Virtual Institute – High Productivity Supercomputing

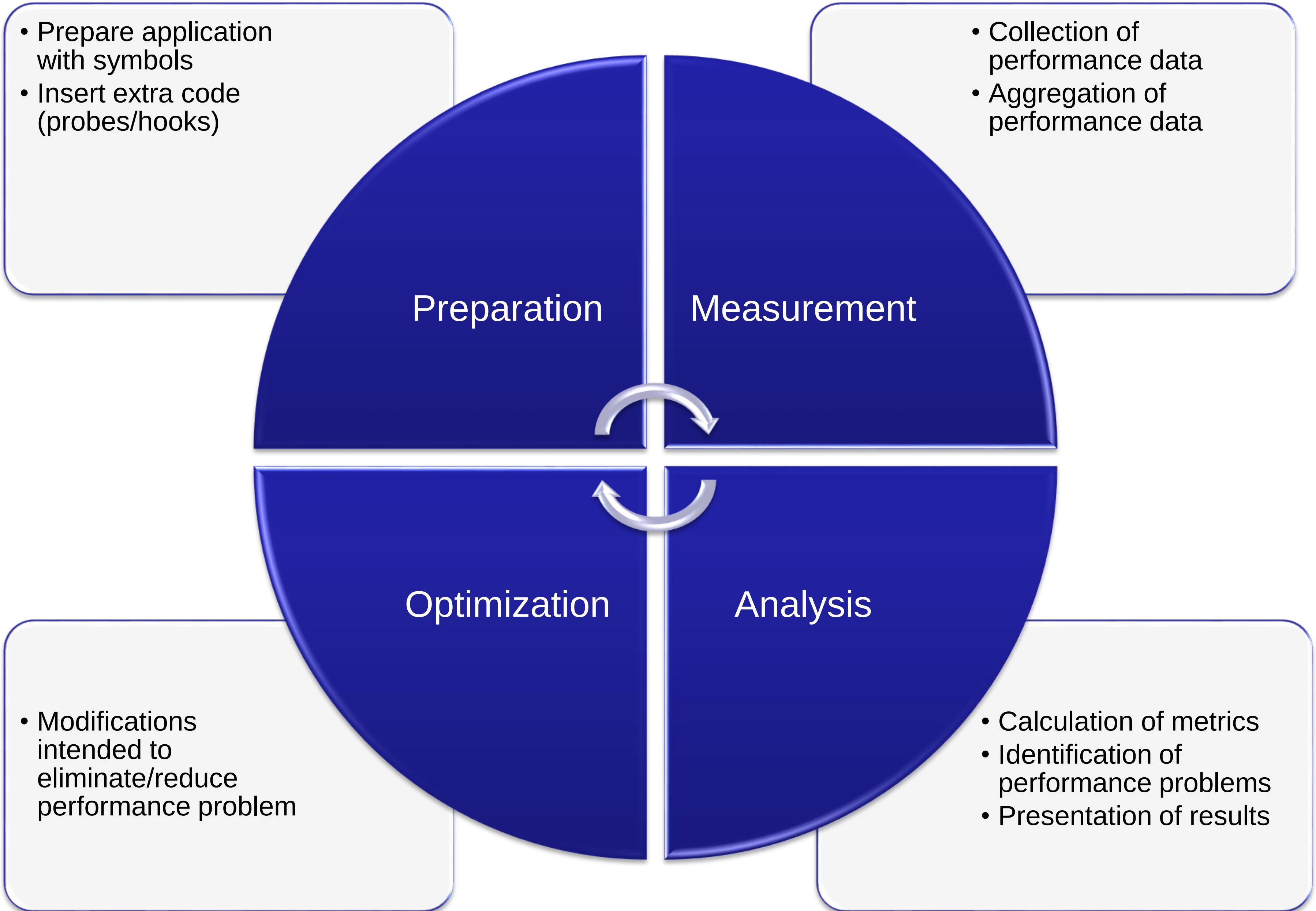
VI-HPS

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

VI-HPS Team

Barcelona Supercomputing Center
Centro Nacional de Supercomputación
German Research School for Simulation Sciences
RWTH Rheinisch-Westfälische Technische Hochschule Aachen
TUM Technische Universität München
Universität Stuttgart
Université de Versailles Saint-Quentin-en-Yvelines
Jülich Forschungszentrum
Lawrence Livermore National Laboratory
Technische Universität Dresden
University of Oregon
The University of Tennessee UT

