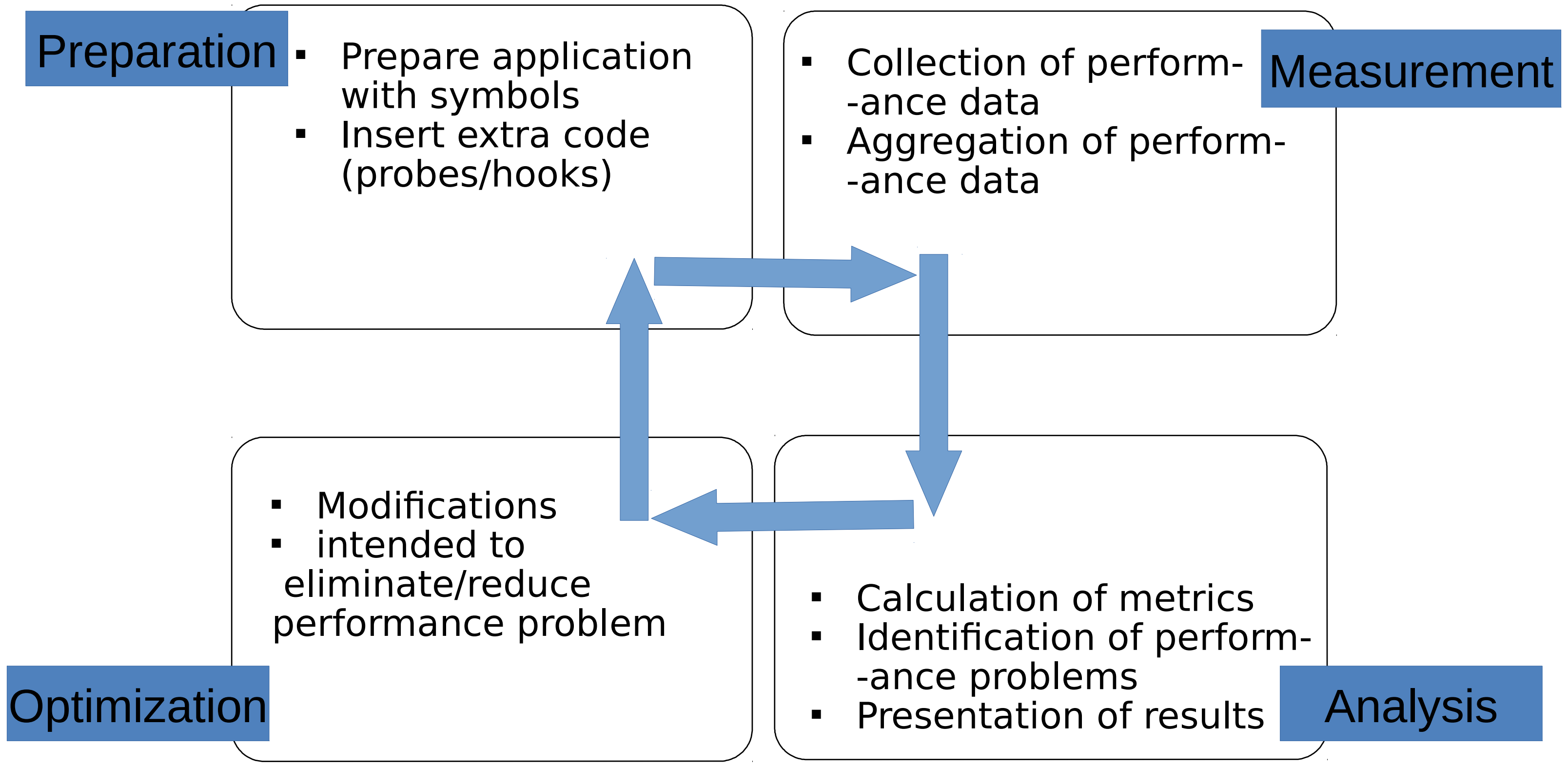
Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir

Scalable performance measurement infrastructure for parallel codes

Performance Engineering Workflow



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

Measurement
- Collection of perform- -ance data
- Aggregation of perform- -ance data

Analysis
- Calculation of metrics
- Identification of perform- -ance problems
- Presentation of results

