

CubeGUI 4.8 | User Guide

Introduction in Cube GUI and its usage

April 2024
The Scalasca Development Team
scalasca@fz-juelich.de

Attention

The Cube GUI User Guide is currently being rewritten and still incomplete. However, it should already contain enough information to get you started and avoid the most common pitfalls.

Contents

1	Copyright	1
2	Cube User Guide	3
2.1	Abstract	3
2.2	Introduction	3
2.3	Command line options	4
2.4	Environment variables	5
2.5	Using the Display	6
2.5.1	Basic Principles	6
2.5.2	GUI Components	9
2.5.2.1	Menu Bar	9
2.5.2.2	Value modes	14
2.5.2.3	System resource subsets	16
2.5.2.4	Tree browsers	16
2.5.2.5	Selected value info	22
2.5.2.6	Color legend	22
2.5.2.7	Status Bar	22
2.6	Client-Server	23
2.6.1	Cube Server	23
2.6.2	Cube Client	23
2.7	Cube GUI Plugins	23
2.7.1	Detach Plugin Tabs	24
2.7.2	Context free plugins	24
2.7.2.1	Plugin "Diff"	24
2.7.2.2	Plugin "Mean"	25
2.7.2.3	Plugin "Merge"	25
2.7.2.4	Plugin "Scaling"	25
2.7.2.5	Plugin "Tau2Cube"	25
2.7.2.6	Plugin "Measurement"	26
2.7.3	Tree Item Marker	26
2.7.4	Advanced Color Map Plugin	27
2.7.5	Metric Editor Plugin	29
2.7.6	Metric Identification Plugin	31
2.7.7	Score-P Configuration Plugin	32
2.7.8	Source Code Viewer	32
2.7.8.1	Source Code Viewer Keyboard control	33
2.7.9	System Barplot Plugin	34
2.7.9.1	Basic Principles	35
2.7.9.2	Toolbar	36
2.7.9.3	Menu Bar	37
2.7.10	System Heatmap Plugin	38
2.7.10.1	Basic Principles	38

2.7.10.2Menu Heatmap	39
2.7.11System Statistics Plugin	40
2.7.12System Sunburst Plugin	41
2.7.13System Topology Plugin	43
2.7.13.1Topology mapping panel	45
2.7.13.2Topology plugin menu	46
2.7.13.3Toolbar	47
2.7.13.4Topology keyboard and mouse control	48
2.7.14Tree Item Marker Plugin	48
2.7.15Launch Plugin	49
2.7.15.1.launch File	49
2.8 Other Features	51
2.8.1 Features enabled through statistic files	51
2.8.2 Statistical information about performance patterns	51
2.8.3 Display of most severe pattern instances using a trace browser	53
2.8.3.1 Troubleshooting	54
2.8.4 Synchronization of several cube instances	55
2.9 Keyboard and mouse control	56
3 Cube Advisor Plugin	59
3.1 Getting Started with Advisor	59
3.2 Supported Assessments	59
3.2.1 Only-MPI Assessment	59
3.2.2 Multiplicative Hybrid Assessment	60
3.2.3 Additive Hybrid Assessment	61
3.2.4 BSC Hybrid Assessment	61
3.2.5 JSC Hybrid Assessment	61
4 AdvisorPOPHybridTestsParallel_efficiency	63
4.1 Parallel Efficiency	63
5 AdvisorPOPHybridTestsMissing_parallel_efficiency	65
5.1 Missing Parallel Efficiency?	65
6 AdvisorPOPHybridTestsProcess_efficiency	67
6.1 Process Efficiency	67
7 AdvisorPOPHybridTestsMissing_process_efficiency	69
7.1 Missing Process Efficiency?	69
8 AdvisorPOPHybridTestsLoad_balance	71
8.1 Computation Load Balance	71
9 AdvisorPOPHybridTestsMissing_load_balance	73
9.1 Missing Computation Load Balance?	73
10AdvisorPOPHybridTestsCommunication_efficiency	75
10.1 MPI Communication Efficiency	75
11AdvisorPOPHybridTestsMissing_communication_efficiency	77
11.1 Missing Communication Efficiency?	77

12 AdvisorPOPHybridTestsSerialisation_efficiency	79
12.1 Serialisation Efficiency	79
13 AdvisorPOPHybridTestsMissing_serialisation_efficiency	81
13.1 Missing Serialisation Efficiency?	81
14 AdvisorPOPHybridTestsTransfer_efficiency	83
14.1 Transfer Efficiency	83
15 AdvisorPOPHybridTestsMissing_transfer_efficiency	85
15.1 Missing Transfer Efficiency?	85
16 AdvisorPOPHybridTestsThread_efficiency	87
16.1 Thread Efficiency	87
17 AdvisorPOPHybridTestsMissing_thread_efficiency	89
17.1 Missing Thread Efficiency?	89
17.2 Missing Amdahl's Efficiency?	89
18 AdvisorPOPHybridTestsAmdahl_efficiency	91
18.1 Amdahl's Efficiency	91
19 AdvisorPOPHybridTestsOmpRegion_efficiency	93
19.1 OpenMP Region Efficiency	93
20 AdvisorPOPHybridTestsMissing_omp_region_efficiency	95
20.1 Missing OpenMP Region Efficiency?	95
21 AdvisorPOPHybridTestsIpc	97
21.1 IPC (only computation)	97
22 AdvisorPOPHybridTestsMissing_ipc	99
22.1 Missing IPC?	99
23 AdvisorPOPHybridTestsStalled_resources	101
23.1 Stalled resources (only computation)	101
24 AdvisorPOPHybridTestsMissing_stalled_resources	103
24.1 Missing "Resource stall cycles"?	103
25 AdvisorPOPHybridTestsNoWaitINS_efficiency	105
25.1 Instructions (only computation)	105
26 AdvisorPOPHybridTestsMissingNoWaitINS_efficiency	107
26.1 Missing Instructions (only computation)?	107
27 AdvisorPOPHybridTestsComputationTime	109
27.1 Computation time	109
28 AdvisorPOPHybridTestsMissingComputationTime	111
28.1 Missing Computation time?	111
29 AdvisorPOPHybridAddTestsParallel_efficiency	113
29.1 Parallel Efficiency	113

30 AdvisorPOPHybridAddTestsMissing_parallel_efficiency	115
30.1 Missing Parallel Efficiency?	115
31 AdvisorPOPHybridAddTestsProcess_efficiency	117
31.1 Process Efficiency	117
32 AdvisorPOPHybridAddTestsMissing_process_efficiency	119
32.1 Missing Process Efficiency?	119
33 AdvisorPOPHybridAddTestsCommunication_efficiency	121
33.1 MPI Communication Efficiency	121
34 AdvisorPOPHybridAddTestsMissing_communication_efficiency	123
34.1 Missing Communication Efficiency?	123
35 AdvisorPOPHybridAddTestsSerialisation_efficiency	125
35.1 Serialisation Efficiency	125
36 AdvisorPOPHybridAddTestsMissing_serialisation_efficiency	127
36.1 Missing Serialisation Efficiency?	127
37 AdvisorPOPHybridAddTestsTransfer_efficiency	129
37.1 Transfer Efficiency	129
38 AdvisorPOPHybridAddTestsMissing_transfer_efficiency	131
38.1 Missing Transfer Efficiency?	131
39 AdvisorPOPHybridAddTestsLoad_balance	133
39.1 Computation Load Balance	133
40 AdvisorPOPHybridAddTestsMissing_load_balance	135
40.1 Missing Computation Load Balance?	135
41 AdvisorPOPHybridAddTestsThread_efficiency	137
41.1 Thread Efficiency	137
42 AdvisorPOPHybridAddTestsMissing_thread_efficiency	139
42.1 Missing Thread Efficiency?	139
42.2 Missing Amdahl's Efficiency?	139
43 AdvisorPOPHybridAddTestsAmdahl_efficiency	141
43.1 Amdahl's Efficiency	141
44 AdvisorPOPHybridAddTestsOmpRegion_efficiency	143
44.1 OpenMP Region Efficiency	143
45 AdvisorPOPHybridAddTestsMissing_omp_region_efficiency	145
45.1 Missing OpenMP Region Efficiency?	145
46 AdvisorPOPHybridAddTestsIpc	147
46.1 IPC (only computation)	147
47 AdvisorPOPHybridAddTestsMissing_ipc	149
47.1 Missing IPC?	149