Performance engineering workflow



Preparation

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

Measurement

- Collection of performance data
- Aggregation of performance data

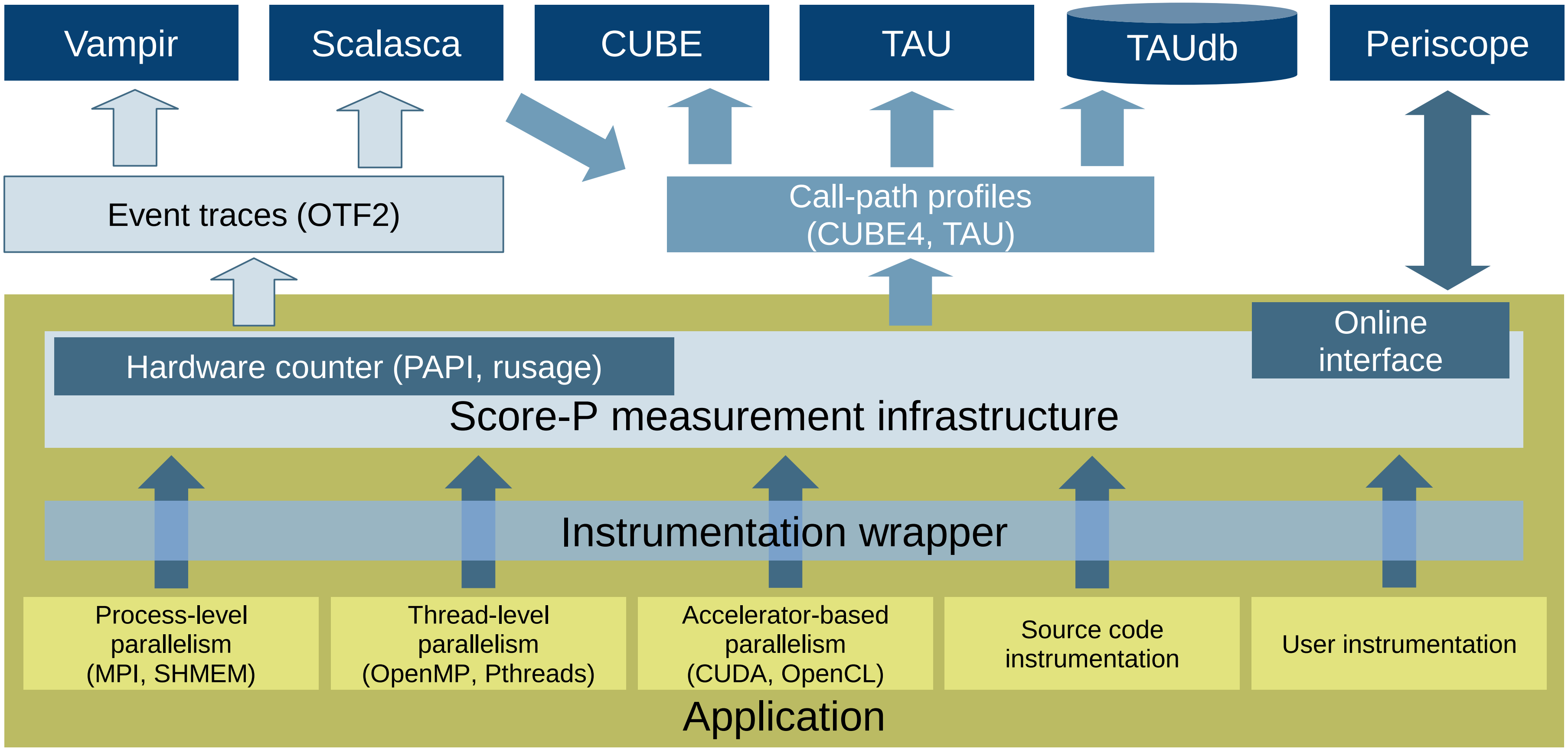
Analysis

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

Optimization

- Modifications intended to eliminate/reduce performance problem

Score-P Overview


Vampir
Scalasca
CUBE
TAU
TAUdb
Periscope
Event traces (OTF2)
Call-path profiles (CUBE4, TAU)
Hardware counter (PAPI, rusage)
Online interface
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism (MPI, SHMEM)
Thread-level parallelism (OpenMP, Pthreads)
Accelerator-based parallelism (CUDA, OpenCL)
Source code instrumentation
User instrumentation
Application

