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

VI-HPS Team

Score-P
Scalable performance measurement infrastructure for parallel codes

# 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

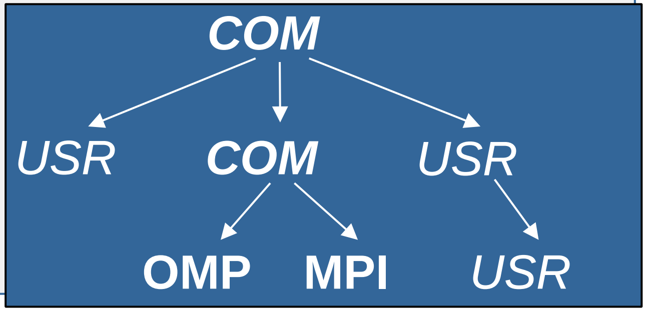
# BT-MZ summary analysis result scoring

```
% scorep-score scorep_bt-mz_B.8x4.<jobid>/profile.cubex

Estimated aggregate size of event trace:                      40 GB
Estimated requirements for largest trace buffer (max_buf):     6 GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):           6 GB
(warning: The memory requirements cannot 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 5,372,029,854 1,634,097,563  429.32   100.0           0.26  ALL
         USR 5,358,738,138 1,631,138,913  180.91    42.1           0.11  USR
         OMP    12,389,148     2,743,808  240.04    55.9          87.49  OMP
         COM       665,210       182,120    1.28     0.3           7.02  COM
         MPI       237,358        32,722    7.08     1.6         216.41  MPI
```

40 GB total memory
6 GB per rank!



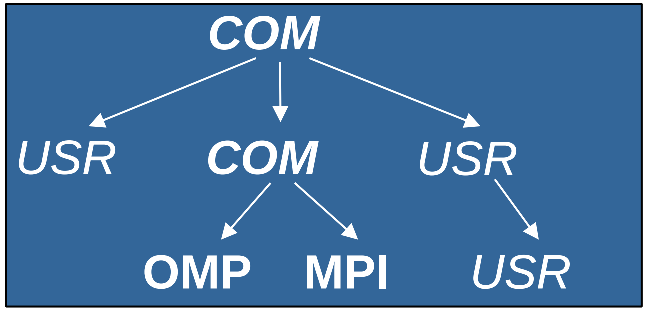
- Report scoring as textual output

- 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_bt-mz_B.8x4.<jobid>/profile.cubex
  [...]
  [...]
flt type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
     ALL 5,372,029,854 1,634,097,563  429.32   100.0           0.26  ALL
     USR 5,358,738,138 1,631,138,913  180.91    42.1           0.11  USR
     OMP    12,389,148     2,743,808  240.04    55.9          87.49  OMP
     COM       665,210       182,120    1.28     0.3           7.02  COM
     MPI       237,358        32,722    7.08     1.6         216.41  MPI

     USR 1,716,505,830   522,844,416   48.13    11.2           0.09  matmul_sub_
     USR 1,716,505,830   522,844,416   37.62     8.8           0.07  matvec_sub_
     USR 1,716,505,830   522,844,416   87.89    20.5           0.17  binvcrhs_
     USR    76,195,080    22,692,096    3.65     0.8           0.16  lhsinit_
     USR    76,195,080    22,692,096    2.29     0.5           0.10  binvrhs_
     USR    56,825,184    17,219,840    1.23     0.3           0.07  exact_solution_
```



More than 4.9 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 ~6 GB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.75% 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 42% 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_bt-mz_B.8x4.<jobid>/profile.cubex

Estimated aggregate size of event trace:                   91 MB
Estimated requirements for largest trace buffer (max_buf): 13 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       21 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=21MB to avoid \
>intermediate flushes
 or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

91 MB of memory in total,
21 MB per rank!

Including 2 metric values:
232 MB
33 MB
41 MB

# BT-MZ summary analysis report filtering

