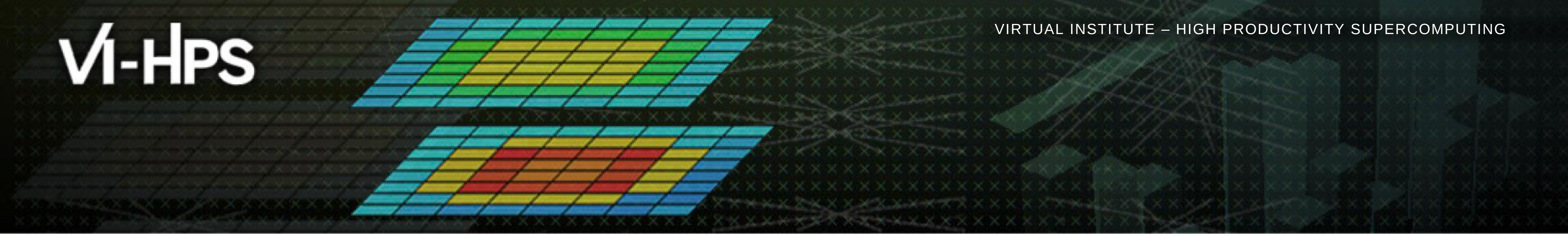
VI-HPS



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

arm · FAU Friedrich-Alexander Universität Erlangen-Nürnberg · Lawrence Livermore National Laboratory · RWTH Aachen University · Technische Universität Dresden · Université de Versailles St-Quentin-en-Yvelines · Universität Stuttgart · Barcelona Supercomputing Center Centro Nacional de Supercomputación · Jülich Forschungszentrum · lrz Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities · Technische Universität Darmstadt · TUM Technische Universität München · University of Oregon · The University of Tennessee

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 of 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
  - but there wasn’t much GPU-related performance information
  - and 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 customisation & 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

Mastering heterogeneous applications

- Record CUDA application events and device activities

```
% export SCOREP_CUDA_ENABLE=default,flushatexit
```

- Record OpenCL application events and device activities

```
% export SCOREP_OPENCL_ENABLE=api,kernel
```

- Record OpenACC application events

```
% export SCOREP_OPENACC_ENABLE=yes
```

  - Can be combined with CUDA if it is a NVIDIA device

```
% export SCOREP_CUDA_ENABLE=kernel
```

TeaLeaf CUDA default summary measurement

```
% cd bin.scorep
% cp ../jobscript/marconi100/scorep.sbatch .
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_tea_leaf_sum.default
export SCOREP_CUDA_ENABLE=default,flushatexit
export SCOREP_CUDA_BUFFER=48MB
#export SCOREP_FILTERING_FILE=../config/scorep.filt

# Run the application
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
mpirun ./tea_leaf



% sbatch scorep.sbatch
```

- Set an experiment directory and re-run measurement with default CUDA event configuration

- Submit job

TeaLeaf CUDA extended summary measurement

```
% edit scorep.sbatch
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_tea_leaf_sum.extended
export SCOREP_CUDA_ENABLE=default,driver,sync,flushatexit
export SCOREP_CUDA_BUFFER=48MB
#export SCOREP_FILTERING_FILE=../config/scorep.filt

# Run the application
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
mpirun ./tea_leaf



% sbatch scorep.sbatch
```

- Set new experiment directory and re-run measurement with extended CUDA event configuration

