Hands-on / Demo: Summary measurement scoring & filtering

Markus Geimer
Jülich Supercomputing Centre

Congratulations!?

- If you made it this far, you successfully used Score-P to
  - instrument the application, and
  - analyze its execution with a summary measurement

- ... 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 execution for validation

- 1.0 Program instrumentation
- 1.1 Summary measurement collection

- 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

Scalasca convenience command: scalasca –examine (square)

```
% scalasca -examine
Scalasca 2.4: analysis report explorer
usage: square [-v] [-s] [-f filtfile] [-F] <experiment archive | cube file>
   -c <none | quick | full> : Level of sanity checks for newly created reports
   -F                       : Force remapping of already existing reports
   -f filtfile              : Use specified filter file when doing scoring
   -s                       : Skip display and output textual score report
   -v                       : Enable verbose mode
   -n                       : Do not include idle thread metric
```

- Performs analysis report post-processing and opens the report in the Cube browser

BT-MZ summary analysis report examination

- Score summary analysis report (using `scorep-score` tool)

```
% scalasca -examine -s  scorep_bt-mz_C_8x6_sum
INFO: Post-processing runtime summarization result...
INFO: Score report written to ./scorep_bt-mz_C_8x6_sum/scorep.score
```

- Post-processing and interactive exploration with Cube

```
% scalasca -examine  scorep_bt-mz_C_8x6_sum
INFO: Displaying ./scorep_bt-mz_C_8x6_sum/summary.cubex...

                    [GUI showing summary analysis report]
```

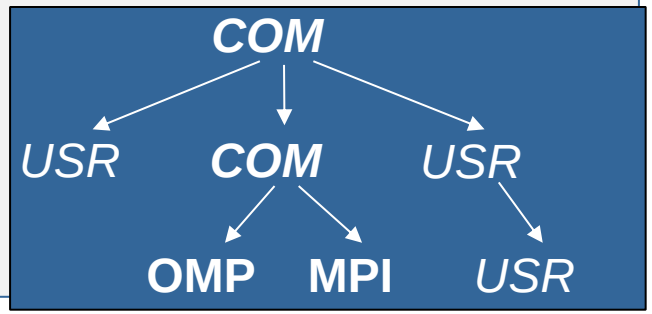
- The post-processing derives additional metrics and generates a structured metric hierarchy

BT-MZ summary analysis result scoring

```
% less scorep_bt-mz_C_8x6_sum/scorep.score

Estimated aggregate size of event trace:                   160 GB
Estimated requirements for largest trace buffer (max_buf):  21 GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):        21 GB
(hint: When tracing set SCOREP_TOTAL_MEMORY=21GB 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 21,518,477,680 6,591,910,441  2590.93   100.0           0.39  ALL
        USR 21,431,996,118 6,574,793,529  1151.70    44.5           0.18  USR
        OMP     83,841,856    16,359,424  1420.95    54.8          86.86  OMP
        COM      2,351,570       723,560     1.93     0.1           2.67  COM
        MPI        288,136        33,928    16.36     0.6         482.19  MPI

 [...]
```

- Report scoring as textual output

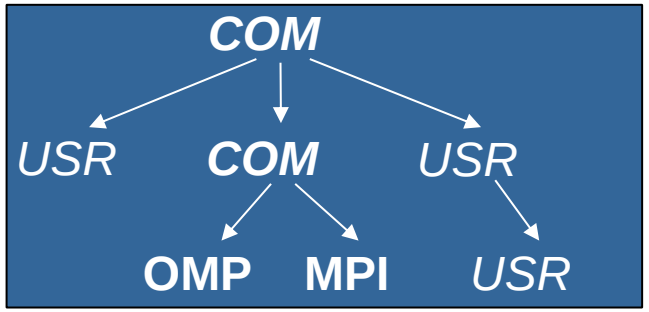
160 GB total memory
21 GB per rank!