48 AdvisorPOPHybridAddTestsStalled_resources	151
48.1 Stalled resources (only computation)	151
49 AdvisorPOPHybridAddTestsMissing_stalled_resources	153
49.1 Missing Stalled resources?	153
50 AdvisorPOPHybridAddTestsNoWaitINS_efficiency	155
50.1 Instructions (only computation)	155
51 AdvisorPOPHybridAddTestsMissingNoWaitINS_efficiency	157
51.1 Missing Instructions (only computation)?	157
52 AdvisorPOPHybridAddTestsComputationTime	159
52.1 Computation time	159
53 AdvisorPOPHybridAddTestsMissingComputationTime	161
53.1 Missing Computation time?	161
54 AdvisorBSPOPHybridTestsParallel_efficiency	163
54.1 Hybrid Parallel Efficiency	163
55 AdvisorBSPOPHybridTestsMissing_parallel_efficiency	165
55.1 Missing Hybrid Parallel Efficiency?	165
56 AdvisorBSPOPHybridTestsLoadBalance_efficiency	167
56.1 Hybrid Load Balance Efficiency	167
57 AdvisorBSPOPHybridTestsMissing_loadbalance_efficiency	169
57.1 Missing Hybrid Load Balance Efficiency?	169
58 AdvisorBSPOPHybridTestsCommunication_efficiency	171
58.1 Hybrid Communication Efficiency	171
59 AdvisorBSPOPHybridTestsMissing_communication_efficiency	173
59.1 Missing Hybrid Communication Efficiency?	173
60 AdvisorBSPOPHybridTestsMPIParallel_efficiency	175
60.1 MPI Parallel Efficiency	175
61 AdvisorBSPOPHybridTestsMissing_MPIparallel_efficiency	177
61.1 Missing MPI Parallel Efficiency?	177
62 AdvisorBSPOPHybridTestsMPILoad_balance_efficiency	179
62.1 MPI Load Balance Efficiency	179
63 AdvisorBSPOPHybridTestsMissing_MPILoad_balance_efficiency	181
63.1 Missing MPI Load Balance Efficiency?	181
64 AdvisorBSPOPHybridTestsMPICommunication_efficiency	183
64.1 MPI Communication Efficiency	183
65 AdvisorBSPOPHybridTestsMissing_MPIcommunication_efficiency	185
65.1 Missing MPI Communication Efficiency?	185

66 AdvisorBSPOPHybridTestsOMPParallel_efficiency	187
66.1 OpenMP Parallel Efficiency	187
67 AdvisorBSPOPHybridTestsMissing_OMPparallel_efficiency	189
67.1 Missing OpenMP Parallel Efficiency?	189
68 AdvisorBSPOPHybridTestsOMPLoadBalance_efficiency	191
68.1 OpenMP Load Balance Efficiency	191
69 AdvisorBSPOPHybridTestsMissing_OMPloadbalance_efficiency	193
69.1 Missing OpenMP Load Balance Efficiency?	193
70 AdvisorBSPOPHybridTestsOMPCommunication_efficiency	195
70.1 OpenMP Communication Efficiency	195
71 AdvisorBSPOPHybridTestsMissing_OMPcommunication_efficiency	197
71.1 Missing OpenMP Communication Efficiency?	197
72 AdvisorBSPOPHybridTestsIpc	199
72.1 IPC (only computation)	199
73 AdvisorBSPOPHybridTestsMissing_ipc	201
73.1 Missing IPC?	201
74 AdvisorBSPOPHybridTestsStalled_resources	203
74.1 Stalled resources (only computation)	203
75 AdvisorBSPOPHybridTestsMissing_stalled_resources	205
75.1 Missing Stalled resources?	205
76 AdvisorBSPOPHybridTestsNoWaitINS_efficiency	207
76.1 Instructions (only computation)	207
77 AdvisorBSPOPHybridTestsMissingNoWaitINS_efficiency	209
77.1 Missing Instructions (only computation)?	209
78 AdvisorBSPOPHybridTestsComputationTime	211
78.1 Computation time	211
79 AdvisorBSPOPHybridTestsMissingComputationTime	213
79.1 Missing Computation time?	213
80 AdvisorJSCTestsLoad_balance	215
80.1 MPI computation Load Balance	215
81 AdvisorJSCTestsMissing_load_balance	217
81.1 Missing MPI computation Load Balance?	217
82 AdvisorJSCTestsCommunication_efficiency	219
82.1 MPI communication Efficiency	219
83 AdvisorJSCTestsMissing_communication_efficiency	221
83.1 Missing MPI communication Efficiency?	221

84 AdvisorJSTestsSerialisation_efficiency	223
84.1 Serialisation Efficiency	223
85 AdvisorJSTestsMissing_serialisation_efficiency	225
85.1 Missing Serialisation Efficiency?	225
86 AdvisorJSTestsTransfer_efficiency	227
86.1 Transfer Efficiency	227
87 AdvisorJSTestsMissing_transfer_efficiency	229
87.1 Missing Transfer Efficiency?	229
88 AdvisorJSTestsAmdahl_efficiency	231
88.1 OpenMP Amdahl's Efficiency	231
89 AdvisorJSTestsMissingAmdahl_efficiency	233
89.1 Missing OpenMP Amdahl's Efficiency?	233
90 AdvisorJSTestsOmpLoad_balance	235
90.1 OpenMP Load Balance Efficiency	235
91 AdvisorJSTestsMissing_omp_load_balance	237
91.1 Missing OpenMP Load Balance Efficiency?	237
92 AdvisorJSTestsOmpSerialisation_efficiency	239
92.1 OpenMP Serialisation Efficiency	239
93 AdvisorJSTestsMissing_omp_serialisation_efficiency	241
93.1 Missing OpenMP Serialisation Efficiency?	241
94 AdvisorJSTestsIpc	243
94.1 IPC (only computation)	243
95 AdvisorJSTestsMissing_ipc	245
95.1 Missing IPC?	245
96 AdvisorJSTestsStalled_resources	247
96.1 Stalled resources (only computation)	247
97 AdvisorJSTestsMissing_stalled_resources	249
97.1 Missing Stalled resources?	249
98 AdvisorJSTestsNoWaitINS_efficiency	251
98.1 Instructions (only computation)	251
99 AdvisorJSTestsMissingNoWaitINS_efficiency	253
99.1 Missing Instructions (only computation)?	253
100 AdvisorJSTestsComputationTime	255
100. Computation time	255
101 AdvisorJSTestsMissingComputationTime	257
101. Missing Computation time?	257

