International HPC Summer School 2016:
Scoring-based measurement configuration and automatic trace analysis with Scalasca

Christian Feld
Jülich Supercomputing Centre
Recap: Local installation

- VI-HPS tools not yet installed system-wide
  - Source provided shell code snippet to add local tool installations to $PATH
  - Required for each shell session

```bash
% source /home/roessel/ihpcss16/tools/source.me.gcc-openmpi
```

- Copy tutorial sources to your working directory, ideally on a parallel file system (recommended: $SCRATCH)

```bash
% cd $HOME
% tar-zxvf /home/roessel/ihpcss16/tutorial/npb3.3-mz-mpi.tar.gz
% cd npb3.3-mz-mpi
```
Recap: BT-MZ summary analysis report examination

- Creates experiment directory including:
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)

- Interactive exploration with Paraprof

```plaintext
% cd bin.scorep
% ls -l
bt-mz_C.8
scorep.sbatch.C.8
scorep-C.8-<jobid>.err
scorep-C.8-<jobid>.out
scorep_bt-mz_C.8x7.<jobid>

% ls scorep_bt-mz_C.8x7.<jobid>
profile.cubex scorep.cfg

laptop> scp userid@bridges.psc.edu:~/NPB3.3-MZ-MPI/bin.scorep/
        scorep_bt-mz_C.8x7.<jobid>/profile.cubex .

laptop> paraprof profile.cubex

[Paraprof GUI showing summary analysis report]
[You can use Cube on profile.cubex as well]
```

Hint:
Copy ‘profile.cubex’ to your laptop using ‘scp’ to improve responsiveness of GUI
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)
  - PAPI hardware-counters
- ... 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 Event trace collection with filtering
  - 2.2 Event trace examination & analysis
BT-MZ Summary Analysis Result Scoring

% scorep-score scorep_bt-mz_C.8x7.<jobid>/profile.cubex

Estimated aggregate size of event trace:
Estimated requirements for largest trace buffer (max_buf):
Estimated memory requirements (SCOREP_TOTAL_MEMORY):
(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 21,389,438,207 6,557,153,121 1934.82 100.0 0.30 ALL
USR 21,309,225,314 6,537,020,537  835.65  43.2  0.13 USR
OMP 76,450,336 19,013,888 1087.70  56.2  57.21 OMP
COM 3,525,730 1,084,840 2.20  0.1  2.03 COM
MPI 236,827 33,856 9.28  0.5 274.03 MPI

□ 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

Report scoring as textual output

159 GB total memory
20 GB per rank!
BT-MZ Summary Analysis Report Breakdown

```bash
% scorep-score -r scorep_bt-mz_C.8x7.<jobid>/profile.cubex

[...]
[...]

flt  type  max_buf[B]  visits  time[s]  time[%]  time/visit[us]  region

<table>
<thead>
<tr>
<th>Type</th>
<th>ALL 21,389,438,207</th>
<th>6,557,153,121</th>
<th>1934.82</th>
<th>100.0</th>
<th>0.30</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>USR 21,309,225,314</td>
<td>6,537,020,537</td>
<td>835.65</td>
<td>43.2</td>
<td>0.13</td>
<td>USR</td>
<td></td>
</tr>
<tr>
<td>OMP 76,450,336</td>
<td>19,013,888</td>
<td>1087.70</td>
<td>56.2</td>
<td>57.21</td>
<td>OMP</td>
<td></td>
</tr>
<tr>
<td>COM 3,525,730</td>
<td>1,084,840</td>
<td>2.20</td>
<td>0.1</td>
<td>2.03</td>
<td>COM</td>
<td></td>
</tr>
<tr>
<td>MPI 236,827</td>
<td>33,856</td>
<td>9.28</td>
<td>0.5</td>
<td>274.03</td>
<td>MPI</td>
<td></td>
</tr>
</tbody>
</table>

USR 6,883,222,086 | 2,110,313,472 | 381.69 | 19.7 | 0.18 | binvcrhs_
USR 6,883,222,086 | 2,110,313,472 | 163.88 | 8.5 | 0.08 | matvec_sub_
USR 6,883,222,086 | 2,110,313,472 | 262.46 | 13.6 | 0.12 | matmul_sub_
USR 293,617,584 | 87,475,200 | 9.83 | 0.5 | 0.11 | binvrhs_
USR 293,617,584 | 87,475,200 | 14.86 | 0.8 | 0.17 | lhsinit_
USR 101,320,128 | 31,129,600 | 2.37 | 0.1 | 0.08 | exact_solution_
```

