SCALASCA v1.4 Quick Reference

General

- SCALASCA is an open-source toolset for scalable performance analysis of large-scale parallel applications.
- Use the **`scalasca`** command with appropriate action flags to *instrument* application object files and executables, *analyze* execution measurements, and interactively *examine* measurement/analysis experiment archives.
- For short usage explanations, use SCALASCA commands without arguments, or add '**`-v`**' for verbose commentary.

Instrumentation

- Prepend **`scalasca -instrument`** (or ***`skin`***) and any instrumentation flags to your compile/link commands.
- By default, MPI and OpenMP operations are automatically instrumented, and many compilers are also able to instrument all routines found in source files (unless explicitly disabled with **`-comp=none`**).
- To enable manual instrumentation (described on page 3) using the EPIK user instrumentation API, add **`-user`**, and/or using POMP directives, add **`-pomp`**.
- If PDToolkit is available, use **`-pdt`** and associated configuration options when additional instrumentation is desired.
- Examples:

Original command:	SCALASCA instrumentation command:
`mpicc -c foo.c`	**`scalasca -instrument`** `mpicc -c foo.c`
`mpicxx -o foo foo.cpp`	**`scalasca -inst -pomp`** `mpicxx -o foo foo.cpp`
`mpif90 -openmp -o bar bar.f90`	**`skin`** `mpif90 -openmp -o bar bar.f90`

- Often it is preferable to prefix Makefile compile/link commands with `$(PREP)` and set `PREP="scalasca -inst"` for instrumented builds (leaving `PREP` unset for uninstrumented builds).

Measurement & Analysis

- Prepend **`scalasca -analyze`** (or ***`scan`***) to the usual execution command line to perform a measurement with SCALASCA runtime summarization and associated automatic trace analysis (if applicable).
- Each measurement is stored in a new experiment archive which is never overwritten by a subsequent measurement.
- By default, only a runtime summary (profile) is collected (equivalent to specifying **`-s`**).
- To enable trace collection & analysis, add the flag **`-t`**.
- To analyze MPI and hybrid OpenMP/MPI applications, use the usual MPI launcher command and arguments.
- Examples:

Original command:	SCALASCA measurement & analysis command:	Experiment archive:
`mpiexec -np 4 foo args`	**`scalasca -analyze`** `mpiexec -np 4 foo args`	`# epik_foo_4_sum`
`OMP_NUM_THREADS=3 bar`	`OMP_NUM_THREADS=3` **`scan -t`** `bar`	`# epik_bar_Ox3_trace`
`mpiexec -np 4 foobar`	**`scan -s`** `mpiexec -np 4 foobar`	`# epik_foobar_4x3_sum`