10AdvisorPOPTestsParallel_efficiency	259
102.Parallel Efficiency	259
10AdvisorPOPTestsMissing_parallel_efficiency	261
103.Missing Parallel Efficiency?	261
10AdvisorPOPTestsLoad_balance	263
104.Load Balance	263
10AdvisorPOPTestsMissing_load_balance	265
105.Missing Load Balance?	265
10AdvisorPOPTestsCommunication_efficiency	267
106.Communication Efficiency	267
10AdvisorPOPTestsMissing_communication_efficiency	269
107.Missing Communication Efficiency?	269
10AdvisorPOPTestsSerialisation_efficiency	271
108.Serialisation Efficiency	271
10AdvisorPOPTestsMissing_serialisation_efficiency	273
109.Missing Serialisation Efficiency?	273
11AdvisorPOPTestsTransfer_efficiency	275
110.Transfer Efficiency	275
11AdvisorPOPTestsMissing_transfer_efficiency	277
111.Missing Transfer Efficiency?	277
11AdvisorPOPTestsIpc	279
112.IPC (only computation)	279
11AdvisorPOPTestsStalled_resources	281
113.Stalled resources (only computation)	281
11AdvisorPOPTestsMissing_stalled_resources	283
114.Missing Stalled resources?	283
11AdvisorPOPTestsNoWaitINS_efficiency	285
115.Instructions (only computation)	285
11AdvisorPOPTestsMissingNoWaitINS_efficiency	287
116.Missing Instructions (only computation)?	287
11AdvisorPOPTestsComputationTime	289
117.Computation time	289
11AdvisorPOPTestsMissingComputationTime	291
118.Missing Computation time?	291
11AdvisorPOPComputation_efficiency	293
119.Computation Efficiency	293

120	AdvisorPOPInstruction_efficiency	295
120.	Instruction Efficiency	295
121	AdvisorPOPTestsIpc_efficiency	297
121.	IPC Efficiency	297
122	AdvisorPOPTestsMissing_ipc	299
122.	Missing IPC?	299
123	Customization with Qt Stylesheets	301
124	Appendix	303
124.	File format of statistics files	303
125	Setting Started with Plugin "Measurement"	305
125.	Start the Plugin	305
125.	Step-by-Step example	305
126	Overview	307
126.	Layout	307
126.	Tabs	307
126.	Virtual Console	307
127	Setup	309
127.	Load Measurement	309
127.1.	Load recent Measurement Button	309
127.1.	Submitted Jobs	309
127.1.2.	Job Status	309
127.	Start new Measurement	310
127.2.	Select Compiler version and MPI	310
127.2.1.	Compiler	310
127.2.1.	MPI	311
127.2.	Score-P version found in PATH	311
127.2.2.	Usable Configuration Status	311
127.2.	Find Score-P versions Button	311
127.2.	Browse Score-P Button	311
127.2.	Proceed Button	312
127.2.	Help Button	312
128	Instrumentation	313
128.	Browse Executable File Button	313
128.	Select Instrumentation Box	313
128.2.	Use Former Instrumentation	314
128.2.	Prepare New Instrumentation	314
128.	Adapt Build System	314
128.3.	Select Build System Box	314
128.3.1.	Adjust Makefile	314
128.3.	Open Makefile for Editing	314
128.3.2.	Browse Makefile Button	314
128.3.2.	Open detected Makefile Button	315

128. Rebuild Application	315
128.4. Build Command Box	315
128.4. Build Application Button	315
128. Continue with Analysis Button	315
129. Measurement	317
129. Presettings	317
129.1. Number of Processes	317
129.1. Number of Threads	317
129.1. Experiment Directory Name	318
129. Runs	318
129.2. Initial Run	318
129.2. Finetuned Run	318
129.2.2. Filter	318
129. Run the program	319
129.3. Prepare job Script	319
129.3.1. Open generated job Script	319
129.3.1. Open own job Script	319
129. Options after running the Program	319
Bibliography	321

1 Copyright

Copyright © 1998–2022 Forschungszentrum Jülich GmbH, Germany

Copyright © 2009–2015 German Research School for Simulation Sciences GmbH, Jülich/Aachen, Germany

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of Forschungszentrum Jülich GmbH or German Research School for Simulation Sciences GmbH, Jülich/Aachen, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2 Cube User Guide

2.1 Abstract

CUBE is a presentation component suitable for displaying performance data for parallel programs including MPI and OpenMP applications. Program performance is represented in a multi-dimensional space including various program and system resources. The tool allows the interactive exploration of this space in a scalable fashion and browsing the different kinds of performance behavior with ease. CUBE also includes a library to read and write performance data as well as operators to compare, integrate, and summarize data from different experiments. This user manual provides instructions of how to use the CUBE display, how to use the operators, and how to write CUBE files.

The version 4 of CUBE implementation has an incompatible API and file format to preceding versions.

2.2 Introduction

CUBE (CUBE Uniform Behavioral Encoding) is a presentation component suitable for displaying a wide variety of performance data for parallel programs including MPI [?] and OpenMP [?] applications. CUBE allows interactive exploration of the performance data in a scalable fashion. Scalability is achieved in two ways: hierarchical decomposition of individual dimensions and aggregation across different dimensions. All metrics are uniformly presented in the same display and thus provide the ability to easily compare the effects of different kinds of program behavior.

CUBE has been designed around a high-level data model of program behavior called the *cube performance space*. The CUBE performance space consists of three dimensions: a metric dimension, a program dimension, and a system dimension. The *metric dimension* contains a set of metrics, such as communication time or cache misses. The *program dimension* contains the program's call-tree, which includes all the call paths onto which metric values can be mapped. The *system dimension* contains the components executing in parallel, which can be processes or threads depending on the parallel programming model. Each point (m, c, s) of the space can be mapped onto a number representing the actual measurement for metric m while the control flow of process/thread s was executing call path c . This mapping is called the *severity* of the performance space.

Each dimension of the performance space is organized in a *hierarchy*. First, the metric dimension is organized in an inclusion hierarchy where a metric at a lower level is a subset of its parent. For example, communication time is a subset of execution time. Second, the program dimension is organized in a call-tree hierarchy. However, sometimes it can be advantageous to abstract away from the hierarchy of the call-tree, for example if one is interested in the severities of certain methods, independently of the position of their invocations. For this purpose CUBE supports also flat call profiles, that are represented as a flat sequence of all methods. Finally, the system dimension is organized in a multi-level

hierarchy consisting of the levels, e.g., machine, smp node, process, and thread. This hierarchy can vary depending on the used system.

CUBE also provides a *library* to read and write instances of the previously described data model in the form of a cubex file (which is a tar ed directory). The file representation is divided into a *metadata* part and a *data* part. The metadata part describes the structure of the three dimensions plus the definitions of various program and system resources and stored in a form of an tar file `anchor.xml` inside of the cubex envelope. The data part contains the actual severity numbers to be mapped onto the different elements of the performance space and stored in binary format in various files inside of the cubex envelope.