More than 19 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
  - 99.7% 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 43% 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

- Report scoring with prospective filter listing 6 USR regions

```bash
% 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_C.8x7.<jobid>/profile.cubex
```

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

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

(Add space for metric values using -c #)
BT-MZ Summary Analysis Report Filtering

```bash
% scorep-score -r -f ../config/scorep.filt [-c 2] \n> scorep_bt-mz_C.8x7.<jobid>/profile.cubex
```

<table>
<thead>
<tr>
<th>flt</th>
<th>type</th>
<th>max_buf[B]</th>
<th>visits</th>
<th>time[s]</th>
<th>time[%]</th>
<th>time/visit[us]</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALL</td>
<td>21,389,438,207</td>
<td>6,557,153,121</td>
<td>1934.82</td>
<td>100.0</td>
<td>0.30</td>
<td>ALL</td>
</tr>
<tr>
<td></td>
<td>USR</td>
<td>21,309,225,314</td>
<td>6,537,020,537</td>
<td>835.65</td>
<td>43.2</td>
<td>0.13</td>
<td>USR</td>
</tr>
<tr>
<td></td>
<td>OMP</td>
<td>76,450,336</td>
<td>19,013,888</td>
<td>1087.70</td>
<td>56.2</td>
<td>0.1</td>
<td>OMP</td>
</tr>
<tr>
<td></td>
<td>COM</td>
<td>3,525,730</td>
<td>1,084,840</td>
<td>2.20</td>
<td>0.1</td>
<td>2.03</td>
<td>COM</td>
</tr>
<tr>
<td></td>
<td>MPI</td>
<td>236,827</td>
<td>33,856</td>
<td>9.28</td>
<td>0.5</td>
<td>274.03</td>
<td>MPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALL</td>
<td>80,212,945</td>
<td>20,132,593</td>
<td>1099.74</td>
<td>56.8</td>
<td>54.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLT</td>
<td>21,309,225,262</td>
<td>6,537,020,528</td>
<td>835.08</td>
<td>43.2</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OMP</td>
<td>76,450,336</td>
<td>19,013,888</td>
<td>1087.70</td>
<td>56.2</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COM</td>
<td>3,525,730</td>
<td>1,084,840</td>
<td>2.20</td>
<td>0.1</td>
<td>2.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td>236,827</td>
<td>33,856</td>
<td>9.28</td>
<td>0.5</td>
<td>274.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>USR</td>
<td>52</td>
<td>9</td>
<td>0.57</td>
<td>0.0</td>
<td>63057.18</td>
</tr>
</tbody>
</table>

* Filtered routines marked with ‘+’

- Score report breakdown by region

```
BT-MZ Filtered Summary Measurement

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

- Submit new job

```bash
% cd bin.scorep
% cp ../jobscript/bridges/scalasca.sbatch.C.8 .
% less scalasca.sbatch.C.8

... # Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scalasca_bt-mz_${CLASS}.${PROCS}x${OMP_NUM_THREADS}.${SLURM_JOB_ID}
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=91M
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_ENABLE_TRACING=true
# Scalasca2 configuration
#export SCAN_ANALYZE_OPTS="--time-correct"
NEXUS="scalasca -analyze -t"

$NEXUS mpirun --report-bindings -np $SLURM_NTASKS $EXE

% sbatch scalasca.sbatch.C.8
```
BT-MZ Summary Analysis Report Examination

- Creates experiment directory
- The analysis report that was collated after measurement (profile.cubex)
- A trace analysis was performed after the measurement (scout.cubex)
- Post-processing with square -s (scalasca -examine -s)
BT-MZ Summary Analysis Report Examination (cont.)

- Creates experiment directory
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)

- Interactive exploration with CUBE

