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

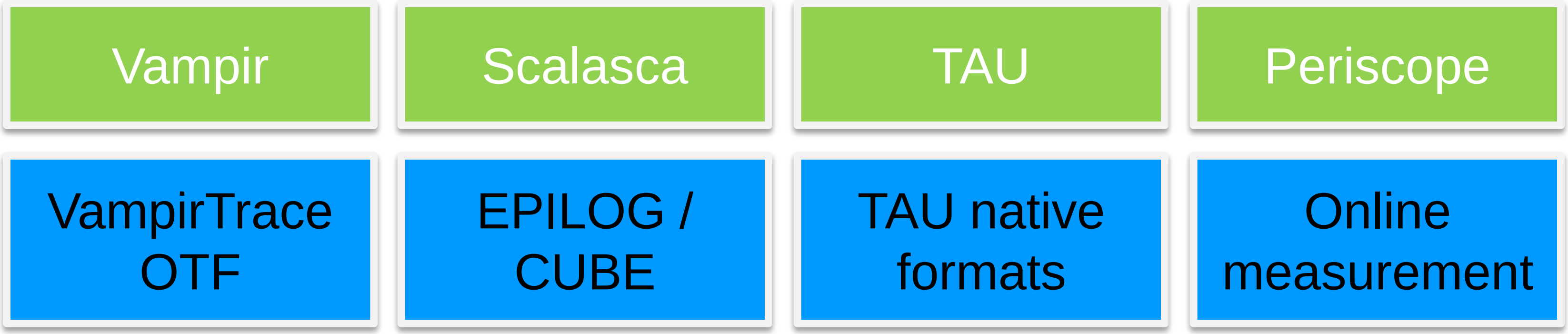
Markus Geimer[1)], Bert Wesarg[2)]

With contributions from
Andreas Knüpfer[2)] and Christian Rössel[1)]

[1)]FZ Jülich, [2)]ZIH TU Dresden

Fragmentation of tools landscape

- Several performance tools co-exist
- With own measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training



Vampir	Scalasca	TAU	Periscope
VampirTrace OTF	EPILOG / CUBE	TAU native formats	Online measurement

SILC Project Idea

- Start a community effort for a common infrastructure
  - Score-P instrumentation and measurement system
  - Common data formats OTF2 and CUBE4
- Developer perspective:
  - Save manpower by sharing development resources
  - Invest in new analysis functionality and scalability
  - Save efforts for maintenance, testing, porting, support, training
- User perspective:
  - Single learning curve
  - Single installation, fewer version updates
  - Interoperability and data exchange
- SILC project funded by BMBF
- Close collaboration PRIMA project funded by DOE

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Partners

- Forschungszentrum Jülich, Germany
- German Research School for Simulation Sciences, Aachen, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

Score-P Functionality

- Provide typical functionality for HPC performance tools
- Support all fundamental concepts of partner‘s tools

