Score-P – A Joint Performance Measurement Run-Time Infrastructure for 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 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
  - 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

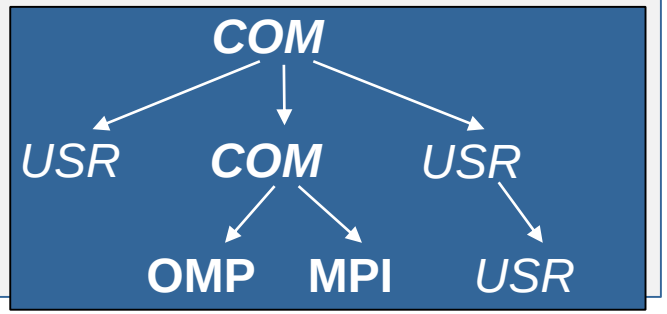
- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

BT-MZ summary analysis result scoring

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

Estimated aggregate size of event trace:                   159GB
Estimated requirements for largest trace buffer (max_buf): 20GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       20GB
(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 21,395,581,557 6,554,106,209 1897.09   100.0           0.29  ALL
         USR 21,309,225,312 6,537,020,537  802.49    42.3           0.12  USR
         OMP     83,713,600    16,327,168 1077.75    56.8          66.01  OMP
         COM      2,355,080       724,640    2.79     0.1           3.85  COM
         MPI        287,524        33,856   14.06     0.7         415.18  MPI
      SCOREP             41             8    0.00     0.0         540.82  SCOREP
```



- Report scoring as textual output

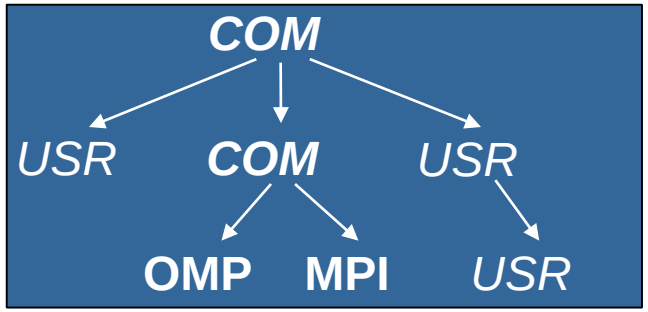
159 GB total memory
20 GB per rank!

- Region/callpath classification
  - **MPI** pure MPI functions
  - **OMP** pure OpenMP regions
  - **USR** user-level computation
  - **COM** "combined" USR+OpenMP/MPI
  - **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 21,395,581,557 6,554,106,209 1897.09   100.0           0.29  ALL
         USR 21,309,225,312 6,537,020,537  802.49    42.3           0.12  USR
         OMP     83,713,600    16,327,168 1077.75    56.8          66.01  OMP
         COM      2,355,080       724,640    2.79     0.1           3.85  COM
         MPI        287,524        33,856   14.06     0.7         415.18  MPI
      SCOREP             41             8    0.00     0.0         540.82  SCOREP

         USR  6,883,222,086 2,110,313,472  322.04    17.0           0.15  binvcrhs
         USR  6,883,222,086 2,110,313,472  249.47    13.2           0.12  matmul_sub
         USR  6,883,222,086 2,110,313,472  206.76    10.9           0.10  matvec_sub
         USR    293,617,584    87,475,200   11.76     0.6           0.13  lhsinit
         USR    293,617,584    87,475,200    8.97     0.5           0.10  binvrhs
         USR    101,320,128    31,129,600    2.81     0.1           0.09  exact_solution
```



More than
19.8 GB just for these
6 regions

BT-MZ summary analysis score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~159 GB
  - Maximum trace buffer size would be ~20 GB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.6% 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 (for tracing)
  - Specify a (compile-time) 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 -c 2 \
      scorep_bt-mz_sum/profile.cubex

Estimated aggregate size of event trace:                   16211MB
Estimated requirements for largest trace buffer (max_buf): 203MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       215MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=215MB to avoid
       intermediate flushes or reduce requirements using
       USR regions filters.)
```

- Report scoring with prospective filter listing 7 USR regions

1.6 GB of memory in total, 215 MB per rank!

(Including 2 metric values)

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
                                                  visit[us]
 -       ALL 21,395,581,557 6,554,106,209 1897.09   100.0       0.29  ALL
 -       USR 21,309,225,312 6,537,020,537  802.49    42.3       0.12  USR
 -       OMP     83,713,600    16,327,168 1077.75    56.8      66.01  OMP
 -       COM      2,355,080       724,640    2.79     0.1       3.85  COM
 -       MPI        287,524        33,856   14.06     0.7     415.18  MPI
 -    SCOREP             41             8    0.00     0.0     540.82  SCOREP

 *       ALL     86,356,295    17,085,681 1095.27    57.7      64.10  ALL-FLT
 +       FLT 21,309,225,262 6,537,020,528  801.82    42.3       0.12  FLT
 -       OMP     83,713,600    16,327,168 1077.75    56.8      66.01  OMP-FLT
 *       COM      2,355,080       724,640    2.79     0.1       3.85  COM-FLT
 -       MPI        287,524        33,856   14.06     0.7     415.18  MPI-FLT
 *       USR             50             9    0.67     0.0   74440.90  USR-FLT
 -    SCOREP             41             8    0.00     0.0     540.82  SCOREP-FLT

 +       USR  6,883,222,086 2,110,313,472  322.04    17.0       0.15  binvcrhs
 +       USR  6,883,222,086 2,110,313,472  249.47    13.2       0.12  matmul_sub
 +       USR  6,883,222,086 2,110,313,472  206.76    10.9       0.10  matvec_sub
 +       USR    293,617,584    87,475,200   11.76     0.6       0.13  lhsinit
 +       USR    293,617,584    87,475,200    8.97     0.5       0.10  binvrhs
 +       USR    101,320,128    31,129,600    2.81     0.1       0.09  exact_solution
```

- Score report breakdown by region (w/o additional metrics)

Filtered routines marked with '+'

BT-MZ filtered summary measurement

```
% cd bin.scorep
% cp ../jobscript/archer2/scorep.sbatch .
# edit scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_METRIC_RUSAGE=ru_stime
#export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss

# Run the application
srun ./bt-mz_C.8

% sbatch scorep.sbatch
```

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

Score-P 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

% 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 and Intel only)
    - 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. `matvec_sub_`)
  - 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
```

New: generate initial filter file

```
% score-scorep --help
[…]
 -g [<list>] Generation of an initial filter file with the name
             'initial_scorep.filter'. A valid parameter list has the form
             KEY=VALUE[,KEY=VALUE]*. By default, uses the following control
             parameters:

                 `bufferpercent=1,timepervisit=1`

             A region is included in the filter file (i.e., excluded from
             measurement) if it matches all of the given conditions, with the
             following keys:
             - `bufferpercent`       : estimated memory requirements exceed the
                                       given threshold in percent of the total
                                       estimated trace buffer requirements
             - `bufferabsolute`      : estimated memory requirements exceed
                                       the given absolute threshold in MB
             - `visits`              : number of visits exceeds the given
                                       threshold
             […]
```

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*

```
% 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 *Makefile*s (`SCOREP_WRAPPER_INSTRUMENTER_FLAGS` and `SCOREP_WRAPPER_COMPILER_FLAGS`)
- Run `scorep-wrapper --help` for a detailed description and the available wrapper scripts of the Score-P installation

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
- 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