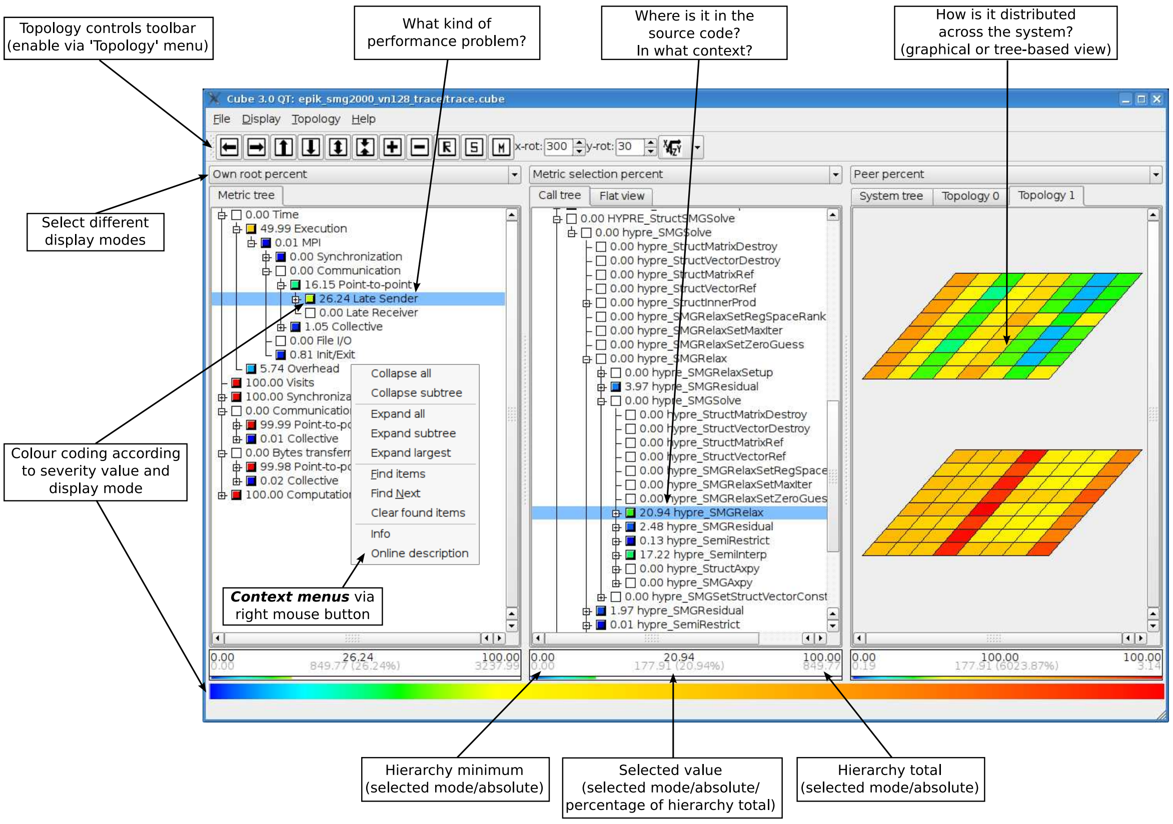
Measurement configuration

SCALASCA measurement is controlled by a number of variables which can be specified in a `EPIK.CONF` file in the working directory or through corresponding environment variables (which override values found in any configuration file): the measurement configuration is stored in the experiment archive as `epik.conf`. The most important variables are:

Variable	**Purpose**	**Default**
`EPK_TITLE`	Experiment archive title, specified by `scalasca -analyze -e epik_`*`title`* or automatically given a reasonable name if not specified.	a
`EPK_FILTER`	Name of file containing a list of names of functions (one per line) which should be ignored during the measurement (when supported).	—
`EPK_METRICS`	Colon-separated list of counter metrics or predefined groups to be measured with events (ignored if not configured with PAPI).	—
`EPK_VERBOSE`	Controls generation of additional (debugging) output by measurement system.	false
`ESD_PATHS`	Maximum number of measured call-paths.	4 096
`ESD_FRAMES`	Maximum stack frame depth of measured call-paths.	32
`ESD_BUFFER_SIZE`	Size of per-process definitions buffers in bytes.	100 000
`ELG_BUFFER_SIZE`	Size of per-thread event trace buffers in bytes.	10 000 000

Analysis Report Examination

- To interactively examine the contents of a SCALASCA experiment, after final processing of runtime summary and trace analyses, use **`scalasca -examine`** (or ***`square`***) with the experiment archive directory name as argument.
- To skip the graphical user interface and get a textual score output (using the `cube3_score` utility), add the **`-s`** flag.
- If multiple analysis reports are available, a trace analysis report is shown in preference to a runtime summary report: other reports can be specified directly or selected from the File/Open menu.
- Results are displayed using three coupled tree browsers showing
  - Metrics (i.e., performance properties/problems)
  - Call-tree or flat region profile
  - System location (alternative: graphical display of physical/virtual topologies, 1D/2D/3D Cartesian only)


