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

Robert Dietrich[1)], Tobias Hilbrich[1)],
Marc Schlütter[2)]

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

[1)]ZIH TU Dresden , [2)]FZ Jülich

Fragmentation of Tools Landscape

- Several performance tools co-exist
- Separate 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

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

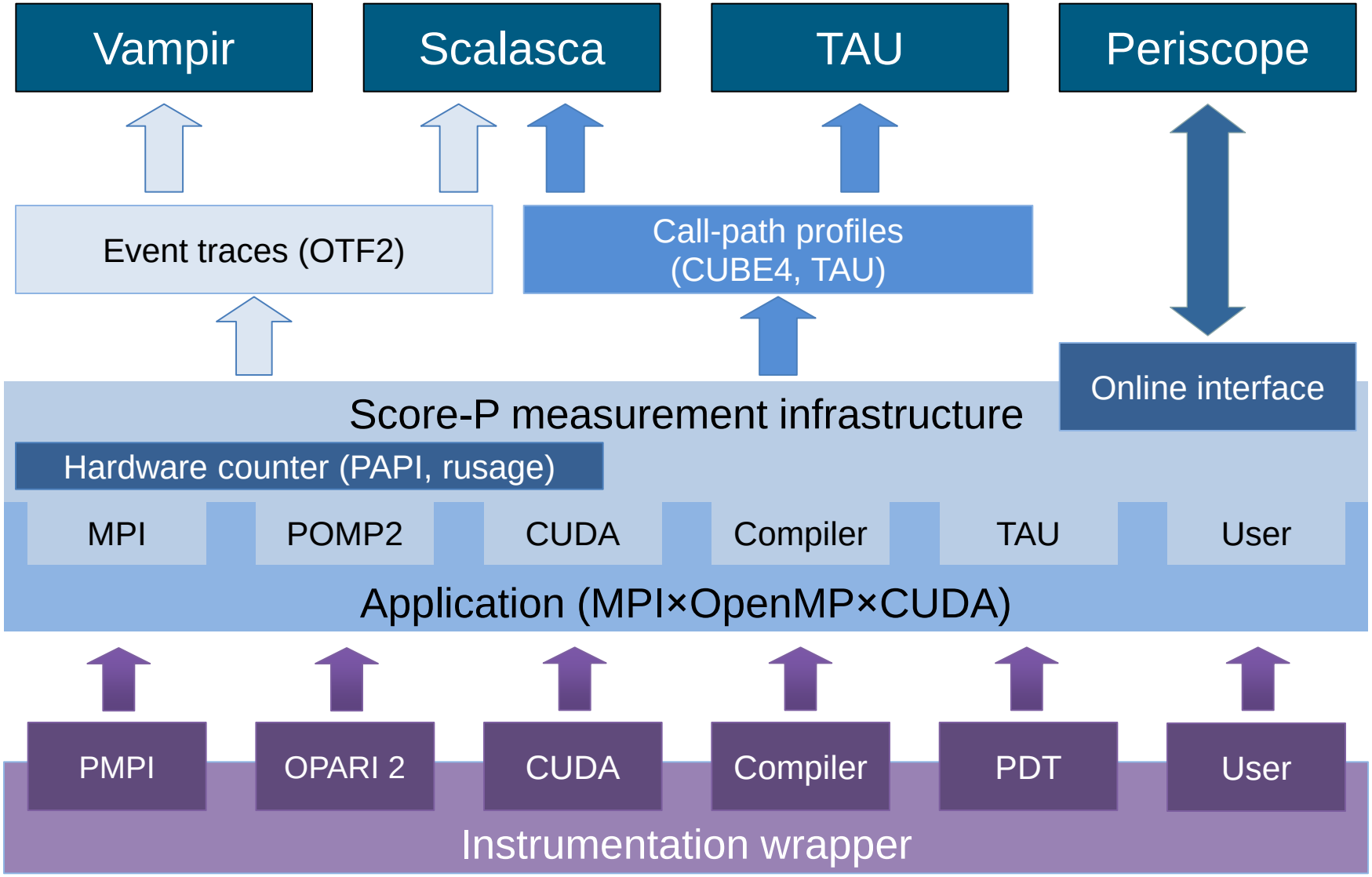
- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- 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, 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 all times 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 (cont.)

