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

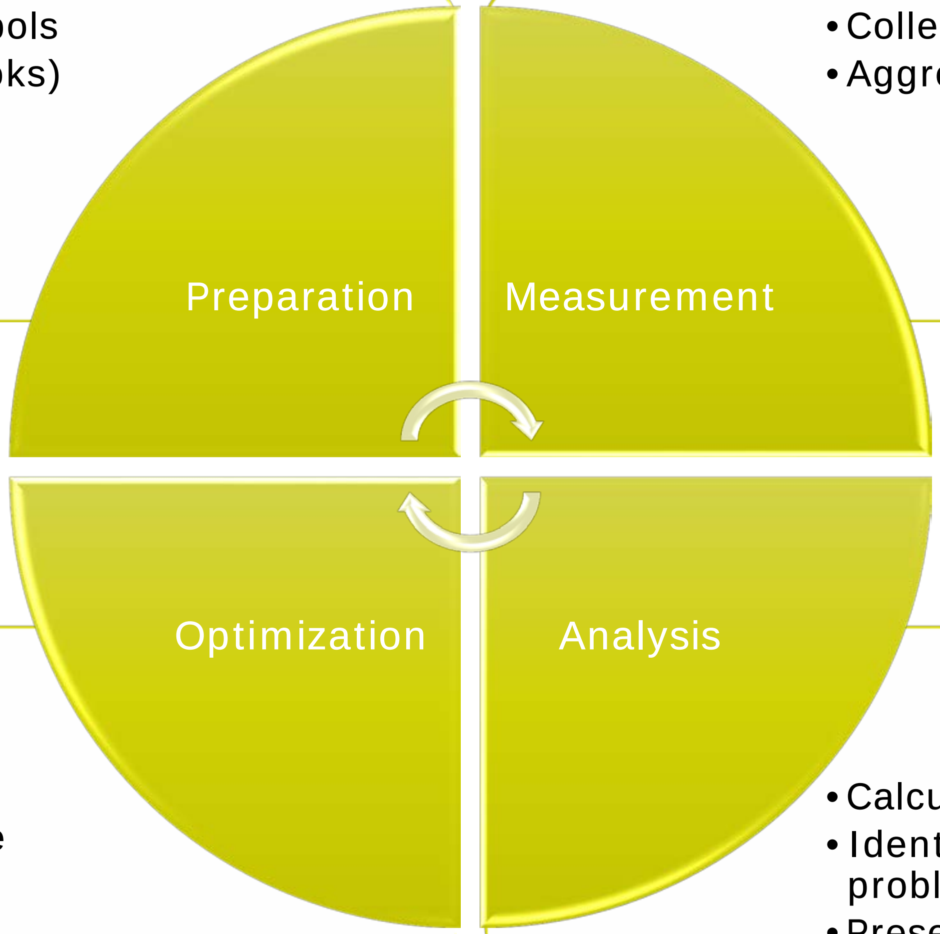
VI-HPS Team

Scalable performance measurement infrastructure for parallel codes

Performance engineering workflow

- Prepare application with symbols
- Insert extra code (probes/hooks)

- Collection of performance data
- Aggregation of performance data



- Modifications intended to eliminate/reduce performance problem

- Calculation of metrics
- Identification of performance problems
- Presentation of results

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

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

Score-P 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
- Project funded by BMBF
- Close collaboration PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