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, SHMEM, OpenMP PTHREAD, CUDA, OpenCL and combinations
- 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

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: Cray XC40 Hornet
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

- Set up prefered program environment compilers
  - PrgEnv-cray with CCE compilers is default
  - PrgEnv-gnu, PrgEnv-intel, PrgEnv-pgi also available

```
% module swap PrgEnv-cray PrgEnv-gnu
```

- Copy tutorial sources to your working directory, ideally on a parallel filesystem ($SCRATCH)

```
% cd $SCRATCH
% tar zxvf ~hpcscabw/tutorial/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

NPB-MZ-MPI / BT Instrumentation

- Load required modules

```
% module use /zhome/academic/HLRS/xhp/xhprt/privatemodules

% module load scorep
% module load cube

% module li
```

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.
#---------------------------------------------------------------------
COMPFLAGS = -fopenmp -ffixed-line-length-none # gnu
...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#-----------------------------------------------------
#MPIF77 = ftn

# Alternative variants to perform instrumentati
...
MPIF77 = scorep --user ftn

# 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=C NPROCS=8
cd BT-MZ; make CLASS=C NPROCS=8 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 B
scorep ftn -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  scorep ftn -c  -O3 -fopenmp timers.f
scorep ftn -O3 -fopenmp -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_C.8
make: Leaving directory 'BT-MZ'
```

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 ...]
```

Summary Measurement Collection

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

```
% cd bin.scorep
% cp ../jobscript/hornet/scorep.pbs .
```

- Check settings

```
% vim scorep.pbs

export NPB_MZ_BLOAD=0
export OMP_NUM_THREADS=6
export SCOREP_EXPERIMENT_DIRECTORY=scorep_sum

aprun -n $NPROCS -d $OMP_NUM_THREADS $EXE
```

- Submit job

```
% qsub scorep.pbs
```

Summary Measurement Collection

- Check the output of the application run

```
% less scorep_mzmpibt.o167691

 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)

 Calculated speedup =     15.96

 Time step    1

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

BT-MZ Summary Analysis Report Examination

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

```
% ls
bt-mz_C.8  scorep_sum
% ls scorep_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with CUBE

```
% cube scorep_sum/profile.cubex

              [CUBE 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

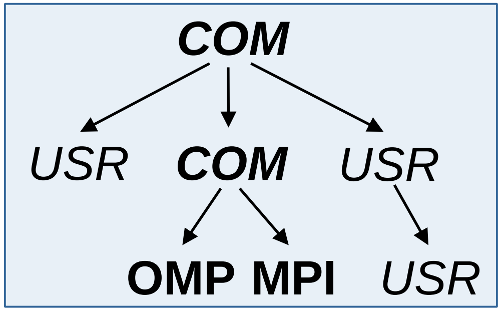
```
% scorep-score scorep_sum/profile.cubex

Estimated aggregate size of event trace:                   159 GB
Estimated requirements for largest trace buffer (max_buf):  20 GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):        20 GB
(hint: When tracing set SCOREP_TOTAL_MEMORY=20GB to avoid intermediate flush
 or reduce requirements using USR regions filters.)

flt     type       max_buf[B]        visits time[s] time[%] ti
         ALL 21,377,442,117 6,554,106,201 4946.18   100.0
         USR 21,309,225,314 6,537,020,537 2326.51    47.0
         OMP     65,624,896    16,327,168 2607.63    52.7         159.71  OMP
         COM      2,355,080       724,640    2.49     0.1           3.43  COM
         MPI        236,827        33,856    9.56     0.2         282.29  MPI
```

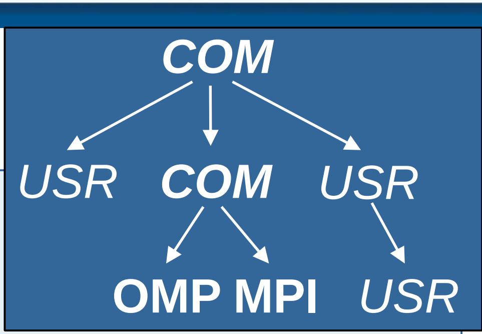
159 GB total memory
20 GB per rank!