```
% ls scalasca_bt-mz_C.8x7.<jobid2>
scorep.filt  scorep.cfg
traces/     traces.def
profile.cubex traces.otf2
scorep.log   trace.stat
scout.cubex  scout.err
scout.log    summary.cubex
trace.cubex  scorep.score
```

```
laptop> scp userid@bridges.psc.edu:~/NPB3.3-MZ-MPI/bin.scorep/ > scalasca_bt-mz_C.8x7.<jobid2>/trace.*
```

```
laptop> cube trace.cubex
```

[cube GUI showing trace analysis report]

**Hint:** Copy ‘*.cubex’ to your laptop using ‘scp’ to improve responsiveness of GUI
Scalasca:
Reference material
Scalasca command – One command for (almost) everything

% scalasca
Scalasca 2.3.1
Toolset for scalable performance analysis of large-scale parallel applications
usage: scalasca [OPTION]... ACTION <argument>...
   1. prepare application objects and executable for measurement:
      scalasca -instrument <compile-or-link-command> # skin (using scorep)
   2. run application under control of measurement system:
      scalasca -analyze <application-launch-command> # scan
   3. interactively explore measurement analysis report:
      scalasca -examine <experiment-archive|report> # square

Options:
   -c, --show-config    show configuration summary and exit
   -h, --help           show this help and exit
   -n, --dry-run        show actions without taking them
   --quickref           show quick reference guide and exit
   --remap-specfile     show path to remapper specification file and exit
   -v, --verbose        enable verbose commentary
   -V, --version        show version information and exit

- The ‘scalasca -instrument’ command is deprecated and only provided for backwards compatibility with Scalasca 1.x., recommended: use Score-P instrumenter directly
Scalasca compatibility command: skin

% skin
Scalasca 2.3.1: application instrumenter (using Score-P instrumenter)
   -comp={all|none|...}: routines to be instrumented by compiler [default: all]
   (... custom instrumentation specification depends on compiler)
   -pdt: process source files with PDT/TAU instrumenter
   -pomp: process source files for POMP directives
   -user: enable EPIK user instrumentation API macros in source code
   -v: enable verbose commentary when instrumenting
   --*: options to pass to Score-P instrumenter

- Scalasca application instrumenter
  - Provides compatibility with Scalasca 1.x
  - **Deprecated! Use Score-P instrumenter directly.**
Scalasca convenience command: scan

% scan
Scalasca 2.3.1: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
where {options} may include:
  -h    Help: show this brief usage message and exit.
  -v    Verbose: increase verbosity.
  -n    Preview: show command(s) to be launched but don't execute.
  -q    Quiescent: execution with neither summarization nor tracing.
  -s    Summary: enable runtime summarization. [Default]
  -t    Tracing: enable trace collection and analysis.
  -a    Analyze: skip measurement to (re-)analyze an existing trace.
  -e    exptdir : Experiment archive to generate and/or analyze.
               (overrides default experiment archive title)
  -f    filtfile : File specifying measurement filter.
  -l    lockfile : File that blocks start of measurement.
  -m    metrics : Metric specification for measurement.

- Scalasca measurement collection & analysis nexus
Scalasca advanced command: scout - Scalasca automatic trace analyzer

Provided in serial (.ser), OpenMP (.omp), MPI (.mpi) and MPI+OpenMP (.hyb) variants
Scalasca advanced command: clc_synchronize

- Scalasca trace event timestamp consistency correction

  Usage: `<launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>`

  - Provided in MPI (.mpi) and MPI+OpenMP (.hyb) variants
  - Takes as input a trace experiment archive where the events may have timestamp inconsistencies
    - E.g., multi-node measurements on systems without adequately synchronized clocks on each compute node
  - Generates a new experiment archive (always called ./clc_sync) containing a trace with event timestamp inconsistencies resolved
    - E.g., suitable for detailed examination with a time-line visualizer
Scalasca convenience command: square