The *display* component can load such a file and display the different dimensions of the performance space using three coupled tree browsers (figure 2.1). The browsers are connected in such a way that you can view one dimension with respect to another dimension. The connection is based on *selections*: in each tree you can select one or more nodes. For example, in Figure 2.1 the Execution metric, the adi call path node, and Process 0 are selected. For each tree, the selections in the trees on its left-hand-side (if any) restrict the considered data: The metric nodes aggregate data over all call paths and all system-tree nodes, the call-tree aggregates data for the Execution metric over all system nodes, and each node of the system-tree shows the severity for the Execution metric of the selected call path for this system node.

If the CUBE file contains topological information, the distribution of the performance metric across the topology can be examined using the *topology view*.

As performance tuning of parallel applications usually involves multiple experiments to compare the effects of certain optimization strategies, CUBE includes a feature designed to simplify cross-experiment analysis. The *CUBE algebra* [?] is an extension of the framework for multi-execution performance tuning by Karavanic and Miller [?] and offers a set of operators that can be used to compare, integrate, and summarize multiple CUBE data sets. The algebra allows the combination of multiple CUBE data sets into a single one that can be displayed and examined like the original ones.

In addition to the information provided by plain CUBE files a statistics file can be provided, enabling the display of additional statistical information of severity values. Furthermore, a statistics file can also contain information about the most severe instances of certain performance patterns – globally as well as with respect to specific call paths. If a trace file of the program being analyzed is available, the user can connect to a trace browser (i.e. Vampir) and then use CUBE to zoom their timelines to the most severe instances of the performance patterns for a more detailed examination of the cause of these performance patterns.

The following sections explain how to use the CUBE display, how to create CUBE files, and how to use the algebra and other tools.

2.3 Command line options

To invoke GUI for CUBE profile exploration one uses command:

```
cube [options] filename
```

A list of main options:

- disable-plugins** start cube with all plugins disabled
- docpath=<path>** path to the html documentation
- presentation** opens cube in presentation mode, which shows a mouse icon next to the cursor
- single** disable parallel execution of cube
- start <plugin> [args]** start context free plugin with the name <plugin>
- verbose** print detailed information
- h|-help** Display list of command line options

A list of developer options:

- disable-calculation** disable automatic calculation of tree items
- expert** start cube in expert mode which shows e.g. ghost metrics or additional analyses in Advisor plugin
- memory=<strategy>** uses given memory strategy. If the option is omitted, CubeGUI reads the data from the .cubex file at the first access. "preload" reads all data into memory during the initialization phase. "lastN" keeps the last N data rows in memory. N is set via environment CUBE_NUMBER_ROWS.

2.4 Environment variables

CUBE provides the option of displaying an online description for entries in the metric-tree via a context menu. By default, it will search for the given HTML description file on all the mirror URLs specified in the CUBE file. In case there is no Internet connection, the Qt-based CUBE GUI can be configured to also search in a list of local directories for documentation files. These additional search paths can be specified via the environment variable CUBE_DOCPATH as a colon-separated list of local directories, e.g.,

```
CUBE_DOCPATH=/opt/software/doc:/usr/local/share/doc
```

Note that this feature is only available in the Qt-based GUI and **not** in the older wxWidgets-based one.

To prevent CUBE from trying to load the HTML documentation via HTTP or HTTPS mirror URLs (e.g., in restricted environments where outbound connections are blocked by a firewall and the timeout is taking very long), the environment variable CUBE_DISABLE_HTTP_DOCS can be set to either 1, yes or true.

Cube searches for plugins in the directory "cube-plugins/" below the installation directory. This is the place where the predefined plugins are installed. With the environment variable CUBE_PLUGIN_DIR one can specify a user defined place where third-party plugins are installed. If CUBE_PLUGIN_DIR contains a colon or semicolon separated list of paths, these

paths are prepended to the default search path.

There are environment variables coming from the CubeLib library. Hence they will have an effect also for the CubeLib tools.

These are two variables, CUBE_TMP and CUBE_DATA_LOADING.

During runtime CubeLib creates some temporary files, which usually are saved into the TMP directory. However, some systems put a quota on file size and on file numbers on the temporary directory. One wants to overcome this limitation by using another place.

Variable CUBE_TMP (following are aliases CUBE_TEMP, CUBE_TMPDIR, SCALASCA_TMP, SCALASCA_TEMP, SCALASCA_TMPDIR, SCOREP_TMP, SCOREP_TEMP, SCOREP_TMPDIR, TMP, TEMP, (TMPDIR win32 only)) informs CubeLib which directory to use for the temporary files.

CUBE C++ library allows to control the way it loads the data using the environment variable CUBE_DATA_LOADING. Following values are possible:

1. **keepall** - data is loaded on demand and kept in memory to the end of lifecycle of the Cube object.
2. **preload** - all data is loaded during the metric initialization and kept in memory to the end of lifecycle of the Cube object.
3. **manual** - Application should request and drop the data sets explicitly. No correctness check is performed. Therefore one has to use this strategy with care.
4. **lastn** - Only N last used data rows are kept in memory. N is specified via environment variable CUBE_NUMBER_ROWS

2.5 Using the Display

This section explains how to use the CUBE-QT display component. After installation, the executable "cube" can be found in the specified directory of executables (specifiable by the "prefix" argument of configure, see the CUBE Installation Manual). The program supports as an optional command-line argument the name of a cube file that will be opened upon program start.

After a brief description of the basic principles, different components of the GUI will be described in detail.

2.5.1 Basic Principles

The CUBE-QT display has three tree browsers, each of them representing a dimension of the performance space (figure 2.1). Per default, the left tree displays the metric dimension, the middle tree displays the program dimension, and the right tree displays the system dimension. The nodes in the metric tree represent metrics. The nodes in the program dimension can have different semantics depending on the particular view that has been selected. In Figure 2.1, they represent call paths forming a call-tree. The nodes in the system dimension represent machines, nodes, processes, or threads from top to bottom.

Each node is associated with a value, which is called the *severity* and is displayed simultaneously using a numerical value as well as a colored square. Colors enable the easy

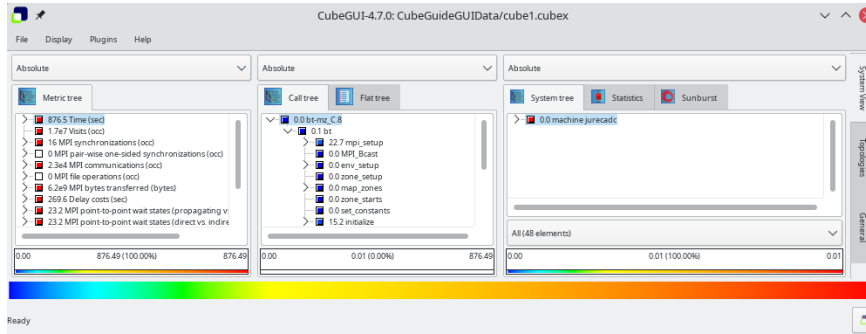


Figure 2.1: CUBE display window

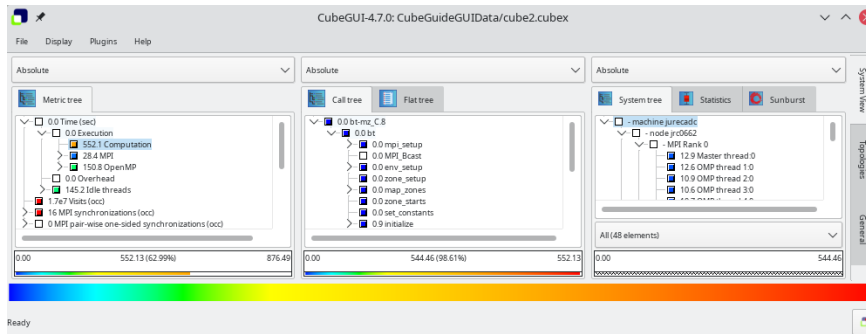


Figure 2.2: CUBE display window with expanded metric node "Execution"

identification of nodes of interest even in a large tree, whereas the numerical values enable the precise comparison of individual values. The sign of a value is visually distinguished by the *relief* of the colored square. A raised relief indicates a positive sign, a sunken relief indicates a negative sign.

