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

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

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

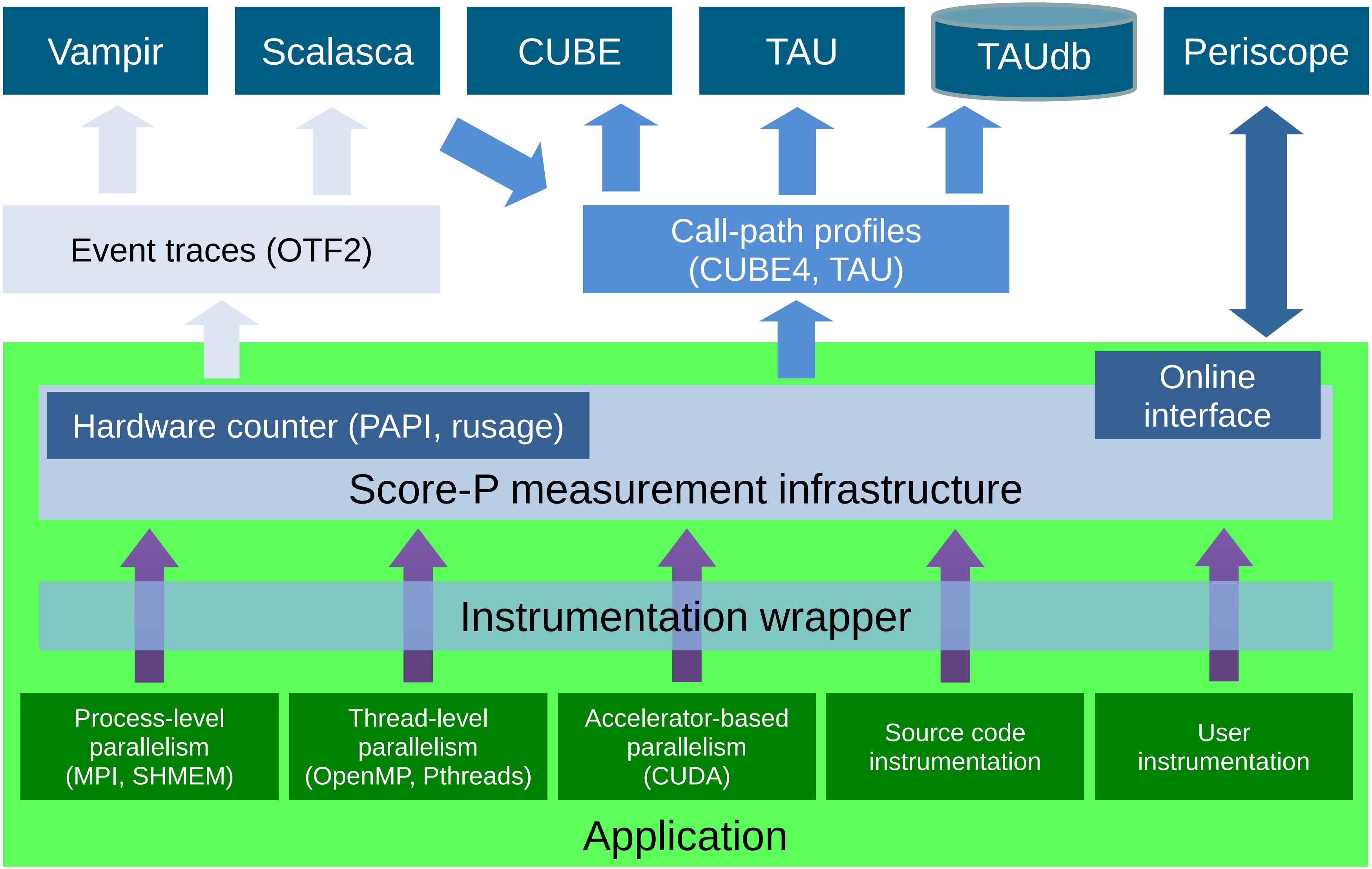
GEFÖRDERT VOM

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 Architecture



Score-P Functionality

- Provide typical functionality for HPC performance tools
- Instrumentation (various methods)
  - Multi-process paradigms (MPI, SHMEM)
  - Thread-parallel paradigms (OpenMP, POSIX threads)
  - Accelerator-based paradigms (CUDA)
  - And their combination
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Highly scalable I/O functionality