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

Score-P Functionality

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

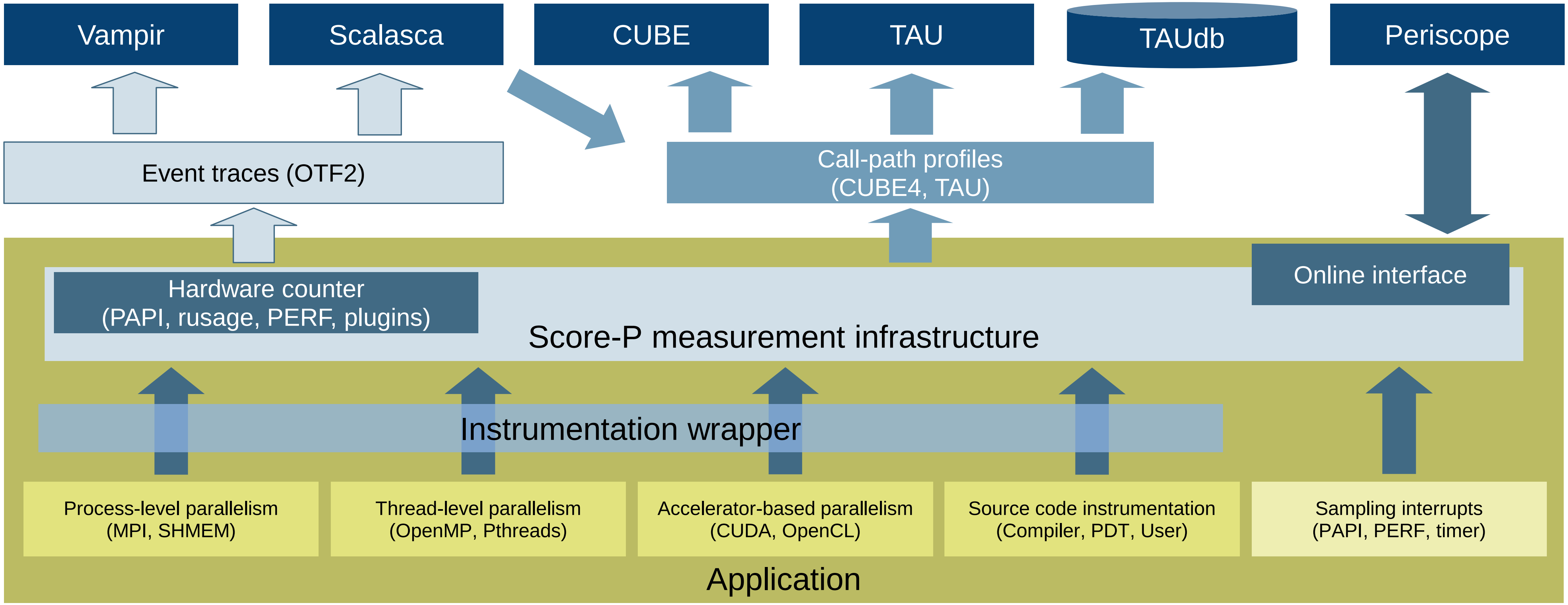
- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI/SHMEM, OpenMP/Pthreads, and hybrid parallelism (and serial)
- Enhanced functionality (CUDA, OpenCL, 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, SHMEM, OpenMP, Pthreads, CUDA, OpenCL and their valid combinations
- Non-functional requirements
  - Portability: all major HPC platforms
  - Scalability: petascale
  - Low measurement overhead
  - Robustness
  - Open Source: New BSD License

Score-P Overview



Future Features and Management

- Scalability to maximum available CPU core count
- 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

Hands-on:
NPB-MZ-MPI / BT

Score-P

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

NPB-MZ-MPI / BT Instrumentation

```
% source ~lu23bud/LRZ-VIHPSTW21/tools/source-me.scorep-2.0.1.mpt.sh
% cd $HOME/NPB3.3-MZ-MPI
```

- Setup tools environment and return to tutorial exercise source directory

NPB-MZ-MPI / BT Instrumentation

```
#             SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#---------------------------------------------------------------
# Items in this file may need to be changed for each platform.
#---------------------------------------------------------------
OPENMP = -openmp
...
#---------------------------------------------------------------
# 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)
...
```

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

Uncomment the Score-P instrumenter specification and **remove** the **–user** flag

NPB-MZ-MPI / BT Instrumented Build

```
% make clean

% make bt-mz CLASS=B NPROCS=4
cd BT-MZ; make CLASS=B NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../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‘
```

- Return to exercise directory and clean-up previous build
- Re-build executable using Score-P compiler wrapper

Summary Measurement Collection

```
% cd bin.scorep
% cp ../jobscript/lrz_uv2_mpt/scorep.mpt.sbatch .
% less scorep.mpt.sbatch
% sbatch ./scorep.mpt.sbatch
export NPB_MZ_BLOAD=0
export OMP_NUM_THREADS=4
CLASS=B
NPROCS=8
EXE=./bt-mz_$CLASS.$NPROCS

export SCOREP_TIMER=clock_gettime
export SCOREP_EXPERIMENT_DIRECTORY=scorep_4x4_sum
#export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
#export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L2_TCM
#export SCOREP_METRIC_RUSAGE=ru_stime
#export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss

srun_ps -n $NPROCS -t $OMP_NUM_THREADS $EXE

% sbatch ./scorep.mpt.sbatch
```

- Change to the directory with the new executable
- Copy the new jobscript with settings for Score-P measurement configuration
- Check/adjust settings

Leave these lines commented out for now

- Submit job

Measurement Configuration: scorep-info

```
% scorep-info config-vars --help
% 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
 [...]
 [... More configuration variables ...]
```

- Score-P measurements are configured via environmental variables
- Execute **scorep-info** for a complete list