Users can perform two basic types of actions: selecting a node or expanding/collapsing a node. In the metric-tree in figure 2.1, the metric Execution is selected. *Selecting* a node in a tree causes the other trees on its right to display values for that selection. For the example of figure 2.1, the metric-tree displays the total metric values over all call-tree and system nodes, the call-tree displays values for the Execution metric over all system entities, and the system-tree for the Execution metric and the adi call-tree node. Briefly, a tree is always an aggregation over all selected nodes of its neighboring trees to the left.

Collapsed nodes with a subtree that is not shown are marked by a [+] sign, *expanded* nodes with a visible subtree by a [-] sign. You can expand/collapse a node by left-clicking on the corresponding [+] / [-] signs. Collapsed nodes have *inclusive* values, i.e., their severity is the sum of the severities over the whole collapsed subtree. For the example of Figure 2.1, the Execution metric value 3496.10 is the total time for all executions. On the other hand, the displayed values of expanded nodes are their *exclusive* values. E.g., the expanded Execution metric node in Figure 2.2 shows that the program needed 2839.54 seconds for execution other than MPI.

Note that expanding/collapsing a selected node causes the change of the current values

in the trees on its right-hand side. As explained above, in our example in [Figure 2.1](#) the call-tree displays values for the Execution metric over all system entities. Since the Execution node is collapsed, the call-tree severities are computed for the whole Execution metric's subtree. When expanding the selected Execution node, as shown in [Figure 2.2](#), the call-tree displays values for the Execution metric without the MPI metric.

2.5.2 GUI Components

The GUI consists (from top to bottom) of

- a menu bar,
- three value mode combo boxes,
- three resizable panes each containing some tabs,
- three selected value information widgets,
- a color legend, and
- a status bar.

The three resizable panes offer different views: the metric, the call, and the system pane. You can switch between the different tabs of a pane by left-clicking on the desired tab at the top of the pane. Note that the order of the panes can be changed (see the description of the menu item *Display* \Rightarrow *Dimension order* in Section 2.5.2.1).

The metric pane provides only the metric-tree browser. The call pane offers a call-tree browser and a flat call profile. If OpenMP tasks have been instrumented, an additional task-tree is inserted. The system pane has a system-tree browser. Tree browsers also provide a context menu.

2.5.2.1 Menu Bar

The menu bar consists of four menus: a file menu, a display menu, a plugin menu and a help menu. Some menu functions also have a keyboard shortcut, which is written besides the menu item's name in the menu. E.g., you can open a file with Ctrl+O without going into the menu. A short description of the menu items is visible in the status bar if you stay for a short while with the mouse above a menu item.

1. **File:** The file menu offers the following functions:

- a) **Open (Ctrl+O):** Offers a selection dialog to open a CUBE file. In case of an already opened file, it will be closed before a new file gets opened. If a file got opened successfully, it gets added to the top of the recent files list (see below). If it was already in the list, it is moved to the top.
- b) **Open URL:** Opens a remote file dialog (see section 2.6)
- c) **Save as (Ctrl+S):** Offers a selection dialog to save a copy of a CUBE file. Opened CUBE file stays loaded in cube.
- d) **Close (Ctrl+W):** Closes the currently opened CUBE file. Disabled if no file is opened.
- e) **Open external:** Opens a file for the external percentage value mode (see section 2.5.2.2).
- f) **Close external:** Closes the current external file and removes all corresponding data. Disabled if no external file is opened.
- g) **Settings:** Offers saving, loading, and deletion of global settings. Global settings don't depend on the loaded cube file and are saved in a system specific format. These settings e.g. store the appearance of the application like the widget sizes,

color and precision settings, the order of panes, etc.

"Restore last state" depends on a loaded cube file. If it is activated, the state of the cube file, e.g. selected and expanded items, is saved before the cube file is closed and restored after loading.

- h) **Screenshot:** The function offers you to save a screen snapshot in a PNG file. Unfortunately the outer frame of the main window is not saved, only the application itself.
- i) **Quit (Ctrl+Q):** Closes the application.
- j) **Recent files:** The last 5 opened files are offered for re-opening, the top-most being the most recently opened one. A full path to the file is visible in the status bar if you move the mouse above one of the recent file items in the menu.

2. **Display:** The display menu offers the following functions:

- a) **Dimension order:** As explained above, CUBE has three resizable panes. Initially the metric pane is on the left, the call pane is in the middle, and the system pane is on the right-hand side. However, sometimes you may be interested in other orders, and that is what this menu item is about. It offers all possible pane orderings. For example, assume you would like to see the metric and call values for a certain thread. In this case, you could place the system pane on the left, the metric pane in the middle, and the call pane on the right, as shown in Figure 2.3. Note that in panes to the left of the metric pane no meaningful values can be presented, since they miss a reference metric; in this case values are specified to be undefined, denoted by a "-" (minus) sign.

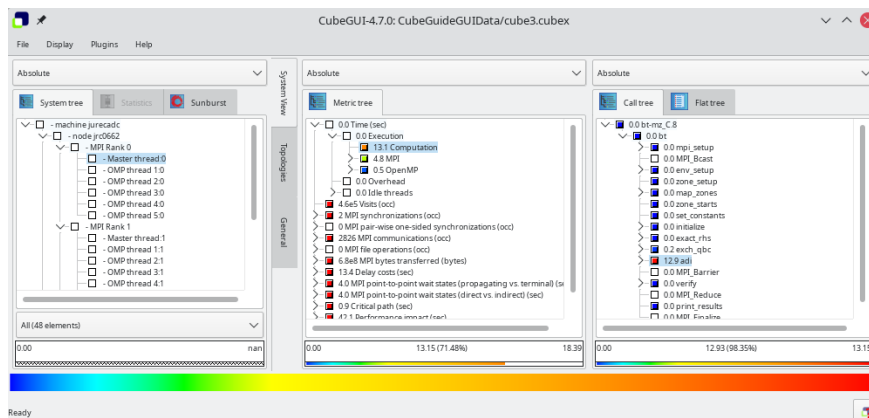


Figure 2.3: Modified pane order via the menu "Display => Dimension order"

- b) **Choose/Edit colormap:** Allows for selection of color maps and changing of color settings in a new dialog. In the configuration dialog, the Ok button applies the settings to the display and closes the dialog, the Apply button applies the settings to the display, and Cancel cancels all changes since the dialog was opened (even if "Apply" was pressed in between) and closes the dialog.

The configuration dialog in Figure 2.4 shows the default color map for Cube. Other colormaps may be added using plugins, see for example the Advanced Colormap Plugin (2.7.4). At the top of the dialog you see a color legend with



Figure 2.4: Configuration dialog of the default colormap which opened via the menu "Display => Edit colormap"

some vertical black lines, showing the position of the color scale start, the colors cyan, green, and yellow, and the color scale end. These lines can be dragged with the left mouse button, or their position can also be changed by typing in some values between 0.0 (left end) and 1.0 (right end) below the color legend in the corresponding spins.