- Support all fundamental concepts of partner's tools

Non-functional Requirements

- Portability: support all major HPC platforms (e.g., Linux, IBM Blue Gene and AIX, SGI, Cray, Fujitsu K/FX10, ARM)
- Scalability: petascale, supporting platforms with more than 100K cores
- Low measurement overhead: typically less than 5%
- Robustness and QA: Nightly Builds, Continuous Integration Testing Framework
- Easy and uniform installation through UNITE framework
- 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
- 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

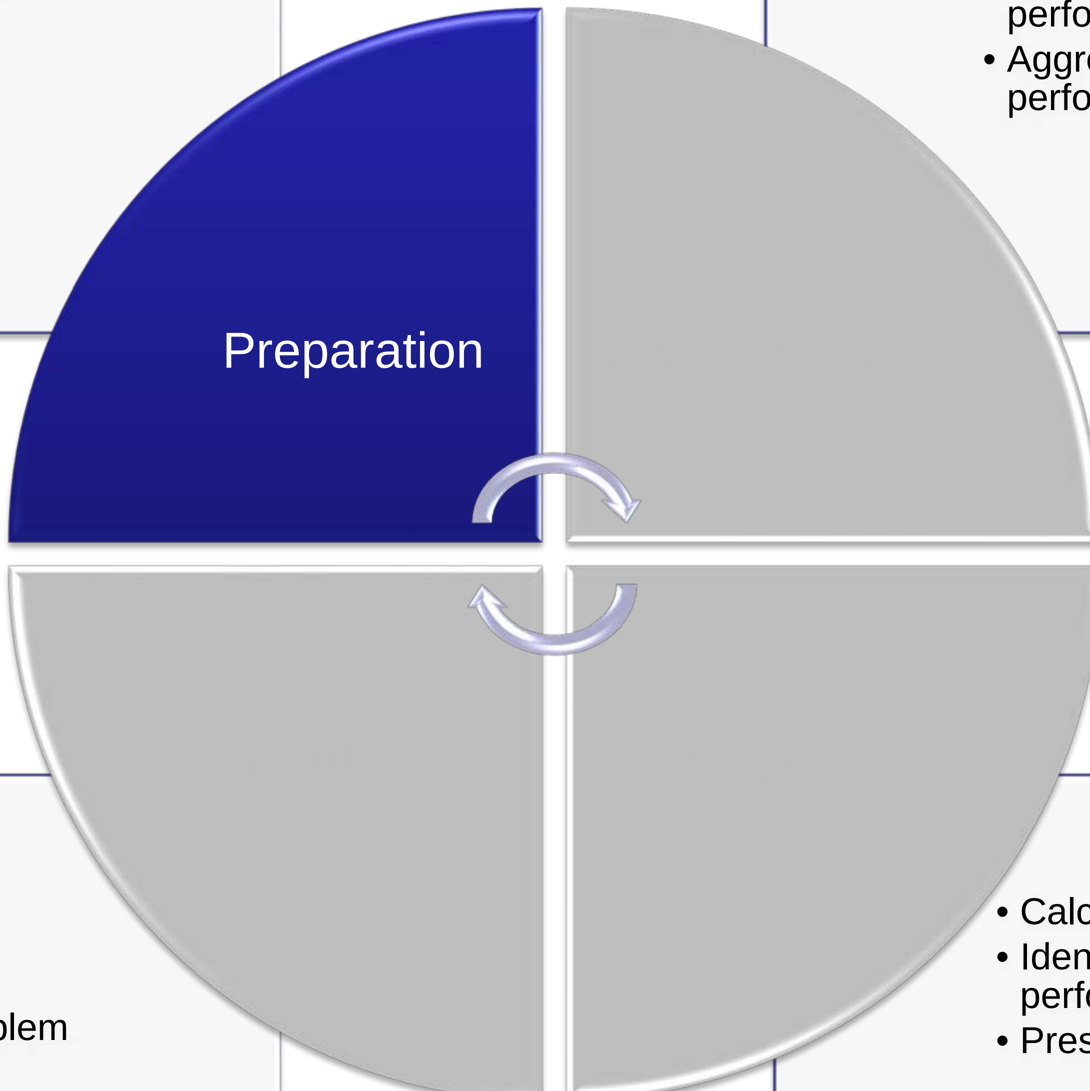
- Commitment to joint long-term cooperation