```
% scorep-score -r –f ../config/scorep.filt \
      scorep_bt-mz_sum/profile.cubex
flt type     max_buf[B]        visits time[s] time[%] time/        region
flt type    max_buf[B]        visits time[s] time[%] time/visit[us]  region
-   ALL 5,372,029,854 1,634,097,563  429.32   100.0      0.26  ALL
 -   USR 5,358,738,138 1,631,138,913  180.91    42.1      0.11  USR
 -   OMP    12,389,148     2,743,808  240.04    55.9     87.49  OMP
 -   COM       665,210       182,120    1.28     0.3      7.02  COM
 -   MPI       237,358        32,722    7.08     1.6    216.41  MPI

 *   ALL    13,296,370     2,960,083  248.48    57.9     83.94  ALL-FLT
 +   FLT 5,358,733,484 1,631,137,480  180.84    42.1      0.11  FLT
 -   OMP    12,389,148     2,743,808  240.04    55.9     87.49  OMP-FLT
 *   COM       665,210       182,120    1.28     0.3      7.02  COM-FLT
 -   MPI       237,358        32,722    7.08     1.6    216.41  MPI-FLT
 *   USR         4,680         1,433    0.07     0.0     50.17  USR-FLT

 +   USR 1,716,505,830   522,844,416   48.13    11.2      0.09  matmul_sub_
 +   USR 1,716,505,830   522,844,416   37.62     8.8      0.07  matvec_sub_
 +   USR 1,716,505,830   522,844,416   87.89    20.5      0.17  binvcrhs_
 +   USR    76,195,080    22,692,096    3.65     0.8      0.16  lhsinit_
 +   USR    76,195,080    22,692,096    2.29     0.5      0.10  binvrhs_
 +   USR    56,825,184    17,219,840    1.23     0.3      0.07  exact_solution_
```

- Score report breakdown by region

Filtered routines marked with '+'

# BT-MZ filtered summary measurement

```
% cd bin.scorep
% <editor> scorep.sbatch.B.8
[…]
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
[…]

% sbatch scorep.sbatch.B.8
```

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

- Submit job

# 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

Score-P

# 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_SAMPLING_EVENTS=perf_cycles@2000000

% OMP_NUM_THREADS=4 mpiexec –np 4 ./bt-mz_W.4
```

- Set new configuration variable to enable sampling

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

# Advanced Measurement Configuration: Memory Recording

- Record calls to memory API functions and there resulting memory usage changes
  - C, C++, MPI, and SHMEM
  - Fortran only for GNU Compilers
- Supports profile and trace generation
  - Memory leaks are recorded in the profile additionally
  - Resulting traces are not supported by Scalasca yet

```
% export SCOREP_MEMORY_RECORDING=true
% export SCOREP_MPI_MEMORY_RECORDING=true

% OMP_NUM_THREADS=4 mpiexec –np 4 ./bt-mz_W.4
```

- Set new configuration variable to enable memory recording

- Available since Score-P 2.0

# 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
```

  - Use the metric names in one of the **SCOREP_METRIC_PAPI** or **SCOREP_METRIC_PAPI_PER_PROCESS** measurement configuration variables separated by comma

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)

- Use the member names in one of these measurement configuration variables separated by comma:
  - **SCOREP_METRIC_RUSAGE**
  - **SCOREP_METRIC_RUSAGE_PER_PROCESS**

# 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
  - For most compilers, this can be automatically achieved by having an uppercase file extension, e.g., `main.F` or `main.F90`

# 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
```

```
#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++

# Score-P:
# Conclusion and Outlook

Score-P

# Project management

- 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

# Future features

- Scalability to maximum available CPU core count
- Support for emerging architectures and new programming models
- Features currently worked on:
  - User provided wrappers to 3rd party libraries
  - Hardware and MPI topologies
  - Basic support of measurement without re-compiling/-linking
  - I/O recording
  - Java recording
  - Persistent memory recording (e.g., PMEM, NVRAM, ...)

# Further Information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods) and sampling
  - 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