```bash
% square
Scalasca 2.3.1: analysis report explorer
    -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
```

- Scalasca analysis report explorer
Automatic measurement configuration

- scan configures Score-P measurement by automatically setting some environment variables and exporting them
  - E.g., experiment title, profiling/tracing mode, filter file, ...
  - Precedence order:
    - Command-line arguments
    - Environment variables already set
    - Automatically determined values

- Also, scan includes consistency checks and prevents corrupting existing experiment directories

- For tracing experiments, after trace collection completes then automatic parallel trace analysis is initiated
  - Uses identical launch configuration to that used for measurement (i.e., the same allocated compute resources)
Further information

**Scalable performance analysis of large-scale parallel applications**

- Toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid parallel applications
- Supporting most popular HPC computer systems
- Available under 3-clause BSD open-source license
- Sources, documentation & publications:
  - [http://www.scalasca.org](http://www.scalasca.org)
  - mailto: scalasca@fz-juelich.de
International HPC Summer School 2016: Analysis report examination with Cube

Christian Feld
Jülich Supercomputing Centre
Cube

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - Requires Qt4 ≥4.6 or Qt 5

- Originally developed as part of the Scalasca toolset

- Now available as a separate component
  - Can be installed independently of Score-P, e.g., on laptop or desktop
  - Latest release: Cube 4.3.4 (April 2016)
Analysis presentation and exploration

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)

- Three coupled tree browsers

- Cube displays severities
  - As value: for precise comparison
  - As color: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes
Inclusive vs. exclusive values

- **Inclusive**
  - Information of all sub-elements aggregated into single value

- **Exclusive**
  - Information cannot be subdivided further