- 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

```
% less scorep_bt-mz_C_8x6_sum/scorep.score
  [...]
flt type     max_buf[B]         visits  time[s] time[%] time/visit[us]  region
     ALL 21,518,477,680 6,591,910,441  2590.93   100.0           0.39  ALL
     USR 21,431,996,118 6,574,793,529  1151.70    44.5           0.18  USR
     OMP     83,841,856    16,359,424  1420.95    54.8          86.86  OMP
     COM      2,351,570       723,560     1.93     0.1           2.67  COM
     MPI        288,136        33,928    16.36     0.6         482.19  MPI

     USR  6,883,222,086 2,110,313,472   362.28    14.0           0.28  matmul_sub_
     USR  6,883,222,086 2,110,313,472   288.67    11.1           0.14  matvec_sub_
     USR  6,883,222,086 2,110,313,472   462.38    17.8           0.33  binvcrhs_
     USR    293,617,584    87,475,200    12.39     0.5           0.14  binvrhs_
     USR    293,617,584    87,475,200    17.78     0.7           0.20  lhsinit_
     USR    224,028,792    68,892,672     8.20     0.3           0.12  exact_solution_
```



More than
20 GB just for these 6 regions

BT-MZ summary analysis score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~160 GB
  - Maximum trace buffer size would be ~21 GB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.9% 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 45% 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

```
% cp ../config/scorep.filt .
% cat scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs*
    matmul_sub*
    matvec_sub*
    exact_solution*
    binvrhs*
    lhs*init*
    timer_*
SCOREP_REGION_NAMES_END
% scalasca -examine -s -f scorep.filt scorep_bt-mz_C_8x6_sum
% less scorep_bt-mz_C_8x6_sum/scorep.score_scorep.filt

Estimated aggregate size of event trace:                   661 MB
Estimated requirements for largest trace buffer (max_buf):  83 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):        95 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=95MB to avoid \
>intermediate flushes
 or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

661 MB of memory in total, 95 MB per rank!

BT-MZ summary analysis report filtering

```
% less scorep_bt-mz_C_8x6_sum/scorep.score_scorep.filt
 [...]
flt type     max_buf[B]         visits  time[s] time[%] time/visit[us]  region
 -   ALL 21,518,477,680 6,591,910,441  2590.93   100.0           0.39  ALL
 -   USR 21,431,996,118 6,574,793,529  1151.70    44.5           0.18  USR
 -   OMP     83,841,856    16,359,424  1420.95    54.8          86.86  OMP
 -   COM      2,351,570       723,560     1.93     0.1           2.67  COM
 -   MPI        288,136        33,928    16.36     0.6         482.19  MPI

 *   ALL     86,513,568    17,126,753  1439.24    55.5          84.03  ALL-FLT
 +   FLT 21,431,964,112 6,574,783,688  1151.69    44.5           0.18  FLT
 -   OMP     83,841,856    16,359,424  1420.95    54.8          86.86  OMP-FLT
 *   COM      2,351,570       723,560     1.93     0.1           2.67  COM-FLT
 -   MPI        288,136        33,928    16.36     0.6         482.19  MPI-FLT
 *   USR         32,006         9,841     0.00     0.0           0.33  USR-FLT

 +   USR  6,883,222,086 2,110,313,472   362.28    14.0           0.17  matmul_sub_
 +   USR  6,883,222,086 2,110,313,472   288.67    11.1           0.14  matvec_sub_
 +   USR  6,883,222,086 2,110,313,472   462.38    17.8           0.22  binvcrhs_
 +   USR    293,617,584    87,475,200    12.39     0.5           0.14  binvrhs_
 +   USR    293,617,584    87,475,200    17.78     0.7           0.20  lhsinit_
 +   USR    224,028,792    68,892,672     8.20     0.3           0.12  exact_solution_
 [...]
```

- Score report breakdown by region

Filtered routines marked with '+'

BT-MZ filtered summary measurement

```
% cd bin.scorep
% mv scorep_bt-mz_C_8x6_sum scorep_bt-mz_C_8x6_sum.nofilt
% vim scalasca.sbatch
 ...
# Measurement configuration
#export SCOREP_TOTAL_MEMORY=215M
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
scalasca -analyze -f scorep.filt srun ./bt-mz_$CLASS.$PROCS


% sbatch scalasca.sbatch
```

- Rename old experiment
- Copy filter file
- Re-run measurement with new filter configuration

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 only)
    - Add cmd-line option `--instrument-filter`
    - No overhead for filtered regions but recompilation
  - Compiler-specific options (Intel, IBM xl)

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
    - **Caution:** This may do the wrong thing if the routine name is a prefix of another routine name…
  - Use `?` and `*` as needed for full signatures or overloading

```
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
```