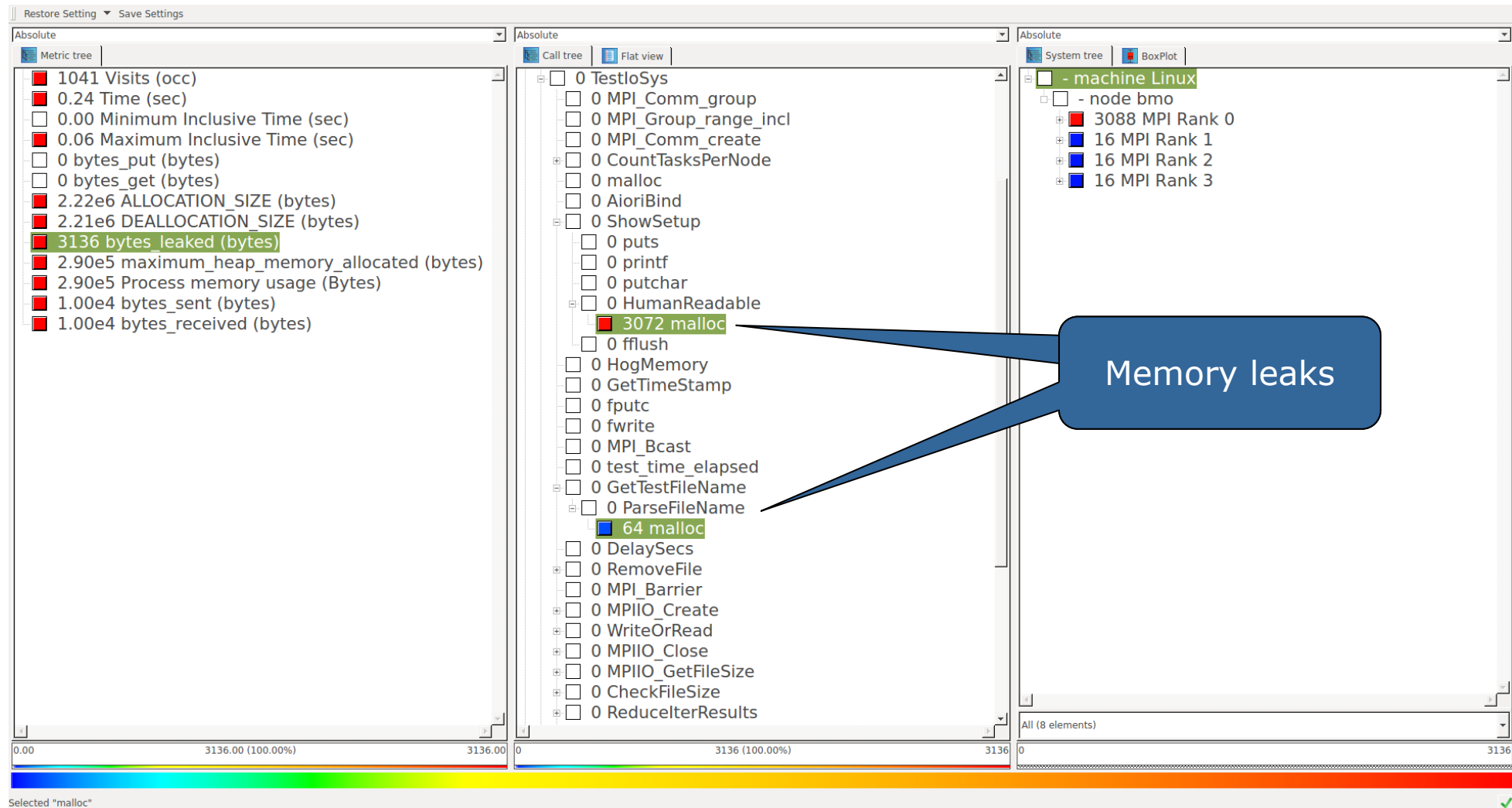
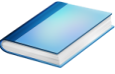
- Available since Score-P 2.0



# Mastering application memory usage



# Mastering application memory usage



# Advanced measurement configuration: Metrics



- `SCOREP_METRIC_PAPI=PAPI_TOT_CYC,PAPI_TOT_INS`
- Available PAPI metrics
  - Preset events: common events deemed relevant and useful for application performance tuning
    - Abstraction from specific hardware performance counters, mapping onto available events done by PAPI internally

```
% papi_avail
```

- Native events: set of all events that are available on the CPU  
(platform dependent)

```
% papi_native_avail
```

## Note:

Due to hardware restrictions

- number of concurrently recorded events is limited
- there may be invalid combinations of concurrently recorded events

# Advanced measurement configuration: Metrics

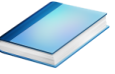


```
% man getrusage
struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    long ru_maxrss; /* maximum resident set size */
    long ru_ixrss; /* integral shared memory size */
    long ru_idrss; /* integral unshared data size */
    long ru_isrss; /* integral unshared stack size */
    long ru_minflt; /* page reclaims (soft page faults) */
    long ru_majflt; /* page faults (hard page faults) */
    long ru_nswap; /* swaps */
    long ru_inblock; /* block input operations */
    long ru_oublock; /* block output operations */
    long ru_msgsnd; /* IPC messages sent */
    long ru_msrvcv; /* IPC messages received */
    long ru_nsignals; /* signals received */
    long ru_nvcsw; /* voluntary context switches */
    long ru_nivcsw; /* involuntary context switches */
};
```

- **SCOREP\_METRIC\_RUSAGE**  
**=ru\_stime,ru\_utime**
  - ("all" for complete set)
- Available resource usage metrics
- **Note:**
  - (1) Not all fields are maintained on each platform.
  - (2) Check scope of metrics (per process vs. per thread)

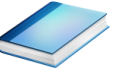
# Score-P user instrumentation API

---



- Can be used to mark initialization, solver & other phases
  - Annotation macros ignored by default
  - Enabled with [--user] flag
- Appear as additional regions in analyses
  - Distinguishes performance of important phase from rest
- Can be of various type
  - E.g., function, loop, phase
  - See user manual for details
- Available for Fortran / C / C++

# Score-P user instrumentation API (Fortran)



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

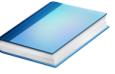
subroutine foo(...)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! 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

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

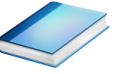


```
#include "scorep/SCOREP_User.h"

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++

## Further information

---

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- 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,html}/`
- Support and feedback: [support@score-p.org](mailto:support@score-p.org)
- Subscribe to [news@score-p.org](mailto:news@score-p.org), to be kept informed