```c
int foo()
{
    int a;
    a = 1 + 1;
    bar();
    a = a + 1;
    return a;
}
```
Analysis presentation

What kind of performance metric?

Where is it in the source code? In what context?

How is it distributed across the processes/threads?
Score-P analysis report exploration (opening view)
Metric selection

Selecting the “Time” metric shows total execution time
Expanding the system tree

Distribution of selected metric for call path by process/thread
Expanding the call tree

<table>
<thead>
<tr>
<th>Metric tree</th>
<th>Call tree</th>
<th>Flat view</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.63e9 Visits</td>
<td>0.01 MAIN</td>
<td></td>
</tr>
<tr>
<td>767.48 Time</td>
<td>0.82 mpi_setup</td>
<td></td>
</tr>
<tr>
<td>0.00 Minimum Inclusive Time</td>
<td>0.00 mpi_Bcast</td>
<td></td>
</tr>
<tr>
<td>48.58 Maximum Inclusive Time</td>
<td>0.00 env_setup</td>
<td></td>
</tr>
<tr>
<td>5.27e8 bytes_sent</td>
<td>0.00 zone_setup</td>
<td></td>
</tr>
<tr>
<td>5.27e8 bytes_recv</td>
<td>0.00 zone_starts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00 set_constants</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.02 initialize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.11 exact_rhs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00 timer_clear</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.67 exch_qbc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.04 adi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>39.91 compute_rhs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>233.49 x_solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>239.34 y_solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.07 z_solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.04 !$omp parallel @z_solve.f43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.67 !$omp do @z_solve.f52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.89 lhsinit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>57.70 binvcrhs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27.24 matvec_sub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>36.11 matmul_sub</td>
<td></td>
</tr>
</tbody>
</table>

Distribution of selected metric across the call tree

Collapsed: inclusive value
Expanded: exclusive value
### Selecting a call path

**Selection updates metric values shown in columns to the right**
Source-code view via context menu
Source-code view

/subroutine binvcrhs(lhs,c,r)/
C-----------------------------------------------
C
C
C-----------------------------------------------
implicit none

double precision pivot, coeff, lhs
dimension lhs(5,5)
double precision c(5,5), r(5)

C-----------------------------------------------
C
C-----------------------------------------------
pivot = 1.00d0/lhs(1,1)
lhs(1,2) = lhs(1,2)*pivot
lhs(1,3) = lhs(1,3)*pivot
lhs(1,4) = lhs(1,4)*pivot
lhs(1,5) = lhs(1,5)*pivot
c(1,1) = c(1,1)*pivot
c(1,2) = c(1,2)*pivot
c(1,3) = c(1,3)*pivot
c(1,4) = c(1,4)*pivot

Note: This feature depends on file and line number information provided by the instrumentation, i.e., it may not always be available.
Flat profile view

Select flat view tab, expand all nodes, and sort by exclusive value.
Box plot view

Box plot shows distribution across the system; with min/max/avg/median/quartiles
Alternative display modes

Data can be shown in various percentage modes.

![Alternative display modes](image)
Important display modes

- **Absolute**
  - Absolute value shown in seconds/bytes/counts

- **Selection percent**
  - Value shown as percentage w.r.t. the selected node “on the left” (metric/call path)

- **Peer percent (system tree only)**
  - Value shown as percentage relative to the maximum peer value
Multiple selection

Select multiple nodes with Ctrl-click
Context-sensitive help available for all GUI items.
Post-processed trace analysis report

Additional trace-based metrics in metric hierarchy
Online metric description

Access online metric description via context menu
Online metric description

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recev or MPI_Wait) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_Waitall), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:
Seconds

Diagnosis:
Try to replace MPI_Recev with a non-blocking receive MPI_Irecv that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:
MPI Point-to-point Communication Time

Children:
Critical-path analysis

Critical-path profile shows wall-clock time impact
Critical-path analysis

Critical-path imbalance highlights inefficient parallelism
Pattern instance statistics

Access pattern instance statistics via context menu

Click to get statistics details
Connect to Vampir trace browser

To investigate most severe pattern instances, connect to a trace browser...

...and select trace file from the experiment directory
Show most severe pattern instances

Select “Max severity in trace browser” from context menu of call paths marked with a red frame
Investigate most severe instance in Vampir

Vampir will automatically zoom to the worst instance in multiple steps (i.e., undo zoom provides more context)
Derived metrics

- Derived metrics are defined using CubePL expressions, e.g.:
  \[ \frac{\text{metric::time(i)}}{\text{metric::visits(e)}} \]
- Values of derived metrics are not stored, but calculated on-the-fly
- Types of derived metrics:
  - Prederived: evaluation of the CubePL expression is performed before aggregation
  - Postderived: evaluation of the CubePL expression is performed after aggregation

- Examples:
  - “Average execution time”: Postderived metric with expression
    \[ \frac{\text{metric::time(i)}}{\text{metric::visits(e)}} \]
  - “Number of FLOP per second”: Postderived metric with expression
    \[ \frac{\text{metric::FLOP()}}{\text{metric::time()}} \]
Derived metrics in Cube GUI

- Collection of derived metrics
- Parameters of the derived metric
- CubePL expression
Example: FLOPS based on PAPI_FP_OPS and time
CUBE algebra utilities

- Extracting solver sub-tree from analysis report
  ```
  % cube_cut -r '<<ITERATION>>' scorep_bt-mz_B_mic15p30x4_sum/profile.cubex
  Writing cut.cubex... done.
  ```

- Calculating difference of two reports
  ```
  % cube_diff scorep_bt-mz_B_mic15p30x4_sum/profile.cubex cut.cubex
  Writing diff.cubex... done.
  ```

- Additional utilities for merging, calculating mean, etc.
- Default output of cube_utility is a new report utility.cubex
- Further utilities for report scoring & statistics
- Run utility with `–h` (or no arguments) for brief usage info
Iteration profiling

- Show time dependent behavior by “unrolling” iterations

- Preparations:
  - Mark loop body by using Score-P instrumentation API in your source code