Score-P application measurement hands-on: NPB-OMP / BT

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

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-OMP / Setup Environment

- Load modules (Intel compilers)

```
% module load intel-compilers/14.0.0

% module load score-p/1.3/intel11.1_intelMPI4.0.0
```

- or use modules for GCC compiler

```
% module load gcc/4.9.2

% module load score-p/1.3/gcc4.8.2_intelMPI4.0.0
```

- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: F77

```
#               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 OpenMP programs
#---------------------------------------------------------------------
#F77 = ifort

# Alternative variants to perform instrumentation
...
F77 = scorep  ifort

# This links OMP Fortran programs; usually the same as ${F77}
FLINK   = $(F77)
...
```

Uncomment the Score-P compiler wrapper specification

NPB-OMP / BT Instrumented Build

- Return to root directory and clean-up

```
% make clean
```

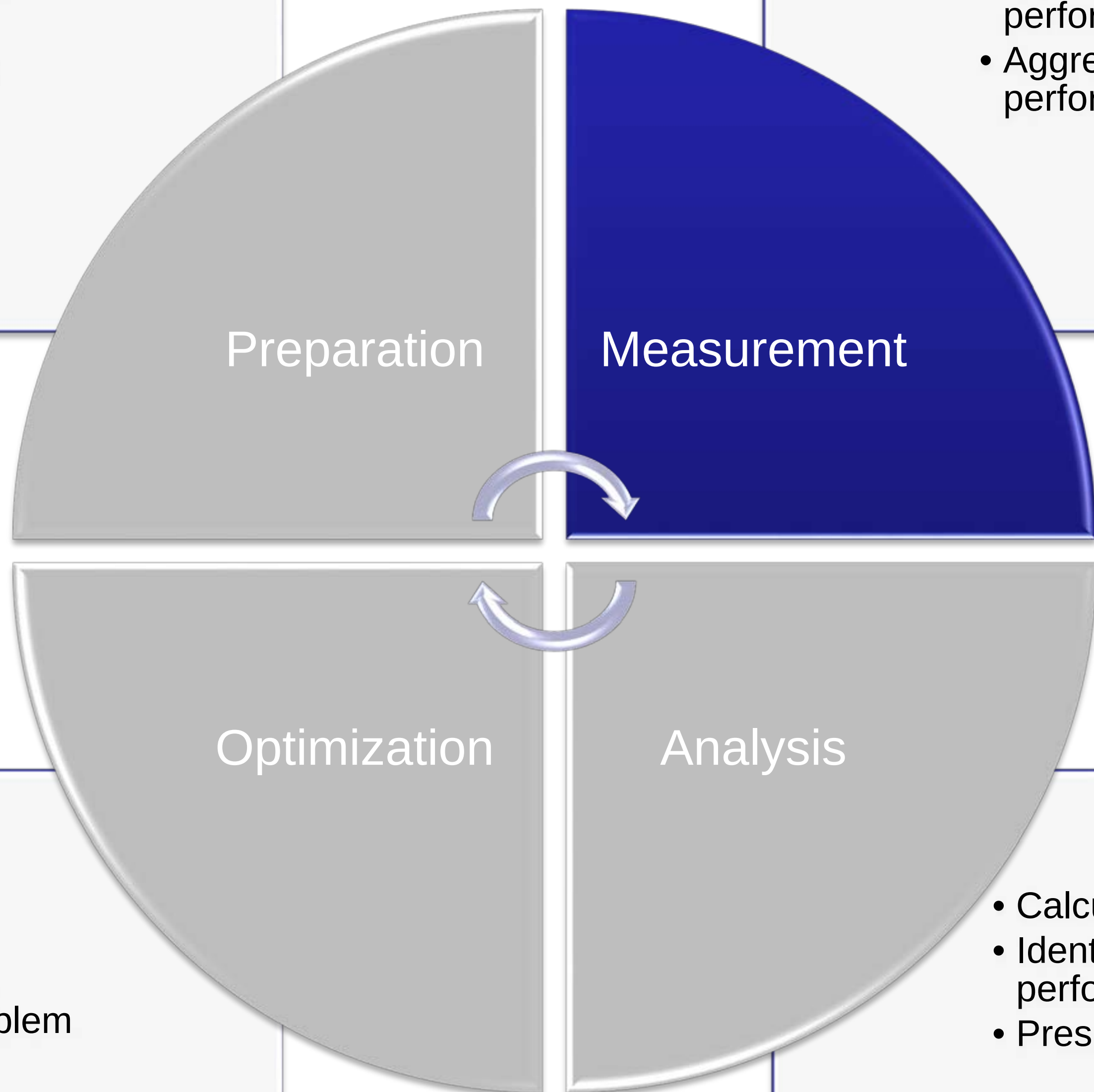
- Re-build executable using Score-P compiler wrapper

```
% make bt CLASS=B
cd BT; make CLASS=B VERSION=
make: Entering directory 'BT'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt B
scorep ifort -O –g -openmp bt.f
 [...]
cd ../common; scorep ifort –O -g -openmp timers.f
scorep ifort –O -g -openmp -o ../bin.scorep/bt_B \
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_B
make: Leaving directory 'BT'
```