Topology controls toolbar (enable via 'Topology' menu)
What kind of performance problem?
Where is it in the source code? In what context?
How is it distributed across the system? (graphical or tree-based view)
Select different display modes
Colour coding according to severity value and display mode
Context menus via right mouse button
Hierarchy minimum (selected mode/absolute)
Selected value (selected mode/absolute/ percentage of hierarchy total)
Hierarchy total (selected mode/absolute)
Cube 3.0 QT: epik_smg2000_vn128_trace/trace.cube
File Display Topology Help
x-rot: 300 y-rot: 30
Own root percent
Metric selection percent
Peer percent
Metric tree
Call tree
Flat view
System tree
Topology 0
Topology 1
Collapse all
Collapse subtree
Expand all
Expand subtree
Expand largest
Find items
Find Next
Clear found items
Info
Online description

- Analyses are presented in trees, where collapsed nodes represent *inclusive* values (consisting of the value of the node itself and all of its child nodes), which can be selectively expanded to reveal *exclusive* values (i.e., the node 'self' value) and child nodes.
- When a node is selected from any tree, its *severity* value (and percentage) are shown in the panel below it, and that value distributed across the tree(s) to the right of it.
- Selective expansion of critical nodes, guided by the colour scale, can be used to hone in on performance problems.
- Each tree browser provides additional information via a context menu (on the right mouse button), such as the description of the selected metric or source code for the selected region (where available).
- Metric severity values can be displayed in various modes:

Mode	**Description**
Absolute	Absolute value in the corresponding unit of measurement.
Root percent	Percentage relative to the inclusive value of the root node of the corresponding hierarchy.
Selection percent	Percentage relative to the value of selected node in corresponding tree browser to the left.
Peer percent	Percentage relative to the maximum of all peer values (all values of the current leaf level).
Peer distribution	Percentage relative to the maximum and non-zero minimum of all peer values.
External percent	Similar to "Root percent," but reference values are taken from another experiment.

Manual source-code instrumentation

- Region or phase annotations manually inserted in source files can augment or substitute automatic instrumentation, and can improve the structure of analysis reports to make them more readily comprehensible.
- These annotations can be used to mark any sequence or block of statements, such as functions, phases, loop nests, etc., and can be nested, provided that every enter has a matching exit.
- If automatic compiler instrumentation is not used (or not available), it is typically desirable to manually instrument at least the `main` function/program and perhaps its major phases (e.g., initialization, core/body, finalization).

EPIK user instrumentation API

C/C++:

```
#include "epik_user.h"
...
void foo() {
   ...  // local declarations
   ...  // more declarations
   EPIK_FUNC_START();
   ...  // executable statements
   if (...)  {
     EPIK_FUNC_END();
     return;
   } else {
     EPIK_USER_REG(r_name,"region");
     EPIK_USER_START(r_name);
     ...
     EPIK_USER_END(r_name);
   }
   ...  // executable statements
   EPIK_FUNC_END();
   return;
}
```

Fortran:

```
#include "epik_user.inc"
...
subroutine bar()
   EPIK_FUNC_REG("bar")
   EPIK_USER_REG(r_name,"region")
   ...  ! local declarations
   EPIK_FUNC_START()
   ...  ! executable statements
   if (...)  then
     EPIK_FUNC_END()
     return
   else
     EPIK_USER_START(r_name)
     ...
     EPIK_USER_END(r_name)
   endif
   ...  ! executable statements
   EPIK_FUNC_END()
   return
end subroutine bar
```

C++:

```
#include "epik_user.h"
...
{
   EPIK_TRACER("name");
   ...
}
```

- `EPIK_FUNC_START` and `EPIK_FUNC_END` are provided explicitly to mark the entry and exit(s) of functions/subroutines.
- Function names are automatically provided by C/C++, however, in annotated Fortran functions/subroutines an appropriate name should be registered with `EPIK_FUNC_REG("func_name")` in its prologue.
- Region identifiers (e.g., `r_name`) should be registered with `EPIK_USER_REG` in each annotated prologue before use with `EPIK_USER_START` and `EPIK_USER_END` in the associated body.
- Every exit/break/continue/return/etc. out of each annotated region must have corresponding `_END()` annotation(s).
- Source files annotated in this way need to be compiled with the `-user` flag given to the SCALASCA instrumenter, otherwise the annotations are ignored. Fortran source files need to be preprocessed (e.g., by CPP).