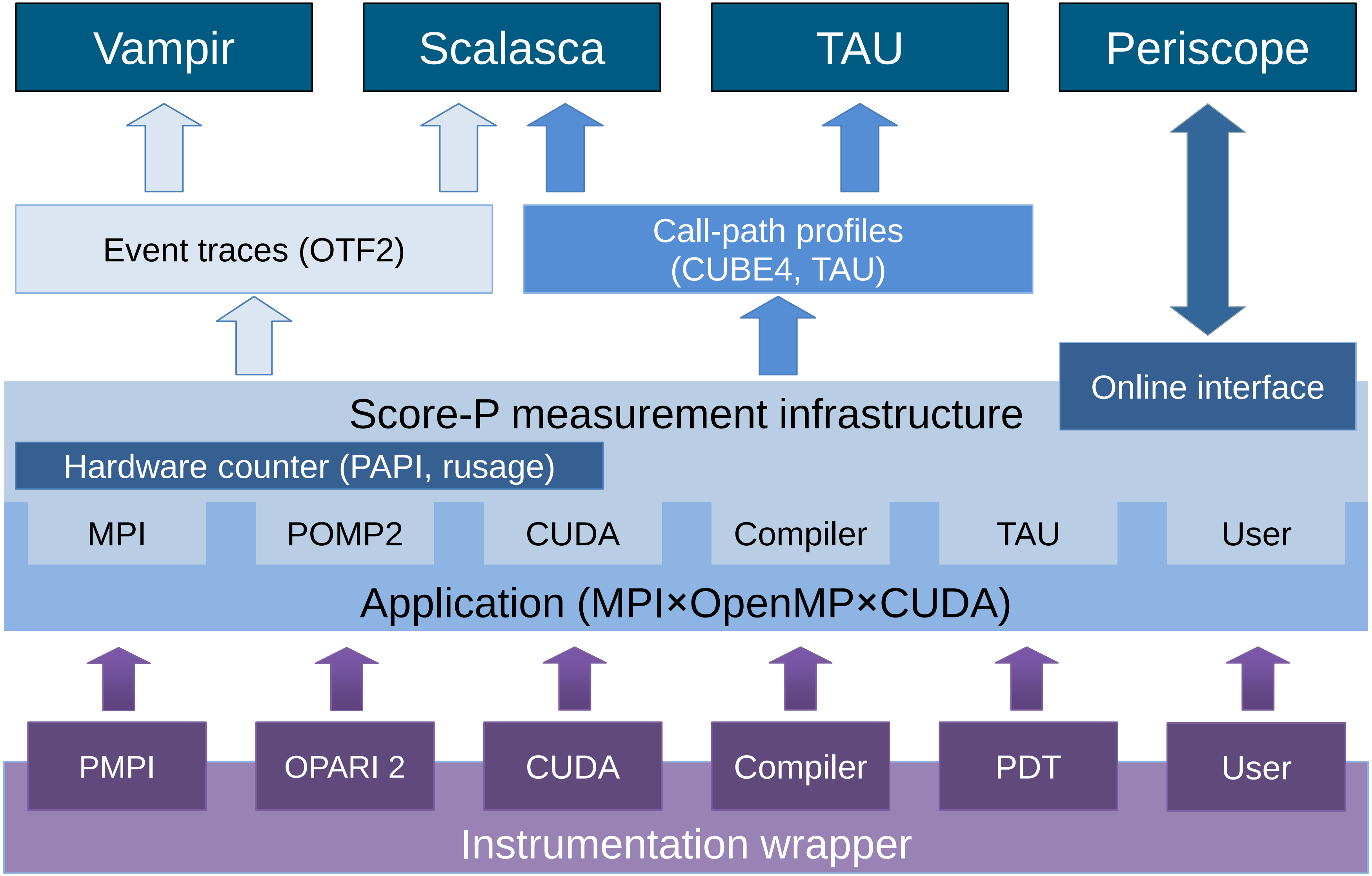
- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI, OpenMP, and hybrid parallelism (and serial)
- Enhanced functionality (OpenMP 3.0, CUDA, highly scalable I/O)

Design Goals

- Functional requirements
  - Generation of call-path profiles and event traces
  - Using direct instrumentation, later also sampling
  - Recording time, visits, communication data, hardware counters
  - Access and reconfiguration also at runtime
  - Support for MPI, OpenMP, basic CUDA, and all combinations
    - Later also OpenCL/HMPP/PTHREAD/…
- Non-functional requirements
  - Portability: all major HPC platforms
  - Scalability: petascale
  - Low measurement overhead
  - Easy and uniform installation through UNITE framework
  - Robustness
  - Open Source: New BSD License

Score-P Architecture



Future Features and Management

- Scalability to maximum available CPU core count
- Support for OpenCL, HMPP, PTHREAD
- Support for sampling, binary instrumentation
- Support for new programming models, e.g., PGAS
- Support for new architectures

- Ensure a single official release version at every time which will always work with the tools
- Allow experimental versions for new features or research

- Commitment to joint long-term cooperation

Score-P hands-on:
NPB-MZ-MPI / BT

Performance analysis steps

1. Reference preparation for validation
2. Program instrumentation
3. Summary measurement collection
4. Summary analysis report examination
5. Summary experiment scoring
6. Summary measurement collection with filtering
7. Filtered summary analysis report examination
8. Event trace collection
9. Event trace examination & analysis

NPB-MZ-MPI / BT instrumentation

- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: MPIF77

```
#            SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#---------------------------------------------------------------------
# Items in this file may need to be changed for each platform.
#---------------------------------------------------------------------
...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------
#MPIF77 = mpif77

# Alternative variants to perform instrumentation
...
MPIF77 = scorep --user mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK   = $(MPIF77)
...
```

Uncomment the Score-P compiler wrapper specification

NPB-MZ-MPI / BT instrumented build

- Return to root directory and clean-up

```
% make clean
```

- Re-build executable using Score-P compiler wrapper

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 W
scorep --user mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  scorep --user mpif77 -c  -O3 -fopenmp timers.f
scorep --user mpif77 -O3 -fopenmp -o ../bin.scorep/bt-mz_W.4 \
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \
solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \
../common/print_results.o ../common/timers.o
Built executable ../bin.scorep/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

Measurement configuration: scorep-info

- Score-P measurements are configured via environmental variables:

```
% scorep-info config-vars
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 ...]
```