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

- Load modules

```
% module load UNITE
UNITE loaded
% module load scorep
Scorep/1.2.3-beta-intel-openmpi loaded

% module load cube4
cube4/4.2.1 loaded

% module load papi
papi/5.3.0 loaded
```

- Change to source directory of NPB BT-MZ

- 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 -fpp

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

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

Uncomment the Score-P compiler wrapper specification

- Return to root directory and clean-up

```
% make clean
```

- Re-build executable using Score-P instrumenter

```
% make bt-mz CLASS=B NPROCS=4
cd BT-MZ; make CLASS=C NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc -o setparams setparams.c
../sys/setparams bt-mz 4 B
scorep mpif77 -c -O3 -openmp bt.f
[...]
cd ../common; scorep mpif77 -c -O3 -openmp timers.f
scorep mpif77 -O3 -openmp -o ../bin.scorep/bt-mz_B.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_B.4
make: Leaving directory 'BT-MZ'
```

Measurement Configuration: scorep-info

- Score-P measurements are configured via environment variables:

```
% 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 ...]
```

- Change to the directory containing the new executable and run it

```
% cd bin.scorep
% cp ../jobscript/marenostrum/run.scorep.lsf .
% vim run.scorep.lsf
% bsub < run.scorep.lsf
% cat .<id>
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:  8 x 8
Iterations: 200   dt:  0.000300
Number of active processes:  4
Total number of threads:     16 ( 4.0 threads/process)

Time step 1
Time step 20
[...]
Time step 180
Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 15.52
```

BT-MZ Summary Analysis Report Examination

- Creates experiment directory ./scorep_bt-mz_B_4x4_sum containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```
% ls
...  scorep_bt-mz_B_4x4_sum
% ls scorep_bt-mz_B_4x4_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with CUBE / ParaProf

```
% cube scorep_bt-mz_B_4x4_sum/profile.cubex

            [CUBE GUI showing summary analysis report]

% paraprof scorep_bt-mz_B_4x4_sum/profile.cubex

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

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

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

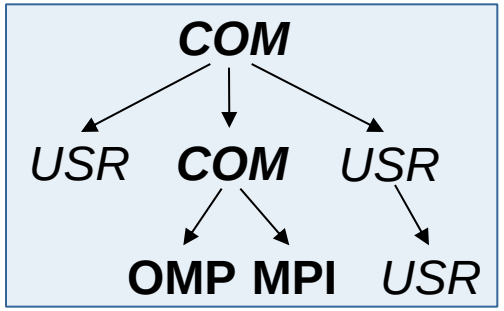
- Report scoring as textual output

```
% scorep-score scorep_bt-mz_B_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   35965836622 bytes
Estimated requirements for largest trace buffer (max_tbc):  9046029930 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      9046034330       747.05  100.0 ALL
     USR      9025830154       339.16   45.4 USR
     OMP        19113728       399.13   53.4 OMP
     COM         1001550         1.18    0.2 COM
     MPI           88898         7.59    1.0 MPI
```

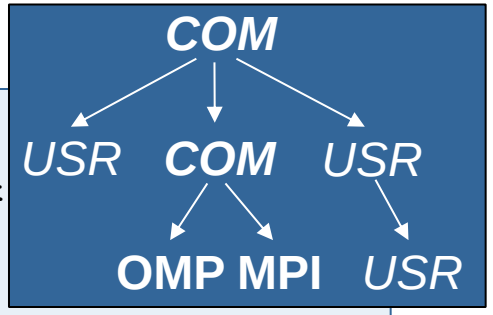
36 GB total memory
9 GB per rank!

- 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_B_4x4_sum/profile.cubex
  [...]
flt type          max_tbc         time      % region
     ALL       9046029930       766.34  100.0 ALL
     USR       9025830154       342.80   44.7 USR
     OMP         19113728       407.08   53.1 OMP
                   997150         1.46    0.2 COM
                    88898        15.00    2.0 MPI

                2894950740       138.43   18.1 binvcrhs_
                2894950740       105.80   13.8 matmul_sub_
     USR        2894950740        85.59   11.2 matvec_sub_
     USR         127716204         4.52    0.6 binvrhs_
     USR         127716204         5.75    0.8 lhsinit_
     USR          94933520         2.70    0.4 exact_solution_
     OMP           1183488         0.04    0.0 !$omp parallel @exch_...
     OMP           1183488         0.04    0.0 !$omp parallel @exch_...
     OMP           1183488         0.04    0.0 !$omp parallel @exch_...
  [...]
```

About 9 GB just for these 6 regions

BT-MZ Summary Analysis Score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~36 GB
  - Maximum trace buffer size would be ~9 GB per rank
    - smaller buffer would require 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 44% 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

- Report scoring with prospective filter listing
  7 USR regions

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

% scorep-score -f ../config/scorep.filt scorep_bt-mz_B_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   80814262 bytes
Estimated requirements for largest trace buffer (max_tbc): 20203582 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.)
```

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

BT-MZ Summary Analysis Report Filtering

- Score report breakdown by region