POMP user instrumentation API

POMP annotations provide a mechanism for preprocessors (such as OPARI2) to conditionally insert user instrumentation.

C/C++:

```
#pragma pomp inst init // once only, in main
...
#pragma pomp inst begin(name)
   ...
   [ #pragma pomp inst altend(name) ]
   ...
#pragma pomp inst end(name)
```

Fortran:

```
!POMP$ INST INIT ! once only, in main program
...
!POMP$ INST BEGIN(name)
   ...
   [ !POMP$ INST ALTEND(name) ]
   ...
!POMP$ INST END(name)
```

- Every intermediate exit/break/return/etc. from each annotated region must have an `altend` or `ALTEND` annotation.
- Source files annotated in this way need to be processed with the `-pomp` flag given to the SCALASCA instrumenter, otherwise the annotations are ignored.

EPIK experiment archives

SCALASCA measurement and analysis artifacts are stored in unique experiment archive directories, with the `epik_` prefix, which can be handled with standard Unix tools.

- The SCALASCA measurement & analysis nexus automatically generates a default experiment title from the target executable, compute node mode (if appropriate), number of MPI processes (or `0` if omitted), number of OpenMP threads (if `OMP_NUM_THREADS` is set), summarization or tracing mode, and optional metric specification, e.g.,

```
% OMP_NUM_THREADS=4 scan -t -m BGP_TORUS mpiexec -mode SMP -np 512 /path/foobar args
→ epik_foobar_smp512x4_trace_BGP_TORUS
```

- An existing measurement archive will block new measurements that would have the same experiment archive name. Existing experiment archives are therefore not overwritten or otherwise corrupted by subsequent measurements.
- An archive directory name can be explicitly specified with **`scan -e epik_title`** (or `EPK_TITLE=title`).

Typical experiment archive contents

File	Description
`epik.conf`	Measurement configuration when the experiment was collected.
`epik.filt`	Measurement filter specified when the experiment was collected.
`epik.log`	Output of the instrumented program and measurement system.
`epik.path`	Callpath-tree recorded by the measurement system.
`epitome.cube`	† Intermediate analysis report of the runtime summarization system.
`scout.log`	Output of the parallel trace analyzer.
`scout.cube`	† Intermediate analysis report of the parallel trace analyzer.
`summary.cube[.gz]`	Post-processed analysis report of runtime summarization. (May include HWC metrics.)
`trace.cube.[gz]`	Post-processed analysis report of the trace analyzer. (Does not include HWC metrics.)
`trace+HWC.cube.[gz]`	Combined summary and trace analysis report including hardware counter metrics.
`ELG/`	† Sub-directory containing EPILOG event traces for each process.
`epik.esd`	† Unified definitions for a set of EPILOG event traces. (Open this file with Vampir.)
`epik.map`	† Definition mappings for a set of EPILOG event traces.

† Intermediate analysis reports, the `ELG` subdirectory and associated EPILOG files can be deleted after final analysis reports have been generated. *Note: These files can be extremely large, especially when hardware counter metrics are measured!*

CUBE3 algebra and utilities

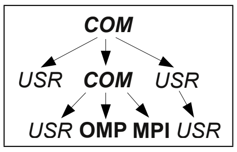
Uniform behavioural encoding, processing and interactive examination of parallel application execution analysis reports.

- CUBE3 provides a variety of utilities for differencing, combining and other operations on analysis reports, e.g., `cube3_diff`, `cube3_mean`, `cube3_merge`.
- `cube3_cut` can be used to prune uninteresting call-trees and/or re-root with a specified call-tree node.
- `cube3_stat` can be used to produce custom statistical reports in CSV or plain text format.
- `cube3_topoassist` can be used to add or modify topology specifications.