Summary measurement collection

- Change to the directory containing the new executable before running it with the desired configuration

```
% cd bin.scorep
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
% export OMP_NUM_THREADS=4
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:   4 x   4
 Iterations: 200    dt:   0.000800
 Number of active processes:     4

 Use the default load factors with threads
 Total number of threads:     16  (  4.0 threads/process)

 Calculated speedup =     15.78

 Time step    1

 [... More application output ...]
```

BT-MZ summary analysis report examination

- Creates experiment directory ./scorep_bt-mz_W_4x4_sum containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```
% ls
bt-mz_W.4  scorep_bt-mz_W_4x4_sum
% ls scorep_bt-mz_W_4x4_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with CUBE / ParaProf

```
% cube scorep_bt-mz_W_4x4_sum/profile.cubex

              [CUBE GUI showing summary analysis report]

% paraprof scorep_bt-mz_W_4x4_sum/profile.cubex

          [TAU ParaProf GUI showing summary analysis report]
```

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

BT-MZ summary analysis result scoring

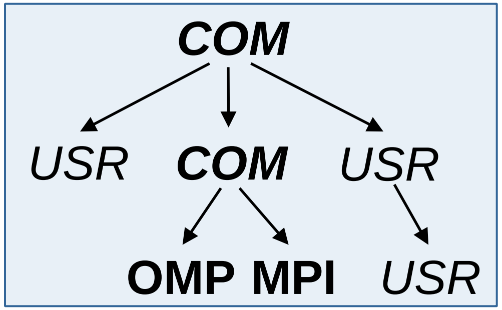
- Report scoring as textual output

```
% scorep-score scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace (total_tbc):       990247448 bytes
Estimated requirements for largest trace buffer (max_tbc): 256229936 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)

flt type         max_tbc         time      % region
     ALL       256229936      5549.78  100.0 ALL
     USR       253654608      1758.27   31.7 USR
     OMP         5853120      3508.57   63.2 OMP
     COM          343344       183.09    3.3 COM
     MPI           93776        99.86    1.8 MPI
```

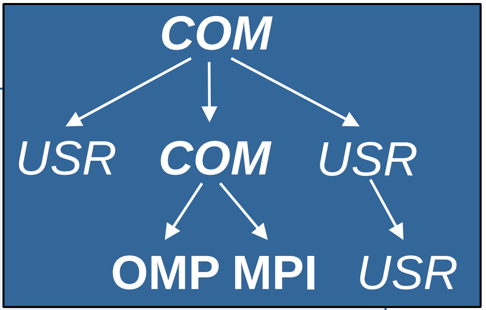
1GB of memory in total, 256 MB per rank!

- Region/callpath classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation)
  - COM (“combined” USR + OpenMP/MPI)
  - ANY/ALL (aggregate of all region types)



BT-MZ summary analysis report breakdown

- Score report breakdown by region



```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex
  [...]
flt  type         max_tbc         time      % region
     ALL        256229936      5549.78  100.0 ALL
     USR        253654608      1758.27   31.7 USR
     OMP          5853120      3508.57   63.2 OMP
     COM           343344       183.09    3.3 COM
                    93776        99.86    1.8 MPI

                 79176312       559.15   31.8 binvcrhs_
                 79176312       532.73   30.3 matvec_sub_
     USR         79176312       532.18   30.3 matmul_sub_
     USR          7361424        50.51    2.9 binvrhs_
     USR          7361424        56.35    3.2 lhsinit_
     USR          3206688        27.32    1.6 exact_solution_
     OMP          1550400      1752.20   99.7 !$omp implicit barrier
     OMP           257280         0.44    0.0 !$omp parallel @exch_qbc.f
     OMP           257280         0.61    0.0 !$omp parallel @exch_qbc.f
     OMP           257280         0.48    0.0 !$omp parallel @exch_qbc.f
 [...]
```

More than 250MB just for these 6 regions

BT-MZ summary analysis score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~990MB
  - Maximum trace buffer size would be ~256MB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 98.9% 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 32% of total time
    - however, much of that is very likely to be measurement overhead for frequently-executed small routines (and due to oversubscription)
- 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

- Report scoring with prospective filter listing 6 USR regions

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

% scorep-score -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum
Estimated aggregate size of event trace (total_tbc):        20210360 bytes
Estimated requirements for largest trace buffer (max_tbc): 6290888 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)
```

20MB of memory in total, 6 MB per rank!

BT-MZ summary analysis report filtering

- Score report breakdown by region

```
% scorep-score -r -f ../config/scorep.filt \
> scorep_bt-mz_W_4x4_sum/profile.cubex
flt  type       max_tbc       time     % region
 +   FLT      253653936    1758.26  31.7 FLT
 *   ALL        6290888    3791.53  68.3 ALL-FLT
 -   OMP        5853120    3508.57  63.2 OMP-FLT
 *   COM         343344     183.09   3.3 COM-FLT
 -   MPI          93776      99.86   1.8 MPI-FLT
 *   USR            672       0.01   0.0 USR-FLT