- 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_sum/profile.cubex
  [...]
 [...] flt type          max_tbc          time      % region
flt type      max_buf[B]          visits time[s] time[%] time/visit[us]region
     ALL 21,377,442,117 6,554,106,201 4946.18   100.0           0.75  ALL
     USR 21,309,225,314 6,537,020,537 2326.51    47.0           0.36  USR
                  4,896    16,327,168 2607.63    52.7         159.71  OMP
                  5,080       724,640    2.49     0.1           3.43  COM
                  6,827        33,856    9.56     0.2         282.29  MPI

                  2,086 2,110,313,472  651.44    13.2           0.31  matvec_sub_
     USR  6,883,222,086 2,110,313,472  720.38    14.6           0.34  matmul_sub_
     USR  6,883,222,086 2,110,313,472  881.32    17.8           0.42  binvcrhs_
     USR    293,617,584    87,475,200   29.93     0.6           0.34  binvrhs_
     USR    293,617,584    87,475,200   33.03     0.7           0.38  lhsinit_
     USR    101,320,128    31,129,600    7.78     0.2           0.25  exact_solution_
```

More than 18 GB just for these 6 regions

BT-MZ Summary Analysis Score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~159 GB
  - Maximum trace buffer size would be ~20 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

- 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_sum/profile.cubex

Estimated aggregate size of event trace:                   521 MB
Estimated requirements for largest trace buffer (max_buf):  66 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):        78 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=78MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)
```

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

BT-MZ Summary Analysis Report Filtering

- Score report breakdown by region

```
% scorep-score -r -f ../config/scorep.filt \
> scorep_sum/profile.cubex
flt type     max_buf[B]        visits time[s] time[%] time/visit[us] region
 -   ALL 21,377,442,117 6,554,106,201 4946.18   100.0           0.75 ALL
 -           09,225,314 6,537,020,537 2326.51    47.0           0.36 USR
 -           65,624,896    16,327,168 2607.63    52.7         159.71 OMP
 -            2,355,080       724,640    2.49     0.1           3.43 COM
 -              236,827        33,856    9.56     0.2         282.29 MPI

 *   ALL     68,216,855    17,085,673 2622.30    53.0         153.48 ALL-FLT
 +   FLT 21,309,225,262 6,537,020,528 2323.88    47.0           0.36 FLT
 -   OMP     65,624,896    16,327,168 2607.63    52.7         159.71 OMP-FLT
 *   COM      2,355,080       724,640    2.49     0.1           3.43 COM-FLT
 -   MPI        236,827        33,856    9.56     0.2         282.29 MPI-FLT
 *   USR             52             9    2.63     0.1      292158.12 USR-FLT

 +   USR  6,883,222,086 2,110,313,472  651.44    13.2           0.31 matvec_sub_
 +   USR  6,883,222,086 2,110,313,472  720.38    14.6           0.34 matmul_sub_
 +   USR  6,883,222,086 2,110,313,472  881.32    17.8           0.42 binvcrhs_
 +   USR    293,617,584    87,475,200   29.93     0.6           0.34 binvrhs_
 +   USR    293,617,584    87,475,200   33.03     0.7           0.38 lhsinit_
 +   USR    101,320,128    31,129,600    7.78     0.2           0.25 exact_solution_
```

Filtered routines marked with '+'

BT-MZ Filtered Summary Measurement

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

```
%vim scorep.pbs

export OMP_NUM_THREADS=6
export SCOREP_EXPERIMENT_DIRECTORY=scorep_sum_with_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt

aprun -n $NPROCS -d $OMP_NUM_THREADS $EXE
```

  - Submit job

```
%qsub scorep.pbs
```

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

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

BT-MZ Trace Measurement Collection...

- Adjust configuration and re-run the application using the tracing mode of Score-P

```
% vim scorep.pbs

export OMP_NUM_THREADS=6
export SCOREP_EXPERIMENT_DIRECTORY=scorep_trace
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_ENABLE_TRACING=true
export SCOREP_ENABLE_PROFILING=false
export SCOREP_TOTAL_MEMORY=300M

aprun -n $NPROCS -d $OMP_NUM_THREADS $EXE
```

- Submit job

```
%qsub scorep.pbs
```

BT-MZ Trace Measurement Collection...

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

```
% vampir scorep_bt-mz_B_4x4_trace/traces.otf2
```

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
```

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

- 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

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: support@score-p.org