# SCALASCA v1.0 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.
- The default uses the capability of (most) compilers to automatically instrument functions during compilation.
- To disable compiler instrumentation and enable manual instrumentation using POMP directives, add **`-pomp`**.
- To additionally enable manual instrumentation using the EPIK user instrumentation API, add **`-user`**.
- 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 SCALASCA measurement with 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_0x3_trace` |
| `mpiexec -np 4 foobar` | **`scan -s`** `mpiexec -np 4 foobar` | `# epik_foobar_4x3_sum` |

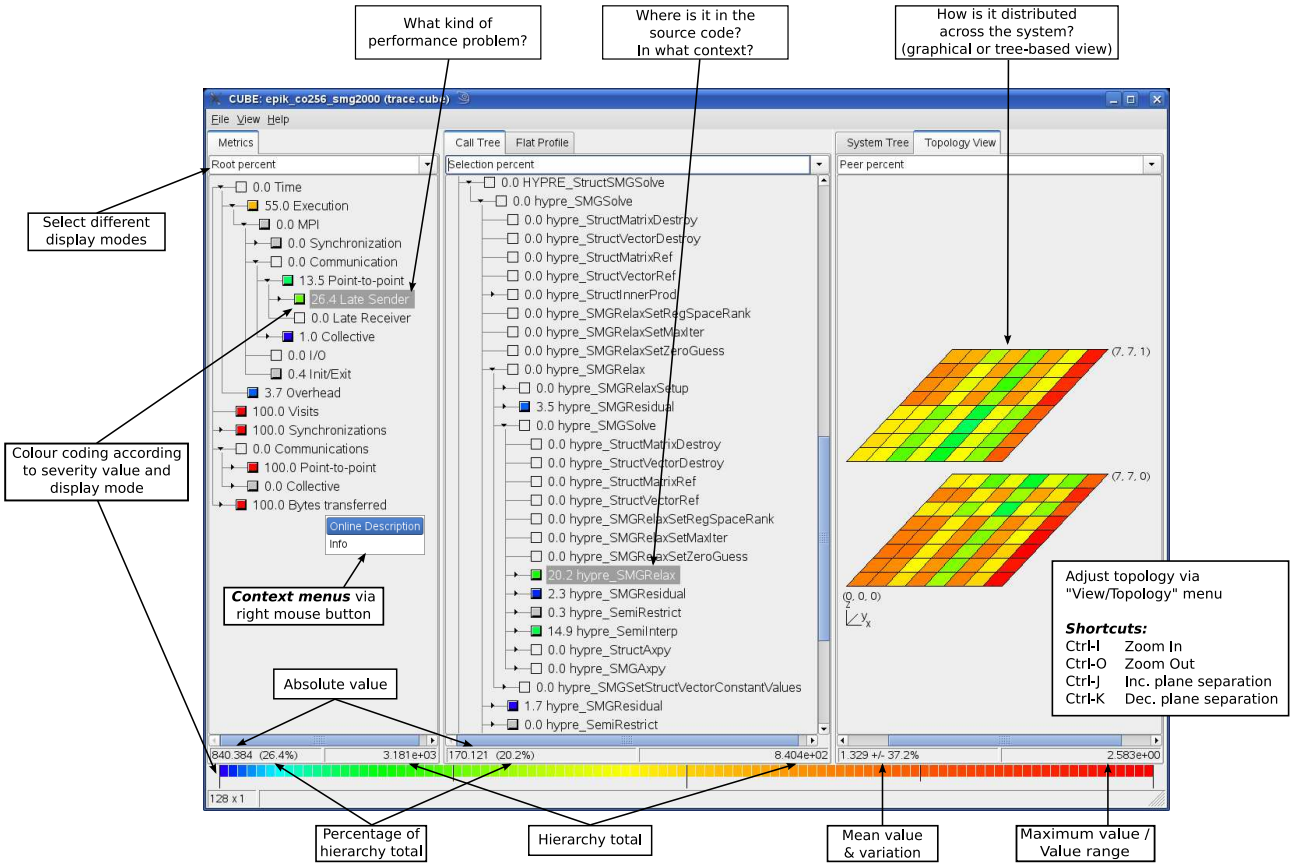
### Measurement configuration

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 (not yet available for all compilers). | — |
| `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-process 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.
- 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)




- 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 the selected node in the 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")
   ...  ! local declarations
   EPIK_FUNC_START()
   ...  ! executable statements
   if (...)  then
     EPIK_FUNC_END()
     return
   else
     EPIK_USER_REG(r_name,"region")
     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 OPARI) 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.

- Experiment archives are not overwritten or otherwise corrupted by subsequent measurements.
- An existing measurement archive will block new measurements that would have the same experiment archive name.
- 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 1024 /path/foobar args
  → epik_foobar_s1024x4_trace_BGP_TORUS
  ```

- 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. |
| `expert.log` | Output of the serial trace analyzer. |
| `expert.cube` | Analysis report of the serial trace analyzer. (Includes measured hardware counter metrics.) |
| `scout.log` | Output of the parallel trace analyzer. |
| `scout.cube` | † Intermediate analysis report of the parallel trace analyzer. |
| `summary.cube` | Post-processed analysis report of the runtime summarization. (Includes hardware counter metrics.) |
| `trace.cube` | Post-processed analysis report of the parallel trace analyzer. (Does not include HWC metrics.) |
| `ELG/` | † Directory containing EPILOG event traces for each process. |
| `epik.elg` | † Merged EPILOG event trace for serial analysis. |
| `epik.esd` | † Unified definitions for a set of EPILOG event traces. |
| `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 includes most of the same functionality and commands of its predecessors, and CUBE3 tools should be able to read files in CUBE2 format (though output is only in CUBE3 format).
- A new `cube3_cut` utility can be used to prune uninteresting call-trees and/or re-root with a specified call-tree node.
- Integrated holistic analysis reports (as formerly produced by `cube_merge`) are currently not supported.

### Determining trace buffer requirements

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

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

- To get detailed information per region (i.e., function), 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.

SCALASCA 1.0 fully integrates KOJAK 3.0, CUBE 3.0, OPARI, etc.

## KOJAK 3.0

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

- This version of KOJAK includes the same functionality and commands of its predecessors, however, it has migrated to 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 now stored in EPIK experiment archive directories, and the commands `elg_merge`, `expert` and `kanal` now also accept experiment archive directories as arguments.
- EXPERT trace analysis reports do not include MPI message statistics, but provide more comprehensive OpenMP analysis than SCALASCA 1.0.
- 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.