The different coloring methods offer different functions to interpolate the colors at positions between the 5 data points specified above.

With the upper spin below the coloring methods you can define a threshold percentage value between 0.0 and 100.0, below which colors are lightened. The nearer to the left end of the color scale, the stronger the lightening (with linear increase).

With the spin at the bottom of the dialog you can define a threshold percentage value between 0.0 and 100.0, below which values should be colored white.

- c) **Set font size:** Opens a dialog to set the font size. The size can also be changed with Control+<mouse-wheel> or Control+<->/<+>
- d) **Customize style sheets** Opens a dialog to define [123](#) to change e.g. the fonts and sizes of GUI elements.
- e) **Configure value view:** This menu item opens a dialog in which the icon and the textual value representation of the tree items can be configured. Depending on the data type of the selected metric, additional options and additional value view plugins may be available. For metrics that consist of more than one value, e.g. tau metrics (see figure [2.5](#)), the user can select which value should be used for the icon and which values for the following text.

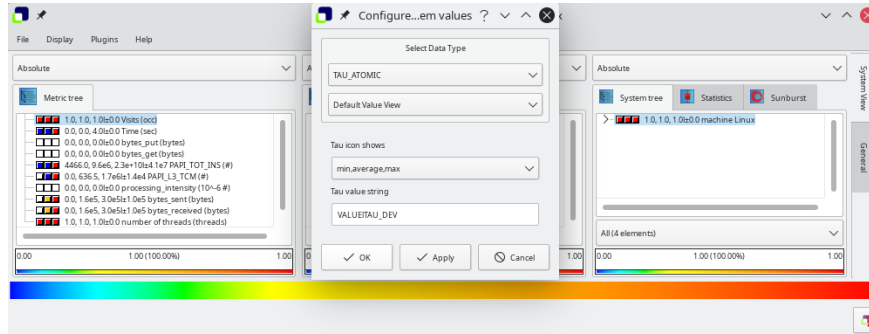


Figure 2.5: Value view config dialog for tau metrics

- f) **Precision:** Activating this menu item opens a dialog for precision settings (see Figure 2.6). Besides Ok and Cancel, the dialog offers an Apply button, that applies the current dialog settings to the display. Pressing Cancel undoes *all* changes due to the dialog, even if you already pressed Apply previously, and closes the dialog. Ok applies the settings and closes the dialog.

It consists of two parts: precision settings for the tree displays, and precision settings for the selected value info widgets and the topology displays. For both formats, three values can be defined:

- i. **Number of digits after the decimal point:** As the name suggests, you can specify the precision for the fraction part of the values. E.g., the number 1.234 is displayed as 1.2 if you set this precision to 1, as 1.234 if you set it to 3, and as 1.2340 if you set it to 4.
 - ii. **Exponent representation above 10^x with x:** Here you can define above which threshold scientific notation should be used. E.g., the value 1000 is displayed as 1000 if this value is larger then 3 and as $1e3$ otherwise.
 - iii. **Display zero values below 10^{-x} with x:** Due to inexact floating point representation, it often happens that users wish to round down values very near by zero to zero. Here you can define the threshold below which this rounding should take place. E.g., the value 0.0001 is displayed as 0.0001 if this value is larger than 3 and as zero otherwise.
 - iv. **Use human readable units for bytes and occ:** If enabled, units will be displayed in a human readable format, e.g. MB or GB.
- g) **Trees:** This menu offers options to change the contents and the appearance of the items of all trees.
- i. **Configure Tree Item Marker** In this dialog, you can change the appearance of defined tree item markers. You may choose if the items should be marked with a special background color or with an icon (see 2.7.3).
 - ii. **Demangle Function Names** (only call trees) If this option is enabled (default), cube tries to demangle function names.
 - iii. **Shorten Function Names** (only call trees) This menu item opens a dialog in which you can hide parts of long function names. You may hide argument

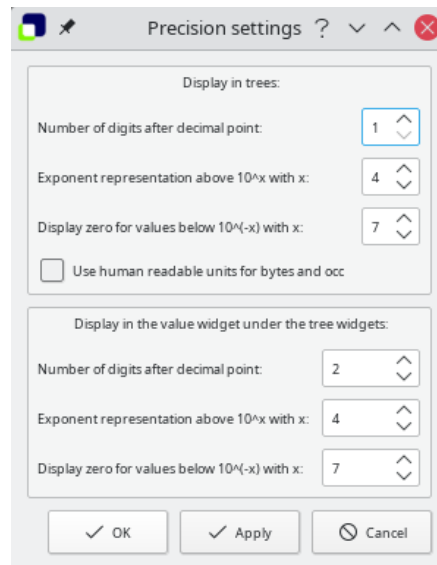


Figure 2.6: Display => Precision

lists and return values of C++ functions. You may also hide namespaces, class and templates from C++ function names. For Fortran subroutines, module names can be hidden.

- iv. **Append rank to system tree items** If this option is enabled, the MPI rank is appended to all system tree leafs. This is useful, if the MPI level is hidden or if there is a large amount of threads.
- h) **Optimize width:** Under this menu item CUBE offers widget rescaling such that the amount of information shown is maximized, i.e., CUBE optimally distributes the available space between its components. You can chose if you would like to stick to the current main window size, or if you allow to resize it.
- i) **Show synchronization toolbar** The synchronization of several cube instances is described in [2.8.4](#).
- j) **Show bookmark toolbar** Shows a toolbar which allows you to save the current state of a loaded cube file along with a name and a textual description. The state implies e.g. the currently selected items, the value mode of the trees, the active tabs and the state of the plugins. These states are saved next to the opened cube file in *cubeasename.ini*.
- k) **Enable presentation mode** If the presentation mode is enable, a mouse icon is shown next to the cursor
- l) **Enable QWebEngine** QWebEngine is used for the HTML-rendering of the documentation, if the module is available. On some systems, problems with the graphics driver or OpenGL cause QWebEngine to display a blank window. For that reason it's possible to disable QWebEngine (*Display* \Rightarrow *QWebEngine*) and to show the documentation in a basic layout instead. This option is also saved as setting and used for the next start of CubeGUI.

3. **Plugins:** The plugin menu allows the user to define which plugins are loaded. For each loaded plugin, a submenu is added. The submenu contains a menu item to enable or disable the plugin and the plugin may add additional menu items.
 - a) **Initial activation settings:** Opens a dialog to define which plugins should be loaded.
 - b) **Activate/deactivate plugins:** Allows to activate or deactivate a plugin for the current session.
4. **Help:** The help menu provides help on usage and gives some information about CUBE.
 - a) **Getting started:** Opens a dialog with some basic information on the usage of CUBE.
 - b) **Mouse and keyboard control:** Lists mouse and keyboard controls as given in Section [2.7.8.1](#).
 - c) **What's this?:** Here you can get more specific information on parts of the CUBE GUI. If you activate this menu item, you switch to the "What's this?" mode. If you now click on a widget, an appropriate help text is shown. The mode is left when help is given or when you press Esc.