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

16

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

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 ...]
```

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

```
% cd bin.scorep
% cp ../jobscript/intel/run.pbs .
% vim run.pbs
```

```
...
#
# Score-P configuration
#
module load score-p

export SCOREP_EXPERIMENT_DIRECTORY=scorep_sum
...
```

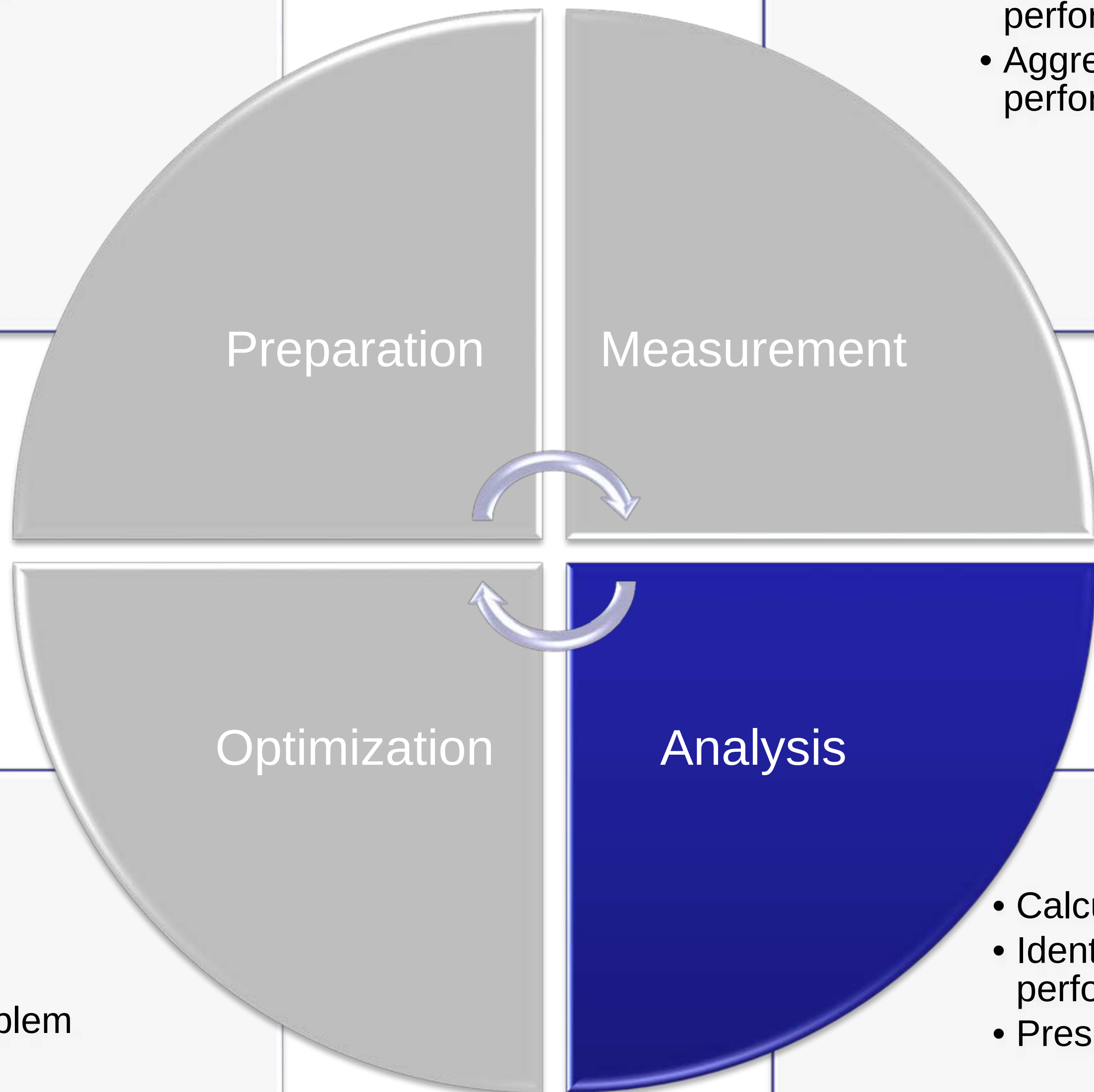
- Launch instrumented application

```
% qsub run.pbs
% qstat –u $USER
% cat run.o<id>

 NAS Parallel Benchmarks (NPB3.3-OMP) - BT Benchmark

 Size:   102x 102x 102
 Iterations: 200    dt:   0.000300
 Number of available threads:     12

 Time step    1
 Time step   20

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

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

21

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

BT 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_B  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 an 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

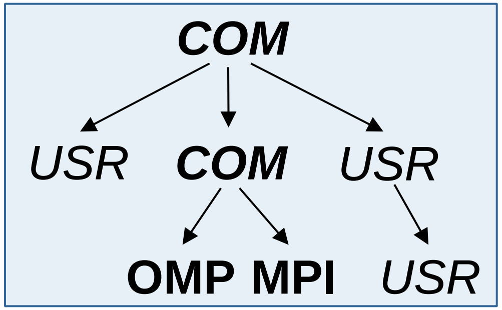
1. Reference preparation for validation
2. Program instrumentation
3. Summary measurement collection
4. Summary analysis report examination
5. **Summary experiment scoring**
6. Summary measurement collection with filtering
7. Filtered summary analysis report examination
8. Event trace collection
9. Event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