Summary Measurement Collection

```
% less bt-mz.mpt.<jobid>.uv2.out

 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

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

Time step    1

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

- Check the output of the application run

BT-MZ Summary Analysis Report Examination

```
% ls
bt-mz_B.4  scorep.mpt.sbatch  bt-mz.mpt.<jobid>.uv2.out
scorep_4x4_sum
% ls scorep_4x4_sum
profile.cubex  scorep.cfg









% cube scorep_4x4_sum/profile.cubex

       [CUBE GUI showing summary analysis report]
```

- Creates experiment directory
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)
- Interactive exploration with CUBE

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

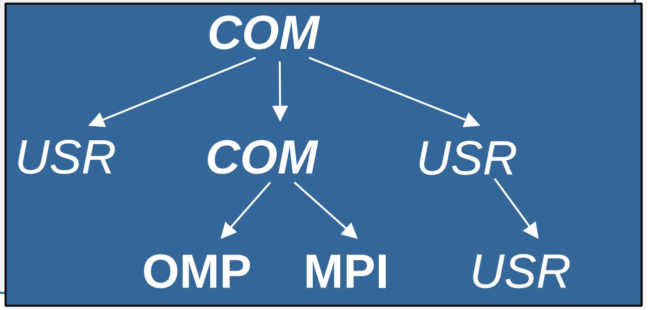
- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

BT-MZ Summary Analysis Result Scoring

```
% scorep-score scorep_4x4_sum/profile.cubex

Estimated aggregate size of event trace:                   40 GB
Estimated requirements for largest trace buffer (max_buf): 10 GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       10 GB
(warning: The memory requirements can not be satisfied by Score-P to avoid
 intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the
 maximum supported memory or reduce requirements using USR regions filters.)

flt     type      max_buf[B]         visits time[s] time[%] time/visit[us]  region
         ALL 10,690,196,070 1,634,070,493 4980.51   100.0           3.05  ALL
         USR 10,666,890,182 1,631,138,069 1676.99    33.7           1.03  USR
         OMP     22,025,152     2,743,808 2416.86    48.5         880.84  OMP
         COM      1,178,450       181,300   86.80     1.7         478.78  COM
         MPI        102,286         7,316  799.86    16.1      109330.89  MPI
```



- Report scoring as textual output

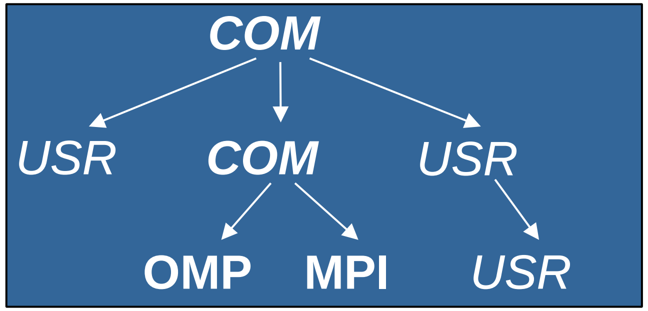
40 GB total memory
10 GB per rank!

- 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_4x4_sum/profile.cubex
  [...]
  [...]
flt   type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
       ALL 10,690,196,070 1,634,070,493 4980.51   100.0           3.05  ALL
       USR 10,666,890,182 1,631,138,069 1676.99    33.7           1.03  USR
       OMP     22,025,152     2,743,808 2416.86    48.5         880.84  OMP
       COM      1,178,450       181,300   86.80     1.7         478.78  COM
       MPI        102,286         7,316  799.86    16.1      109330.89  MPI

       USR  3,421,305,420   522,844,416  530.43    10.7           1.01  matmul_sub_
       USR  3,421,305,420   522,844,416  502.05    10.1           0.96  matvec_sub_
       USR  3,421,305,420   522,844,416  582.11    11.7           1.11  binvcrhs_
       USR    150,937,332    22,692,096   21.98     0.4           0.97  binvrhs_
       USR    150,937,332    22,692,096   24.64     0.5           1.09  lhsinit_
```



More than
9 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 ~10 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 32% 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
binvcrhs*
matmul_sub*
matvec_sub*
exact_solution*
binvrhs*
lhs*init*
timer_*


% scorep-score -f ../config/scorep.filt –c 2 \
> scorep_4x4_sum/profile.cubex

Estimated aggregate size of event trace:                   229MB
Estimated requirements for largest trace buffer (max_buf): 58MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       66MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=66MB to
> avoid intermediate flushes
 or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

521 MB of memory in total,
66 MB per rank!

(Including 2 metric values)

