Score-P:
Specialized Measurements and Analyses
Mastering build systems

- Hooking up the Score-P instrumenter `scorep` into complex build environments like `Autotools` or `CMake` was always challenging.
- Score-P provides 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`.

```bash
% SCOREP_WRAPPER=off \
> cmake .. \
> -DCMAKE_C_COMPILER=scorep-icc \
> -DCMAKE_CXX_COMPILER=scorep-icpc
```

- 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.
Mastering C++ applications

- Automatic compiler instrumentation greatly disturbs C++ applications because of frequent/short function calls => Use sampling instead
- Novel combination of sampling events and instrumentation of MPI, OpenMP, ...
  - Sampling replaces compiler instrumentation (instrument with --nocompiler to further reduce overhead) => Filtering not needed anymore
  - 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
Mastering C++ applications
Wrapping calls to 3\textsuperscript{rd} party libraries

- Enables users to install library wrappers for any C/C++ library
- Intercept calls to a library API
  - no need to either build the library with Score-P or add manual instrumentation to the application using the library
  - no need to access the source code of the library, header and library files suffice
- Score-P needs to be executed with \texttt{--libwrap=}...

- Execute \texttt{scorep-libwrap-init} for directions:

  \begin{itemize}
  \item Step 1: Initialize the working directory
  \item Step 2: Add library headers
  \item Step 3: Create a simple example application
  \item Step 4: Further configure the build parameters
  \item Step 5: Build the wrapper
  \item Step 6: Verify the wrapper
  \item Step 7: Install the wrapper
  \item Step 8: Verify the installed wrapper
  \end{itemize}
Wrapping calls to 3\textsuperscript{rd} party libraries

- Generate your own library wrappers by telling \texttt{scorep-libwrap-init} how you would compile and link an application, e.g. using FFTW

\begin{verbatim}
\% scorep-libwrap-init \ 
> --name=fftw \ 
> --prefix=$PREFIX \ 
> -x c \ 
> --cppflags="-O3 -DNDEBUG -openmp -I$FFTW_INC" \ 
> --ldflags="-L$FFTW_LIB" \ 
> --libs="-lfftw3f -lfftw3" \ 
> working_directory
\end{verbatim}

- Generate and build wrapper

\begin{verbatim}
\% cd working_directory 
\% ls                         # (Check README.md for instructions) 
\% make                       # Generate and build wrapper 
\% make check                 # See if header analysis matches symbols 
\% make install               # 
\% make installcheck          # More checks: Linking etc.
\end{verbatim}
Wrapping calls to 3rd party libraries

- MPI + OpenMP
- Calls to FFTW library
Mastering application memory usage

- Determine the maximum heap usage per process
- Find high frequent small allocation patterns
- Find memory leaks
- Support for:
  - C, C++, MPI, and SHMEM (Fortran only for GNU Compilers)
  - Profile and trace generation (profile recommended)
    - Memory leaks are recorded only in the profile
    - 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
Mastering application memory usage

Different maximum heap usages per ranks
Mastering application memory usage

Memory leaks
Mastering heterogeneous applications

- Record CUDA applications and device activities
  
  ```
  % export SCOREP_CUDA_ENABLE=gpu,kernel,idle
  ```

- Record OpenCL applications and device activities
  
  ```
  % export SCOREP_OPENCL_ENABLE=api,kernel
  ```

- Record OpenACC applications
  
  ```
  % export SCOREP_OPENACC_ENABLE=yes
  ```

- Can be combined with CUDA if it is a NVIDIA device
  
  ```
  % export SCOREP_CUDA_ENABLE(kernel
  ```
Mastering heterogeneous applications
Enriching measurements with performance counters

- Record metrics from PAPI:
  ```bash
  % export SCOREP_METRIC_PAPI=PAPI_TOT_CYC
  % export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_TCM
  
  # Use PAPI tools to get available metrics and valid combinations:
  % papi_avail
  % papi_native_avail
  ``

- Record metrics from Linux perf:
  ```bash
  % export SCOREP_METRIC_PERF=cpu-cycles
  % export SCOREP_METRIC_PERF_PER_PROCESS=LLC-load-misses
  
  # Use the perf tool to get available metrics and valid combinations:
  % perf list
  ``

- Write your own metric plugin
  - Repository of available plugins: https://github.com/score-p

Only the master thread records the metric (assuming all threads of the process access the same L3 cache)
Score-P user instrumentation API

- No replacement for automatic compiler instrumentation

- Can be used to further subdivide functions
  - E.g., multiple loops inside a function

- Can be used to partition application into coarse grain phases
  - E.g., initialization, solver, & finalization

- Enabled with --user flag to Score-P instrumenter

- Available for Fortran / C / C++
Score-P user instrumentation API (Fortran)

```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++)

```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++)

```cpp
#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

```fortran
#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
```

```c
#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
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:
- MPI-3 RMA support
- OpenMP tool support (OMPT)
- I/O recording
- Basic support of measurements without re-compiling/-linking
- Java recording
- Persistent memory recording (e.g., PMEM, NVRAM, ...)

ANF WORKSHOP (OHP, FRANCE, 16-20 SEPT 2019)
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 3-clause 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