- Report scoring as textual output

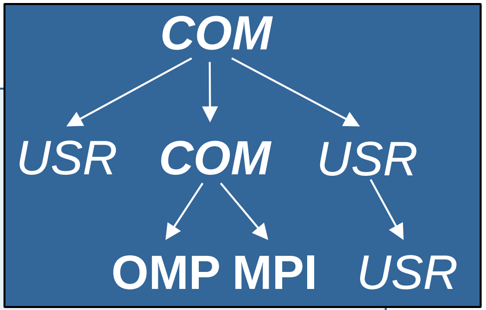
```
% scorep-score scorep_sum/profile.cubex
Estimated aggregate size of event trace:                   35967144402 bytes
Estimated requirements for largest trace buffer (max_tbc): 4545841910 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)

flt type         max_tbc         time       % region
     ALL      9046029930       799.89   100.0 ALL
     USR      9025830154       383.72    48.0 USR
     OMP        19113728       411.49    51.4 OMP
     COM          997150         0.75     0.1 COM
     MPI           88898         3.92     0.5 MPI
```

33.5 GB total memory
4.5 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)



- Score report breakdown by region




```
% scorep-score -r scorep_sum/profile.cubex
  [...]
flt type          max_tbc         time      % region
         ALL   4545841910       799.89  100.0 ALL
         USR   4534316886       383.72   48.0 USR
         OMP     10751472       411.49   51.4 OMP
                   567270         0.75    0.1 COM
                   206282         3.92    0.5 MPI

               1452428010       152.50   19.1 binvcrhs_
               1452428010        98.73   12.3 matvec_sub_
         USR   1452428010       117.78   14.7 matmul_sub_
         USR     64472760         5.01    0.6 binvrhs_
         USR     64472760         6.62    0.8 lhsinit_
         USR     48082848         3.07    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_...
 [...]
```

More than
4 GB just for
these 6 regions

BT-MZ Summary Analysis Score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~34 GB
  - Maximum trace buffer size would be ~4.5 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

- 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:                   82119842 bytes
Estimated requirements for largest trace buffer (max_tbc): 11528962 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.)
```

77 MB of memory in total,
10 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_tbc         time      % region
 *   ALL         20203582       416.17   52.0 ALL-FLT
 +   FLT       9025826370       383.72   48.0 FLT
 -   OMP         19113728       411.49   51.4 OMP-FLT
 *   COM           997150         0.75    0.1 COM-FLT
 -   MPI            88898         3.92    0.5 MPI-FLT
 *   USR             3806         0.00    0.0 USR-FLT

 +   USR       2894950740       152.50   19.1 binvcrhs_
 +   USR       2894950740        98.73   12.3 matvec_sub_
 +   USR       2894950740       117.78   14.7 matmul_sub_
 +   USR        127716204         5.01    0.6 binvrhs_
 +   USR        127716204         6.62    0.8 lhsinit_
 +   USR         94933520         3.07    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_...
  [...]
```

Filtered routines marked with ‘+’

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

BT-MZ Filtered Summary Measurement

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

```
% vim run.pbs
```

  – Adjust configuration

```
...
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_sum_with_filter
% export SCOREP_FILTERING_FILE=../config/scorep.filt
...
```

  – Submit job

```
% qsub run.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

BT-MZ Tuned Summary Analysis Report Score

- Scoring of new analysis report as textual output

```
% scorep-score scorep_sum_with_filter/profile.cubex
Estimated aggregate size of event trace:                   82119842 bytes
Estimated requirements for largest trace buffer (max_tbc): 11528962 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        218.95  100.0 ALL
     OMP        19113728        216.94   99.1 OMP
     COM          997150          0.73    0.3 COM
     MPI           88898          1.27    0.6 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.

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
```

- 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

- 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)

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

BT Trace Measurement Collection...

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

```
% 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=50M
% export SCOREP_METRIC_PAPI=PAPI_L2_TCM,PAPI_FP_OPS
```

  - Submit job