BT-MZ Summary Analysis Report Filtering

```
% scorep-score -r –f ../config/scorep.filt –c 2 \
> scorep_4x4_sum/profile.cubex
flt   type     max_buf[B]          visits time[s] time[%] time/visit[us]  region
 -     ALL 31,240,101,720 1,634,070,493 4980.51   100.0           3.05  ALL
 -     USR 31,180,140,532 1,631,138,069 1676.99    33.7           1.03  USR
 -     OMP     56,322,752     2,743,808 2416.86    48.5         880.84  OMP
 -     COM      3,444,700       181,300   86.80     1.7         478.78  COM
 -     MPI        193,736         7,316  799.86    16.1      109330.89  MPI

 *     ALL     59,974,336     2,933,113 3303.53    66.3        1126.29  ALL-FLT
 +     FLT 31,180,127,460 1,631,137,380 1676.99    33.7           1.03  FLT
 -     OMP     56,322,752     2,743,808 2416.86    48.5         880.84  OMP-FLT
 *     COM      3,444,700       181,300   86.80     1.7         478.78  COM-FLT
 -     MPI        193,736         7,316  799.86    16.1      109330.89  MPI-FLT
 *     USR         13,148           689    0.00     0.0           1.71  USR-FLT

 +     USR 10,000,738,920   522,844,416  530.43    10.7           1.01  matmul_sub_
 +     USR 10,000,738,920   522,844,416  502.05    10.1           0.96  matvec_sub_
 +     USR 10,000,738,920   522,844,416  582.11    11.7           1.11  binvcrhs_
 +     USR    441,201,432    22,692,096   21.98     0.4           0.97  binvrhs_
 +     USR    441,201,432    22,692,096   24.64     0.5           1.09  lhsinit_
 +     USR    327,952,160    17,219,840   15.77     0.3           0.92  exact_solution_
```

- Score report breakdown by region

Filtered routines marked with ‘+’

BT-MZ Filtered Summary Measurement

```
% cd bin.scorep
% cp ../jobscript/lrz_uv2_mpt/scorep.mpt.sbatch .
% less scorep.mpt.sbatch
% sbatch ./scorep.mpt.sbatch
export NPB_MZ_BLOAD=0
export OMP_NUM_THREADS=4
CLASS=B
NPROCS=8
EXE=./bt-mz_$CLASS.$NPROCS

export SCOREP_TIMER=clock_gettime
export SCOREP_EXPERIMENT_DIRECTORY=scorep_4x4_sum_filtered
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
#export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L2_TCM
#export SCOREP_METRIC_RUSAGE=ru_stime
#export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss

srun_ps -n $NPROCS -t $OMP_NUM_THREADS $EXE

% sbatch ./scorep.mpt.sbatch
```

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

BT-MZ Summary Analysis Report Examination

```
% ls
bt-mz_B.4 scorep.mpt.sbatch bt-mz.mpt.<jobid>.uv2.out
scorep_4x4_sum scorep_4x4_sum_filtered
% ls scorep_4x4_sum_filtered
profile.cubex  scorep.cfg






% square scorep_4x4_sum_filtered

       [CUBE GUI showing summary analysis report]
```

- Creates experiment directory
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)

- Interactive exploration with CUBE (including profile post-procesing)

Score-P:
Advanced Application Instrumentation

Advanced Application Instrumentation: Score-P Wrapper Scripts

- 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

Score-P: Advanced Measurement Configuration

Advanced Measurement Configuration: Sampling

- Sampling as an additional source of events while measurement
- Novel combination of sampling events and instrumentation of MPI, OpenMP, ...
  - Sampling replaces compiler instrumentation (instrument with `--nocompiler` to further reduce overhead)
  - 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_SMPLING_EVENTS=perf_cycles@2000000

srun_ps -n $NPROCS -t $OMP_NUM_THREADS $EXE

% sbatch ./scorep.mpt.sbatch
```

- Set new configuration variable to enable sampling

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

Advanced Measurement Configuration: Metrics

- 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

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

- Available resource usage metrics
- Note:

(1) Not all fields are maintained on each platform.

(2) Check scope of metrics (per process vs. per thread)

Advanced Measurement Configuration: CUDA

- Record CUDA events with the CUPTI interface

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

- 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

Score-P User Instrumentation API

- Can be used to mark initialization, solver & other phases
  - Annotation macros ignored by default
  - Enabled with [--user] flag of instrumenter
  - Defines SCOREP_USER_ENABLE
- 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 Cpreprocessor)

```
#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 New 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
- Subscribe to news@score-p.org, to be up to date