Determining trace buffer capacity requirements

Based on an analysis report, the required trace buffer capacity can be estimated using

```
cube3_score [-r] [-f filter_file] experiment_archive/summary.cube
```



- To get detailed information per region (i.e., function or subroutine), use `-r`
- To take a proposed filter file into account, use `-f` *filter_file*
- The reported value `max_tbc` specifies the maximum estimated trace buffer content per process, which can be used to set `ELG_BUFFER_SIZE` appropriately to avoid intermediate flushes in subsequent tracing experiments.

Tips for effective use of the SCALASCA toolset

1. Determine one or more repeatable execution configurations (input data, number of processes/threads) and time their overall execution to have a baseline for reference. (If possible, also identify maximum memory requirements.)
   - Ensure that the execution terminates cleanly, e.g., with `MPI_Finalize`.
   - Excessively long execution durations can make measurement and analysis inconvenient, therefore the test configuration shouldn't be longer than sufficient to be representative.
2. Modify the application build procedure (Makefile) to prepend the SCALASCA instrumenter to compile and link commands, and produce an instrumented executable.
   - MPI library calls and OpenMP parallel regions will be instrumented by default, along with user functions if supported by the compiler.
   - Serial libraries and source modules using neither MPI nor OpenMP are generally not worth instrumenting with SCALASCA, and indeed may result in undesirable measurement overheads.
3. Prefix the usual launch/run command with the SCALASCA analyzer to run the instrumented executable under control of the SCALASCA measurement collection and analysis nexus to produce an experiment archive directory.
   - By default the experiment archive is produced in the current working directory, and its name will start with '`epik_`' followed by some configuration descriptors.
   - If a similarly configured experiment has already been run and its archive directory blocks new measurement experiments, rename (or remove) the old archive.
   - A call-path profile summary report containing Time and Visits metrics (and when appropriate also MPI file I/O and message statistics and hardware counters) for each process/thread is produced by default.
   - If the (default) measurement configuration is inadequate for a complete measurement to be collected, EPIK warnings will indicate that one or more configuration variables should be adjusted (e.g., `ESD_BUFFER_SIZE`).
   - Compare the runtime to the (uninstrumented) reference to estimate implicit instrumentation dilation overhead.
4. Use the SCALASCA examiner to explore the analysis report in the experiment archive.
   - If there are 'Unknown' callpaths reported, this indicates that the measurement configuration should be adjusted to allocate additional capacity for storing callpath measurements (`ESD_PATHS`).
   - Uninstrumented or filtered routines will not appear in the analysis report, and their associated metric severities will be attributed to the last measured routine from which they are called (as if they were 'inlined').
   - OpenMP teams will only have measurements for `ESD_MAX_THREADS` threads (default: `OMP_NUM_THREADS`).
   - Additional structure can be included in the analysis report by using the EPIK user instrumentation API to specify (nested) regions or phases as annotations in the source code.
5. Score the quality of the summary analysis report (particularly if dilation is significant), adjust measurement configuration using a filter file, adjust OpenMP instrumentation, or selectively instrument source modules (or routines).
   - OpenMP synchronization constructs that have prohibitively large measurement costs should not be instrumented, and instrumentation of these constructs can be disabled. [See OPEN_ISSUES/OPARI2.]
   - Investigate use of a filter file specifying instrumented routines (one per line, using shell wildcards) to be ignored during measurement collection.
   - Routines with very high visit counts and relatively low total times (which are not MPI functions and OpenMP parallel regions) are appropriate candidates for filtering, and can be identified from the flat profile view in the GUI, or score reports generated with '`scalasca -examine -s`' or using '`cube3_score -r`'.
   - Highly-recursive functions are typically also worth removing: recursion is often indicated by a large maximum frame depth and the culprits are obvious by examining `epik.path` in the experiment archive.
   - A prospective filter file can be specified to scoring with '`-f`' for evaluation prior to being used to re-do measurement and re-check dilation.
   - Some routines might still present excessive overhead even when filtered, and these should not be instrumented. The build procedure may need to be adjusted not to prefix the SCALASCA instrumenter when compiling the associated source modules. When SCALASCA is configured with PDToolkit, it can be used to selectively instrument entire source modules or individual routines (see PDToolkit documentation for details).