Optimization
- Modifications
- intended to eliminate/reduce performance problem

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
- German Research School for Simulation Sciences, Aachen, 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)
  - Can be used for all measurement system and analysis tools
  - Insert measurement calls into code at important event.
  - Automatic instrumentation during compilation, semi-automatic using POMP2 directives, manual(Score-P User API), automatic source-code instrumentation using the PDToolkit-based instrumenter
  - `scorep` command need to be prefixed while compilation and linking of application
- After instrumentation, measurement run using application executable

Score-P functionality

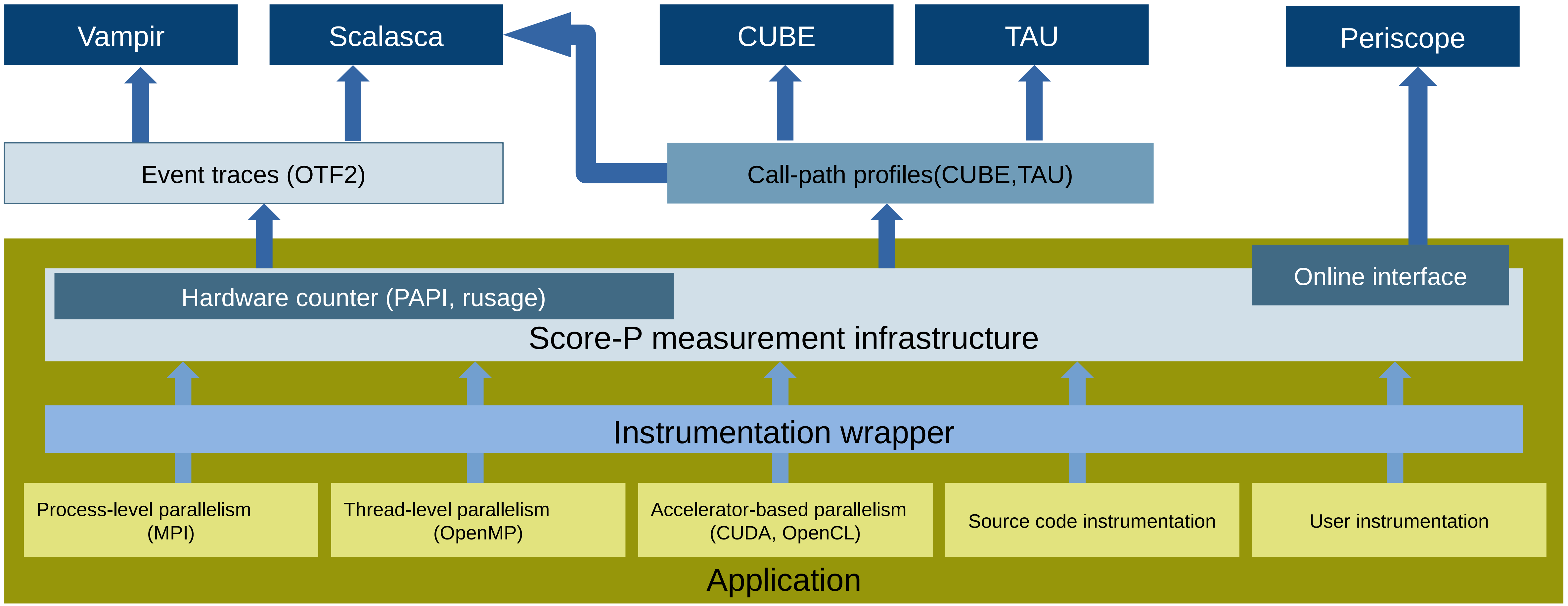
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI, OpenMP, and hybrid parallelism (and serial)
- Enhanced functionality (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, OpenMP, basic CUDA 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 Architecture


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

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
  - Development based on meritocratic governance model
  - Open for contributions and new partners

Hands-on:

NPB-MZ-MPI / BT

NPB-MZ-MPI: Setup Environment

- NPB (NAS Parallel Benchmark)- unstructured adaptive mesh, parallel I/O, multi-zone applications, and computational grids.
- Problem sizes - different classes
- BT-MZ (MPI+ OpenMP) - uneven-size zones within a problem class, increased number of zones as problem class grows
- Benchmark Classes: A, B(8X8 Zones), C, D, E, W, S
- Copy NPB3.3-MZ-MPI from Master Account to your supermuc account.
- Extract the benchmark

```
% tar zxvf tutorial.tar.gz
% cd tutorial/NPB3.3-MZ-MPI
```

NPB-MZ-MPI: Load Module

- Load module scorep. The default version of scorep (1.4.2) will be loaded.

```
% module load scorep
```

- PAPI Metric Support: Use the scorep trunk binary as Environment variable. Do not load scorep module.
  - Check if PAPI Environment parameters are available.

```
%  scorep-info config-summary
```

NPB-MZ-MPI / BT instrumentation

```
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------
OPENMP = -fopenmp    # GCC Compiler
#OPENMP =-openmp     # Intel Compiler
#MPIF77 = mpiifort –mmic

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

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

- Turn on GCC Compiler
- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: 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=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 --user --static mpiifort –mmic -c  -O3 -openmp bt.f
 [...]
cd ../common;  scorep --user --static mpiifort –mmic -c  -O3 \
-openmp timers.f
scorep --user --static mpiifort –mmic –O3 -openmp \
-o ../bin.scorep/bt-mz_B.4 \
…....
Built executable ../bin.scorep/bt-mz_B.4
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/supermuc/run.ll .
% vim run.ll
```

- Check settings

```
export MP_SINGLE_THREAD=no
export OMP_NUM_THREADS=4
PROCS=4
CLASS=B
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum
….
# Application load balancing
export NPB_MZ_BLOAD=0
# Execute the Application
mpiexec -n $PROCS ./bt-mz_$CLASS.$PROCS
```

Summary measurement collection

- Submit the job

```
% llsubmit run.ll
```

- Check the output of the application run named as job$(jobid).out

```
AS 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 =     71.69
 Time step    1
  ...
 Time step    200
```

BT-MZ summary analysis report examination

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

```
% ls
bt-mz_B.4  scorep_bt-mz_sum
% ls scorep_bt-mz_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with Cube

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

       [CUBE GUI showing summary analysis report]
```

Hint:
Copy 'profile.cubex' to Live-DVD environment using 'scp' to improve responsiveness of GUI

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

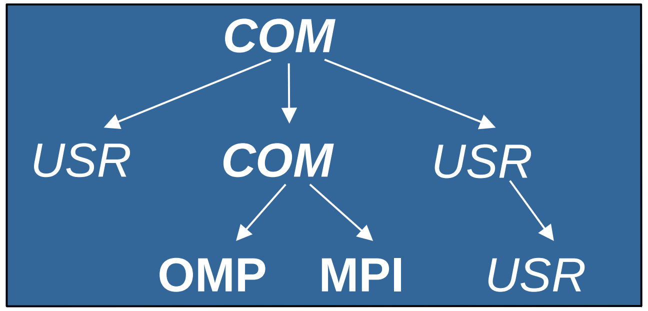
```
% scorep-score scorep_bt-mz_sum/profile.cubex

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
(hint: When tracing set SCOREP_TOTAL_MEMORY=10GB to avoid intermediate flushes or [illegible]uce requirements using USR
 regions filters.)

flt     type     max_buf[B]        visits  time[s] time[%] time/visit[us]  region
         ALL 10,690,201,270 1,634,071,293 25428.04   100.0          15.56  ALL
         USR 10,666,890,182 1,631,138,069 11936.07    46.9           7.32  USR
         OMP     22,025,152     2,743,808 13468.98    53.0        4908.87  OMP
         COM      1,183,650       182,100     7.99     0.0          43.88  COM
         MPI        102,286         7,316    14.99     0.1        2049.34  MPI
```