Another way to ask the question is to move the focus to the relevant widget and press Shift+F1.
 - d) **About:** Opens a dialog with release information.
 - e) **Plugin info** Shows information about the plugin version, a short description and its location in the file system
 - f) **Plugin documentation** shows the plugin documentation in a browser window
 - g) **Selected metric description:** Opens a new window showing the description of the currently selected metric, equivalent to *Documentation* in the metric-tree context menu. Disabled if online documentation is unavailable.
 - h) **Selected region description:** Opens a new window showing the description of the currently selected region, equivalent to *Documentation* in the call-tree context menu. Disabled if online documentation is unavailable.

2.5.2.2 Value modes

Each tree view has its own value mode combobox, a drop-down menu above the tree, where it is possible to change the way the severity values are displayed.

The default value mode is the **Absolute** value mode. In this mode, as explained below, the severity values from the CUBE file are displayed. However, sometimes these values may be hard to interpret, and in such cases other value modes can be applied. Basically, there are three categories of additional value modes.

- The first category presents all severities in the tree as percentage of a reference value. The reference value can be the absolute value of a selected or a root node from the same tree or in one of the trees on the left-hand side. For example, in the **Own root percent** value mode the severity values are presented as percentage of

the own root's (inclusive) severity value. This way you can see how the severities are distributed within the tree. All the value modes (2 – 8) fall into this category.

All nodes of trees on the left-hand side of the metric-tree have undefined values. (Basically, we could compute values for them, but it would sum up the severities over all metrics, that have different meanings and usually even different units, and thus those values would not have much expressiveness.) Since we cannot compute percentage values based on undefined reference values, such value modes are not supported. For example, if the call-tree is on the left-hand side, and the metric-tree is in the middle, then the metric-tree does not offer the **Call root percent** mode.

- The second category is available for system-trees only, and shows the distribution of the values within hierarchy levels. E.g., the **Peer percent** value mode displays the severities as percentage of the maximal value on the same hierarchy depth. The value modes (9 – 10) fall into this category.
- Finally, the **External percent** value mode relates the severity values to severities from another external CUBE file (see below for the explanation).

Depending on the type and position of the tree, the following value modes may be available:

1. **Absolute (default):** Available for all trees. The displayed values are the severity value as read from the cube file, in units of measurement (e.g., seconds). Note that these values can be negative, too, i.e., the expression "absolute" is not used in its mathematical sense here.
2. **Own root percent:** Available for all trees. The displayed node values are the percentage of their absolute values with respect to the absolute value of their root node in collapsed state.
3. **Metric root percent:** Available for trees on the right-hand side of the metric-tree. The displayed node values are the percentage of their absolute values with respect to the absolute value of the collapsed metric root node. If there are several metric roots, the root of the selected metric node is taken. Note, that multiple selection in the metric-tree is possible within one root's subtree only, thus there is always a unique metric root for this mode.
4. **Metric selection percent:** Available for trees on the right-hand side of the metric-tree. The displayed node values are the percentage of their absolute values with respect to the selected metric node's absolute value in its current collapsed/expanded state. In case of multiple selection, the sum of the selected metrics' values for the percentage computation is taken.
5. **Call root percent:** Available for trees on the right-hand side of the call-tree. Similar to the metric root percent, but the call-tree root instead of the metric-tree root is considered. In case of multiple selection with different call roots, the sum of those root values is considered.
6. **Call selection percent:** Available for trees on the right-hand side of the call-tree. Similar to the metric selection percent, percentage is computed with respect to the selected call node's value in its current collapsed/expanded state. In case of multiple selections, the sum of the selected call values is considered.
7. **System root percent:** Available for trees on the right-hand side of the system-tree. Similar to the call root percent, the sum of the inclusive values of all roots of selected system nodes are considered for percentage computation.

8. **System selection percent:** Available for trees on the right-hand side of the system-tree. Similar to the call selection percent, percentage is computed with respect to the selected system node(s) in its current collapsed/expanded state.
9. **Peer percent:** For the system-tree only. The peer percentage mode shows the percentage of the nodes' inclusive absolute values relative to the largest inclusive absolute peer value, i.e., to the largest inclusive value between all entities on the current hierarchy depth. For example, if there are 3 threads with inclusive absolute values 100, 120, and 200, then they have the peer percent values 50, 60, and 100.
10. **Peer distribution:** For the system-tree only. The peer distribution mode shows the percentage of the system nodes' inclusive absolute values on the scale between the minimum and the maximum of peer inclusive absolute values. For example, if there are 3 threads with absolute values 100, 120 and 200, then they have the peer distribution values 0, 20 and 100.
11. **External percent:** Available for all trees, if the metric tree is the left-most widget. To facilitate the comparison of different experiments, users can choose the external percentage mode to display percentages relative to another data set. The external percentage mode is basically like the metric root percentage mode except that the value equal to 100% is determined by another data set.

Note that in all modes, only the leaf nodes in the system hierarchy (i.e., processes or threads) have associated severity values. All other hierarchy levels (i.e., machines, nodes and eventually processes) are only used to structure the hierarchy. This means that their severity is undefined—denoted by a “-” (minus) sign—when they are expanded.

2.5.2.3 System resource subsets

By default, all system resources (typically threads) are included when determining boxplot statistics. Other defined subsets can be chosen from the combobox below the boxplot, such as “Visited” threads which are only those threads that visited the currently selected callpath. The current subset is retained until another is explicitly chosen or a new subset is defined.

Additional subsets are defined from the system-tree with the *Define subset* context menu using the currently selected threads via multiple selection (Ctrl+<left-mouse click>) or with the *Find Items* context menu selection option.

2.5.2.4 Tree browsers

A tree browser displays different hierarchical data structures in form of trees. Currently supported tree types are metric-trees, call-trees and their flat call profiles, and system-trees. The structure of the displayed data is common in all trees: The indentation of the tree nodes reflects the hierarchical structure. Expandable nodes, i.e., nodes with non-hidden children, are equipped with a [+]/[-] sign ([+] for collapsed and [-] for expanded nodes). Furthermore, all nodes have a color icon, a value, and a label.

The value of a node is computed, as explained earlier, basing on the current selections in the trees on the left-hand side and on the current value mode. The precision of the value display in trees can be modified, see the menu item *Display* ⇒ *Precision* in Section

2.5.2.1. The color icon reflects the position of the node's value between 0.0 and a maximal value. This maximal value is the maximal value in the tree for the absolute value mode, or 100.0 otherwise. See the menu item *Display* \Rightarrow *Choose colormap* in Section 2.5.2.1 and the context menu item *Min/max values* in the context menu description below for color settings.

A label in the metric-tree shows the metric's name. A label in the call-tree shows the last callee of a particular call path. If you want to know the complete call path, you must read all labels from the root down to the particular node you are interested in. After switching to the flat profile view (see below), labels in the flat call profile denote methods or program regions. A label in the system-tree shows the name of the system resource it represents, such as a node name or a machine name. Processes and threads are usually identified by a rank number, but it is possible to give them specific names when creating a CUBE file. The thread level of single-threaded applications is hidden. Multiple root nodes are supported.

After opening a data set, the middle panel shows the call-tree of the program. However, a user might wish to know which fraction of a metric can be attributed to a particular region (e.g., method) regardless of from where it was called. In this case, you can switch from the call-tree view (default) to the flat-profile view (Figure 2.7). In the flat-profile view, the call-tree hierarchy is replaced with a source-code hierarchy consisting of two levels: regions and their subroutines. Any subroutines are displayed as a single child node labeled *Subroutines*. A subroutine node represents all regions directly called from the region above. In this way, you are able to see which fraction of a metric is associated with a region exclusively, that is, without its regions called from there.