 +   USR       79176312     559.15  31.8 binvcrhs_
 +   USR       79176312     532.73  30.3 matvec_sub_
 +   USR       79176312     532.18  30.3 matmul_sub_
 +   USR        7361424      50.51   2.9 binvrhs_
 +   USR        7361424      56.35   3.2 lhsinit_
 +   USR        3206688      27.32   1.6 exact_solution_
 -   OMP        1550400    1752.20  99.7 !$omp implicit barrier
 -   OMP         257280       0.44   0.0 !$omp parallel @exch_qbc.f
 -   OMP         257280       0.61   0.0 !$omp parallel @exch_qbc.f
 -   OMP         257280       0.48   0.0 !$omp parallel @exch_qbc.f
  [...]
```

Filtered routines marked with '+'

BT-MZ filtered summary measurement

- Set new experiment directory and re-run measurement with new filter configuration

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum_with_filter
% export SCOREP_FILTERING_FILE=../config/scorep.filt
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:   4 x   4
 Iterations: 200    dt:   0.000800
 Number of active processes:     4

 Use the default load factors with threads
 Total number of threads:     16  (  4.0 threads/process)

 Calculated speedup =     15.78

 Time step    1

 [... More application output ...]
```

BT-MZ tuned summary analysis report score

- Scoring of new analysis report as textual output

```
% scorep-score scorep_bt-mz_W_4x4_sum_with_filter/profile.cubex
Estimated aggregate size of event trace (total_tbc):       20210360 bytes
Estimated requirements for largest trace buffer (max_tbc): 6290888 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)

flt type         max_tbc         time      % region
     ALL         6290888       241.77  100.0 ALL
     OMP         5853120       168.94   69.9 OMP
     COM          343344        35.57   14.7 COM
     MPI           93776        37.25   15.4 MPI
     USR             672         0.01    0.0 USR
```

- Significant reduction in runtime (measurement overhead)
  - Not only reduced time for USR regions, but MPI/OMP reduced too!
- Further measurement tuning (filtering) may be appropriate
  - e.g., use “timer_*” to filter timer_start_, timer_read_, etc.

BT-MZ trace measurement collection...

- Re-run the application using the tracing mode of Score-P

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_ENABLE_PROFILING=false
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Separate trace file per thread written straight into new experiment directory ./scorep_bt-mz_W_4x4_trace
- Interactive trace exploration with Vampir

```
% vampir scorep_bt-mz_W_4x4_trace/traces.otf2

                              [Vampir GUI showing trace]
```

Warnings and tips regarding tracing

- Traces can become extremely large and unwieldy
  - Size is proportional to number of processes/threads (width), duration (length) and detail (depth) of measurement
- Traces containing intermediate flushes are of little value
  - Uncoordinated flushes result in cascades of distortion
  - Reduce size of trace such that it fits in available buffer space
- Traces should be written to a parallel file system
  - /work or /scratch are typically provided for this purpose
- Moving large traces between file systems is often impractical
  - However, systems with more memory can analyze larger traces
  - Alternatively, run trace analyzers with undersubscribed nodes

Advanced measurement configuration: metrics

- Recording hardware counters via PAPI

```
% export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Also possible to record them only per rank

```
% export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_DCM
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Recording operating system resource usage

```
% export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss,ru_stime
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

Advanced measurement configuration: MPI

- Record only for subset of the MPI functions events

```
% export SCOREP_MPI_ENABLE_GROUPS=cg,coll,p2p,xnonblock
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 [... More application output ...]
```

- All possible sub-groups

– cg	Communicator and group management
– coll	Collective functions
– env	Environmental management
– err	MPI Error handling
– ext	External interface functions
– io	MPI file I/O
– misc	Miscellaneous
– perf	PControl
– p2p	Peer-to-peer communication
– rma	One sided communication
– spawn	Process management
– topo	Topology
– type	MPI datatype functions
– xnonblock	Extended non-blocking events
– xreqtest	Test events for uncompleted requests

Advanced measurement configuration: CUDA

- Record CUDA events with the CUPTI interface

```
% export SCOREP_CUDA_ENABLE=gpu,kernel,idle
% mpiexec -np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- All possible recording types

– runtime	CUDA runtime API
– driver	CUDA driver API
– gpu	GPU activities
– kernel	CUDA kernels
– idle	GPU compute idle time
– memcpy	CUDA memory copies (not available yet)

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

Score-P

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under New BSD open-source license
- Documentation & Sources:
  - http://www.score-p.org
- User guide also part of installation:
  - <prefix>/share/doc/scorep/{pdf,html}/
- Contact: info@score-p.org
- Bugs: scorep-bugs@groups.tu-dresden.de