40 GB total memory
10 GB per rank!

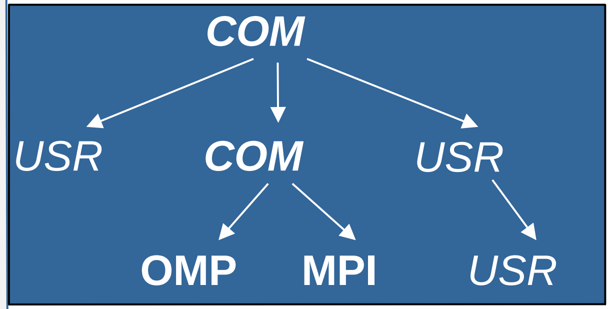
- Region/callpath classification
  - **MPI** pure MPI functions
  - **OMP** pure OpenMP functions/regions
  - **USR** user-level local computation
  - **COM** "combined" USR+OpenMP/MPI
  - **ANY/ALL** aggregate of all region types



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   10,690,201,270   1,634,071,293    25428.04     100.0            15.56      ALL
      USR   10,666,890,182   1,631,138,069    11936.07      46.9             7.32      USR
      OMP       22,025,152       2,743,808    13468.98      53.0          4908.87      OMP
      COM        1,183,650         182,100        7.99       0.0            43.88      COM
      MPI          102,286           7,316       14.99       0.1          2049.34      MPI

      USR    3,421,305,420     522,844,416     3828.28      15.1             7.32  binvcrhs_
      USR    3,421,305,420     522,844,416     3825.05      15.0             7.32  matvec_sub_
      USR    3,421,305,420     522,844,416     3821.52      15.0             7.31  matmul_sub_
      USR      150,937,332      22,692,096      167.71       0.7             7.39  binvrhs_
      USR      150,937,332      22,692,096      171.41       0.7             7.55  lhsinit_
      USR      112,194,160      17,219,840      122.10       0.5             7.09  exact_solution_
  [...]
```



More than
10 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 47% 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_REGION_NAMES_END

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

Estimated aggregate size of event trace:                   89 MB
Estimated requirements for largest trace buffer (max_buf): 23 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       31 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=31MB to avoid \
>intermediate flushes
 or reduce requirements using USR regions filters.)
```

- Reduce Measurement overhead
  - filter small number of frequently executed user functions.

89 MB of memory in total, 23 MB per rank!

(Including 2 metric values)

- Report scoring with prospective filter listing 6 USR regions

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

flt     type         max_buf[B]         time[s]      time[%]         region

 *       ALL         23,315,586        13491.97         53.1         ALL-FLT
 +       FLT     10,666,885,710        11936.07         46.9         FLT
 -       OMP         22,025,152        13468.98         53.0         OMP-FLT
 *       COM          1,183,650            7.99          0.0         COM-FLT
 -       MPI            102,286           14.99          0.1         MPI-FLT
 *       USR              4,498            0.00          0.0         USR-FLT


 +       USR      3,421,305,420         3828.28         15.1          binvcrhs_
 +       USR      3,421,305,420         3825.05         15.0          matvec_sub_
 +       USR      3,421,305,420         3821.52         15.0          matmul_sub_
 +       USR        150,937,332          167.71          0.7          binvrhs_
 +       USR        150,937,332          171.41          0.7          lhsinit_
 +       USR        112,194,160          122.10          0.5          exact_solution_

 […]
```

Filtered routines marked with '+'

BT-MZ filtered summary measurement

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

```
[..]
export MP_SINGLE_THREAD=no
export OMP_NUM_THREADS=4
PROCS=4
CLASS=B
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_with_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
```

- Submit job

BT-MZ filtered summary measurement

- 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:                   89MB
Estimated requirements for largest trace buffer (max_buf): 23MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       31MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=31MB 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 23,315,586 2,933,913  298.38   100.0         101.70  ALL
         OMP 22,025,152 2,743,808  291.36    97.6         106.19  OMP
         COM  1,183,650   182,100    5.33     1.8          29.25  COM
         MPI    102,286     7,316    1.69     0.6         231.32  MPI
         USR      4,498       689    0.00     0.0           4.08  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.

Score-P: Advanced Measurement Configuration

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 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 ...]
```

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

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

Warnings and Tips Regarding Tracing

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

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_with_trace
% export SCOREP_FILTERING_FILE=../config/scorep.filt
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_ENABLE_PROFILING=false
% export SCOREP_TOTAL_MEMORY=50M
% export SCOREP_METRIC_PAPI=PAPI_L2_TCM,PAPI_FP_OPS
```

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

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

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 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
```

```
#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… */
}
```

Fortran (requires C preprocessor)

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