```c
SCOREP_USER_REGION_DEFINE( scorep_bt_loop )
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )
SCOREP_USER_REGION_END( scorep_bt_loop )
```

- Result in the Cube profile:
  - Iterations shown as separate call trees
  - Useful for checking results for specific iterations
    - or
    - Select your user-instrumented region and mark it as loop
    - Choose “Hide iterations”
    - View the Barplot statistics or the (thread x iterations) Heatmap
Iteration profiling: Barplot

Aggregation selection

Iterations
Iteration profiling: Heatmap
Cube: Further information

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - [http://www.scalasca.org](http://www.scalasca.org)
- User guide also part of installation:
  - `cube-config --cube-dir`/share/doc/CubeGuide.pdf
- Contact:
  - [mailto: scalasca@fz-juelich.de](mailto:scalasca@fz-juelich.de)
International HPC Summer School 2016: Performance analysis and optimization Tools overview

VI-HPS Team
Christian Feld – Jülich Supercomputing Centre
Virtual Institute – High Productivity Supercomputing

- **Goal**: Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems
- **Start-up funding (2006–2011)** by Helmholtz Association of German Research Centres
- **Activities**
  - Development and integration of HPC programming tools
  - Correctness checking & performance analysis
  - Academic workshops
  - Training workshops
  - Service
    - Support email lists
    - Application engagement

http://www.vi-hps.org
VI-HPS partners (founders)

Forschungszentrum Jülich
  ■ Jülich Supercomputing Centre

RWTH Aachen University
  ■ Centre for Computing & Communication

Technische Universität Dresden
  ■ Centre for Information Services & HPC

University of Tennessee (Knoxville)
  ■ Innovative Computing Laboratory
VI-HPS partners (cont.)

- Barcelona Supercomputing Center
  - Centro Nacional de Supercomputación
- Lawrence Livermore National Lab.
  - Center for Applied Scientific Computing
- Technical University of Darmstadt
  - Laboratory for Parallel Programming
- Technical University of Munich
  - Chair for Computer Architecture
- University of Oregon
  - Performance Research Laboratory
- University of Stuttgart
  - HPC Centre
- University of Versailles St-Quentin
  - LRC ITACA
- Allinea Software Ltd
Productivity tools

- MUST
  - MPI usage correctness checking
- PAPI
  - Interfacing to hardware performance counters
- Periscope Tuning Framework
  - Automatic analysis and Tuning
- Scalasca
  - Large-scale parallel performance analysis
- TAU
  - Integrated parallel performance system
- Vampir
  - Interactive graphical trace visualization & analysis
- Score-P
  - Community-developed instrumentation & measurement infrastructure

For a brief overview of tools consult the VI-HPS Tools Guide:
Productivity tools (cont.)

- **DDT/MAP/PR**: Parallel debugging, profiling & performance reports
- **Extra-P**: Automated performance modelling
- **KcacheGrind**: Callgraph-based cache analysis [x86 only]
- **MAQAO**: Assembly instrumentation & optimization [x86-64 only]
- **mpiP/mpiPview**: MPI profiling tool and analysis viewer
- **Open MPI**: Integrated memory checking
- **Open|SpeedShop**: Integrated parallel performance analysis environment
- **Paraver/Dimenas/Extrae**: Event tracing and graphical trace visualization & analysis
- **Rubik**: Process mapping generation & optimization [BG only]
- **SIONlib/Spindle**: Optimized native parallel file I/O & shared library loading
- **STAT**: Stack trace analysis tools
- **SysMon**: Batch system monitor plugin for Eclipse PTP
Non VI-HPS performance tools

- ... 

Commercial tools:
- CrayPat (Cray)
- ...
Technologies and their integration
Workshops/Tutorials

- Tuning Workshop Series
  - Three to five days *bring-your-own-code* workshops at HPC centres
  - Usually free of charge
  - [http://www.vi-hps.org/training/tws/](http://www.vi-hps.org/training/tws/)

- Tutorials at various conferences
  - E.g., SC16: Hands-on Practical Hybrid Parallel Application Performance Engineering
Performance Audits/Plans/Proof-of-concepts

- Performance Optimisation and Productivity (POP)
  - Offers performance optimisation and productivity services
  - Time-limited offer/project
  - Using VI-HPS tools
  - Funded by European Unions Horizon 2020 research and innovation programme
  - [https://pop-coe.eu/services](https://pop-coe.eu/services)

- They help you fix your code, for free!!!