```
% qsub run.pbs
```

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

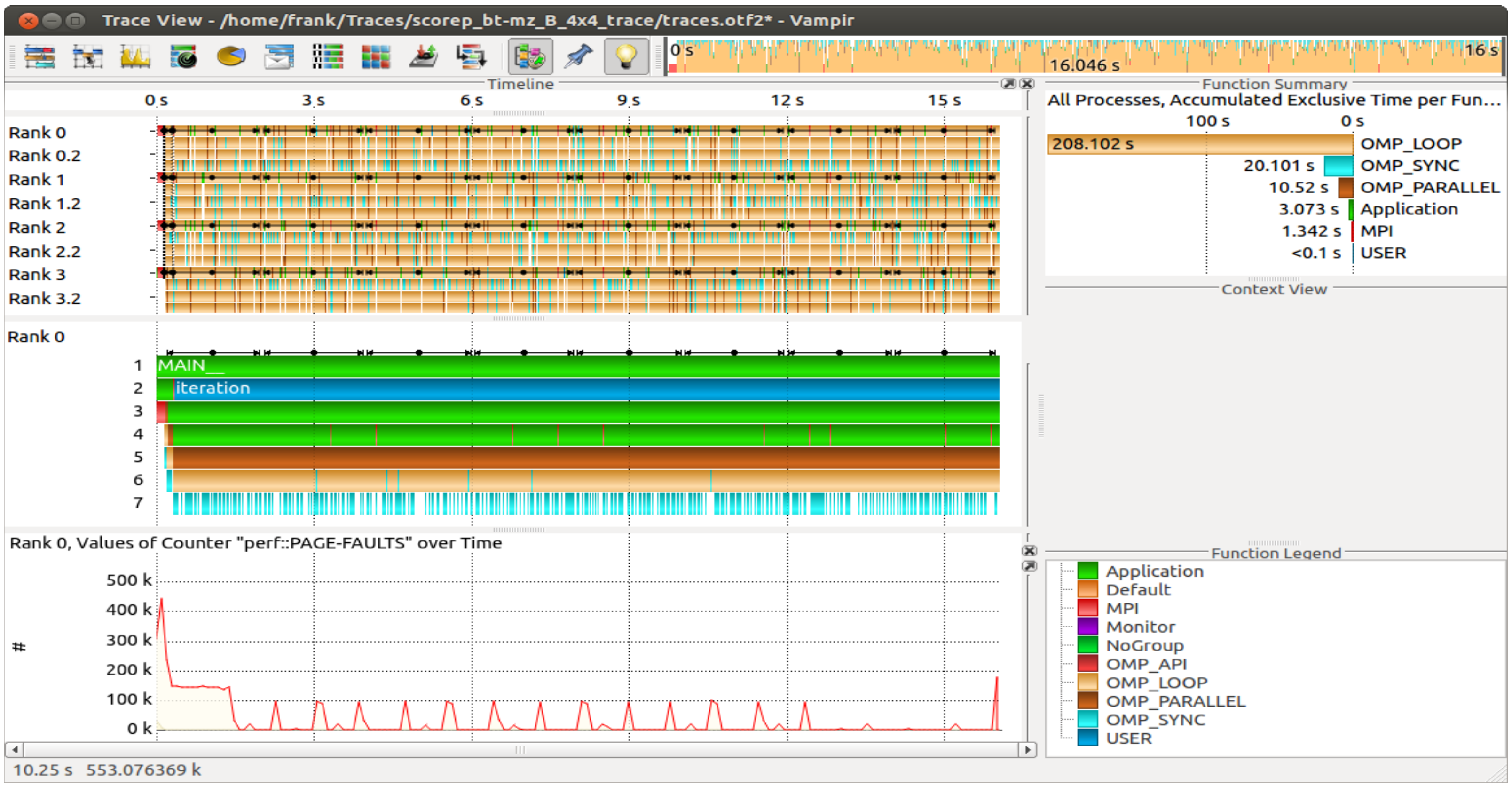
```
% vampir scorep_trace/traces.otf2
```

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**

- Interactive trace exploration with Vampir

```
% vampir scorep_bt-mz_B_8x4_trace/traces.otf2
```



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

- Disable OPARI instrumentation of fine-grained OpenMP constructs

```
% PREP="scorep --opari='--disable=flush,locks'"
```

- Comma-separated list of constructs
  - atomic
  - critical
  - master
  - flush
  - single
  - ordered
  - locks
  - sync (all of the above)
  - region (explicit POMP annotations)

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++

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