6. Use scoring on the (revised/filtered) summary analysis report to determine an appropriate size for trace buffers.
   - The maximum trace buffer content (`max_tbc`) estimated for all events of any type can be used to specify trace buffer sizes via the `ELG_BUFFER_SIZE` configuration variable.
   - This value (in bytes) specifies the amount of memory to be allocated by each thread and the size of trace file potentially written by *each* thread. The trace file sizes on disk may be somewhat less when compression is used when writing, and many times larger if intermediate flushes are necessary during measurement.
   - `ELG_BUFFER_SIZE` should be set after consideration of the memory available when measuring the instrumented application execution and the system's I/O and filesystem performance and capacity. (These vary enormously from system to system and can quickly be overwhelmed by large traces!)
   - Additional user routines can be included in a filter file to reduce trace buffer requirements.
7. Repeat measurement specifying the '`-t`' flag to the SCALASCA analyzer (along with other configuration settings if necessary) to collect and automatically analyze execution traces.
   - Traces are generally written directly into the experiment archive to avoid copying at completion. If a temporary location is specified with `EPK_LDIR` ensure that it contains sufficient capacity.
   - If there are EPIK messages reporting trace flushing to disk prior to closing the experiment, these intermediate flushes are often highly disruptive. These also show up as `TRACING` Overhead during measurement outside of measurement initialization and finalization (e.g., during `MPI_init` and `MPI_Finalize`). Enlarging trace buffer sizes and/or adjusting instrumentation or the measurement filter and/or configuring a shorter execution (perhaps with fewer iterations or timesteps) may be appropriate.
   - Parallel trace analysis requires several times as much memory as the size of the respective (uncompressed process) traces, and it is currently not possible to analyze incomplete traces. When memory is restricted, trace sizes should be reduced accordingly.
   - If clock synchronization inconsistencies are reported during trace analysis, specify `SCAN_ANALYZE_OPTS=-s` to incorporate a logical clock correction step during analysis.
   - In order to turn on the worst instance tracking option, `SCAN_ANALYZE_OPTS=-i` needs to be specified.
   - Traces from hybrid OpenMP/MPI application executions are analyzed in parallel by default. If an OpenMP-aware trace analyzer is not available, metrics are only calculated for the master thread of OpenMP teams. If the traces are sufficiently small, they may be merged and analyzed with the EXPERT analyzer to generate a report including additional OpenMP metrics.
   - After the analysis report has been examined and verified to be complete, it is generally unnecessary to keep the often extremely large trace files used to generate it (unless further analysis or convertion is planned): these are in the `ELG` subdirectory of a trace experiment archive, which can be deleted.
8. In addition to interactive exploration of analysis reports with the SCALASCA examiner, they can be processed with a variety of CUBE algebra tools and utilities.
9. If you encounter difficulties using SCALASCA to instrument applications, configuring measurement collection and analysis, or interpreting analysis reports, contact `scalasca@fz-juelich.de` for assistance.

KOJAK

An open source kit for objective judgement and knowledge-based detection of performance bottlenecks.

- KOJAK is no longer released separately, but the version included with this distribution includes the same functionality and commands of its predecessors using the EPIK measurement system and CUBE3 report format and viewer.
- EPILOG event trace files can be generated by EPIK, though this is inactive by default: set configuration/environment variable `EPK_TRACING=1` to activate tracing (and optionally `EPK_SUMMARY=0` to deactivate runtime summarization).
- Trace files and other experiment artifacts are stored in EPIK experiment archive directories, and the commands `elg_merge`, `expert` and `kanal` also accept experiment archive directories as arguments.
- EXPERT trace analysis reports have different OpenMP metrics and do not include MPI message and I/O statistics.
- Run the `expert` command with no arguments for usage information for generating pattern statistics and traces.
- Merged KOJAK traces (e.g., `epik_a/epik.elg`) can be converted from EPILOG format to OTF, VTF3 and PARAVER formats, for use with external analysis and visualization tools.