```
% scorep-score -r -f ../config/scorep.filt \
> scorep_bt-mz_B_4x4_sum/profile.cubex
[...]
 *    ALL        20204132       423.54   55.3 ALL-FLT
 +    FLT      9025825820       342.80   44.7 FLT
 -    OMP        19113728       407.08   53.1 OMP-FLT
 *    COM          997150         1.46    0.2 COM-FLT
 -    MPI           88898        15.00    2.0 MPI-FLT
 *    USR            4356         0.00    0.0 USR-FLT

 +    USR      2894950740       138.43   18.1 binvcrhs_
 +    USR      2894950740       105.80   13.8 matmul_sub_
 +    USR      2894950740        85.59   11.2 matvec_sub_
 +    USR       127716204         4.52    0.6 binvrhs_
 +    USR       127716204         5.75    0.8 lhsinit_
 +    USR        94933520         2.70    0.4 exact_solution_
[...]
```

Filtered routines marked with '+'

BT-MZ Filtered Summary Measurement

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

```
% vim run.scorep.lsf
```

  - Adjust configuration

```
...
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_B_4x4_sum_with_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
...
```

  - Submit job

```
% bsub < run.scorep.lsf
```

BT-MZ Tuned Summary Analysis Report Score

- Scoring of new analysis report as textual output

```
% scorep-score scorep_bt-mz_B_4x4_sum_with_filter/profile.cubex
Estimated aggregate size of event trace:                   80814262 bytes
Estimated requirements for largest trace buffer (max_tbc): 20203582 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        20203582       250.77  100.0 ALL
     OMP        19113728       243.35   97.0 OMP
     COM          997150         1.29    0.5 COM
     MPI           88898         6.13    2.4 MPI
     USR            3806         0.00    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.

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

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
  - Increase 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

Measurement Configuration: scorep-info

- Score-P measurements are configured via environmental variables:

```
% 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 ...]
```

BT-MZ Trace Measurement Collection...

- Re-run the application using the tracing mode of Score-P
  - Edit scorep.lsf to adjust configuration

```
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_B_4x4_trace
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_ENABLE_TRACING=true
export SCOREP_ENABLE_PROFILING=false
export SCOREP_METRIC_PAPI=PAPI_FP_OPS
export SCOREP_TOTAL_MEMORY=50M
```

  - Submit job

```
% bsub < scorep.lsf
```

- Separate trace file per thread written straight into new experiment directory ./scorep_bt-mz_B_4x4_trace

Advanced Measurement Configuration: Metrics

- Recording hardware counters via PAPI

```
export SCOREP_METRIC_PAPI=PAPI_L2_TCM,PAPI_FP_OPS
```

- Also possible to record them only per rank

```
export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_TCM
```

- Recording operating system resource usage

```
export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss,ru_stime
```

Note: Additional memory is needed to store metric values. Therefore, you may have to adjust SCOREP_TOTAL_MEMORY, for example as reported using “scorep-score -c”

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

Run this on the compute nodes!

```
% 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 measured events is limited
- there may be unsupported combinations of concurrent events

Advanced Measurement Configuration: Metrics

- Available resource usage metrics

```
% man getrusage
 [... Output ...]

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_msgrcv;        /* IPC messages received */
    long   ru_nsignals;      /* signals received */
    long   ru_nvcsw;         /* voluntary context switches */
    long   ru_nivcsw;        /* involuntary context switches */
};

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

Note:
(1) Not all fields are maintained on each platform.
(2) Check scope of metrics (per process vs. per thread)

Advanced Measurement Configuration: MPI

- Record only for subset of the MPI functions events

```
export SCOREP_MPI_ENABLE_GROUPS=cg,coll,p2p,xnonblock
```

- 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=driver,kernel,memcpy
% ./cuda_program

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

- Possible recording types
  - runtime — CUDA runtime API
  - driver — CUDA driver API
  - kernel — CUDA kernels
  - kernel_counter — CUDA kernel launch parameters
  - concurrent — Concurrent kernel recording (deprecated)
  - idle — GPU compute idle time
  - pure_idle — GPU idle time (memory copies are not idle)
  - memcpy — CUDA memory copies
  - sync — show implicit CUDA device synchronization
  - gpumemusage — CUDA memory (de)allocations as counter
  - stream_reuse — Reuse destroyed/closed CUDA streams
  - device_reuse — Reuse destroyed/closed CUDA devices

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++

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: support@score-p.org