- Submit job

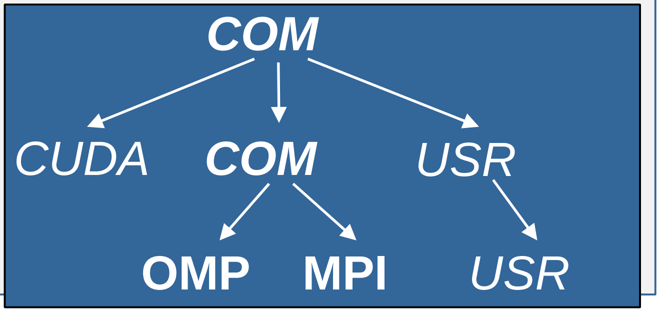
TeaLeaf summary analysis result scoring

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

Estimated aggregate size of event trace:                   535MB
Estimated requirements for largest trace buffer (max_buf): 78MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       86MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=86MB 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     81,323,591    18,774,730  255.08   100.0          13.59  ALL
        CUDA     34,527,454     9,042,214  197.95    77.6          21.89  CUDA
         COM     27,138,254     7,221,112    8.48     3.3           1.17  COM
         MPI     17,421,892     1,823,459   48.19    18.9          26.43  MPI
         USR      2,236,962       687,841    0.43     0.2           0.62  USR
         OMP            678            96    0.03     0.0         343.16  OMP
      SCOREP             41             8    0.00     0.0         201.38  SCOREP
```

535MB total memory
86MB per MPI process



- Report scoring as textual output

- Region/callpath classification
  - **MPI** pure MPI functions
  - **OMP** pure OpenMP regions
  - **CUDA** API regions & kernels
  - **USR** user-level computation
  - **COM** “combined” USR+OpenMP/MPI
  - **ALL** aggregate of all region types

TeaLeaf filtered summary measurement

```
% edit scorep.sbatch
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_tea_leaf_sum.filtered
export SCOREP_CUDA_ENABLE=default,driver,sync,flushatexit
export SCOREP_CUDA_BUFFER=48MB
export SCOREP_FILTERING_FILE=../config/scorep.filt

# Run the application
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
mpirun ./tea_leaf




% sbatch scorep.sbatch
```

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

- Submit job

Score-P filtering

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    void\ __device_stub__*
    void\ __nv_*
    void\ __sti__*
    void\ __wrapper__*
    timer
    cudaGetLastError
    cuLaunchKernel
SCOREP_REGION_NAMES_END

% export SCOREP_FILTERING_FILE=\
../config/scorep.filt
```

Region name filter block using wildcards

Apply filter

- Filtering by source file name
  - All regions in files that are excluded by the filter are ignored
- Filtering by region name
  - All regions that are excluded by the filter are ignored
  - Overruled by source file filter for excluded files
- Apply filter by
  - exporting **`SCOREP_FILTERING_FILE`** environment variable
- Apply filter at
  - Run-time
  - Compile-time (GCC-plugin only, Intel in 7.0 release)
    - Add cmd-line option **`--instrument-filter`**
    - No overhead for filtered regions but recompilation

Source file name filter block

- Keywords
  - Case-sensitive
  - `SCOREP_FILE_NAMES_BEGIN`, `SCOREP_FILE_NAMES_END`
    - Define the source file name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated source file names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
    - Unlike bash, `*` may match a string that contains slashes
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions in source files that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_FILE_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE */foo/bar*
  INCLUDE */filter_test.c
SCOREP_FILE_NAMES_END
```

Region name filter block

- Keywords
  - Case-sensitive
  - `SCOREP_REGION_NAMES_BEGIN`, `SCOREP_REGION_NAMES_END`
    - Define the region name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated region names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_REGION_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE *
  INCLUDE bar foo
          baz
          main
SCOREP_REGION_NAMES_END
```

Region name filter block, mangling

- Name mangling
  - Filtering based on names seen by the measurement system
    - Dependent on compiler
    - Actual name may be mangled
- `scorep-score` names as starting point (e.g. `timer`)
  - Use `*` for Fortran trailing underscore(s) for portability
  - Use `?` and `*` as needed for full signatures or overloading
  - Use `\` to escape special characters

```
void bar(int* a) {
    *a++;
}
int main() {
    int i = 42;
    bar(&i);
    return 0;
}
```

```
# filter bar:
# for gcc-plugin, scorep-score
# displays 'void bar(int*)',
# other compilers may differ

SCOREP_REGION_NAMES_BEGIN
  EXCLUDE void?bar(int?)
SCOREP_REGION_NAMES_END
```

Further information

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