When tasks are encountered while reading the Cube file, a third tab next to call-tree is provided to display them separately. In general terms, tasks are pieces of code scheduled and executed by a runtime asynchronously. Due to their asynchronous nature and their ability to be suspended and continued at a potentially different position in the call-tree handling them inside the call-tree itself may lead to inconsistent results. For OpenMP, the call-tree therefore contains only stub nodes with visit and time metric values at those execution points, while the executions and their task local call-trees will be displayed in a separate tasks tab. Currently, only OpenMP tasks are generated by Score-P, however the paradigm attribute of those task instances allows handling of tasks of different paradigms.

If tasks are involved, the values of the trees on the left (default: metric tree) depend on the active call tab. The task-tree only contains the task related paths. The call-tree contains all paths except for the task-local trees, which are replaced by stub nodes at their execution points. The flat-tree on the other hand, still contains all execution paths. There may be items of the flat-tree, that cannot be calculated for exclusive metrics. These items consist of paths from the task tree and of paths from the call-tree. Their exact contributions cannot be determined. These values are marked with a dash and a warning message is displayed on the status line.

Tree displays are controlled by the left and right mouse buttons and some keyboard keys. The left mouse button is used to select or expand/collapse a node: You can expand/collapse a node by left-clicking on the attached [+]/[-] sign, and select it by left-clicking elsewhere in the node's line. To select multiple items, `Ctrl+<left-mouse click>` can be used. Selection without the `Ctrl` key deselects all previously selected nodes and selects the clicked node. In single-selection mode you can also use the up/down arrows to move the selection one node up/down. The right mouse button is used to pop up a context menu with node-specific information, such as online documentation (see the description of the context menu below).

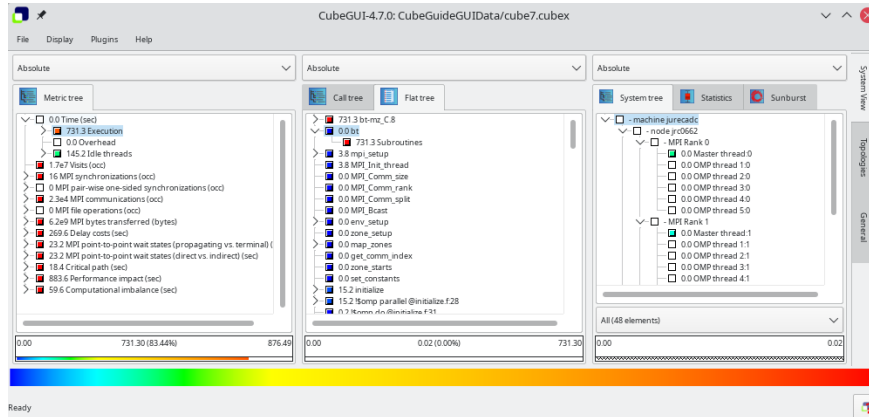


Figure 2.7: CUBE flat profile

Each tree has its own context menu which can be activated by a right mouse click within the tree's window. If you right-click on one of the tree's nodes, this node gets framed, and serves as a *reference node* for some of the menu items. If you click outside of tree items, there is no reference node, and some menu items are disabled.

The context menu consists, depending on the type of the tree, of some of the following items. If you move the mouse over a context menu item, the status bar displays some explanation of the functionality of that item.

1. **Collapse all:** Collapses all nodes in the tree.
2. **Collapse subtree:** Enabled only if there is a reference node. It collapses all nodes in the subtree of the reference node (including the reference node).
3. **Expand all:** Expands all nodes in the tree.
4. **Expand subtree:** Enabled only if there is a reference node. Expands all nodes in the subtree of the reference node (including the reference node).
5. **Expand largest:** Enabled only if there is a reference node. Starting at the reference node, expands its child with the largest inclusive value, and continues recursively with that child until it finds a leaf.
6. **Expand marked:** Shows all marked nodes by expanding their parents (see 2.7.3).
7. **Expand current level:** For system-trees only. Shows all nodes that are on the same hierarchy level as the chosen one by expanding their parents.
8. **Dynamic hiding:** Not available for metric-trees. This menu item activates dynamic hiding. All currently hidden nodes get shown. You are asked to define a percentage threshold between 0.0 and 100.0. All nodes whose color position on the color scale (in percent) is below this threshold get hidden. As default value, the color percentage position of the reference node is suggested, if you right-clicked over a node. If not, the default value is the last threshold. The hiding is called dynamic, because upon value changes (caused for example by changing the node selection) hiding is re-computed for the new values. In other words, value changes may change the visibility of the nodes.

- a) **Redefine threshold:** This menu item is enabled if dynamic hiding is already activated. This function allows to re-define the dynamic hiding threshold as described above.

During dynamic hiding, for expanded nodes with some hidden children and for nodes with all of its children hidden, their displayed (exclusive) value includes the hidden children's inclusive value. The percentage of the hidden children is shown in brackets next to this aggregate value.

- 9. **Static hiding:** Not available for metric-trees. This menu item activates static hiding. All currently hidden nodes stay hidden. Additionally, you can hide and show nodes using the now enabled sub-items:

- a) **Static hiding of minor values:** Enabled only in the static hiding mode. As described under dynamic hiding, you are asked for a hiding threshold. All nodes whose current color position on the color scale is below this percentage threshold get hidden. However, in contrast to dynamic hiding, these hidings are static: Even if after some value changes the color position of a hidden node gets above the threshold, the node stays hidden.
- b) **Hide this:** Enabled only in the static hiding mode if there is a reference node. Hides the reference node.
- c) **Show children of this:** Enabled only in the static hiding mode if there is a reference node. Shows all hidden children of the reference node, if any.

Like for dynamic hiding, for expanded nodes with some hidden children and for nodes with all of its children hidden, their displayed (exclusive) value includes the hidden children's inclusive value. The percentage of the hidden children is shown in brackets next to this aggregate value.

- 10. **No hiding:** Not available for metric-trees. This menu item deactivates any hiding, and shows all hidden nodes.
- 11. **Find items:** For all trees. Opens a text input widget below the corresponding tree to enter a regular expression to search for. If the user called the context menu over an item, the default text is the name of the reference node. All non-hidden nodes whose names contain the given expression are marked with a yellow background, and all collapsed nodes whose subtree contains such a non-hidden node by a light yellow background.

The button *expand all* expands all found items.

The button *select all* selects all found items. The selected items may still be collapsed.

The arrow buttons select the next or the previous found item. The shortcuts for these actions are F3 and Shift+F3.

- 12. **Clear found items:** For all trees. Removes the background markings of the preceding "find items" action.
- 13. **Define subset:** Only for system-tree. Uses the currently selected system resources (e.g., from a preceding *Find items*) to create a new subset of all system resources (typically threads) with the provided name. This is added to the combobox at the bottom of the system-tree and boxplot statistics panes, and becomes the currently

active subset for which statistics are calculated.

14. **Info/Documentation:** For metric and call-trees Shows combined information about the selected metric and call-tree items in a new tab. For the selected metric, information about display, unique name, data type, unit of measurements and kind of metric is shown. If the metric is derived, the CubePL expression is shown.

For the selected call path, information about call path id (to use it with command line tools like `cube_dump`), region beginning line, region ending line, region module, url with the online help and finally description of the region is shown.

If online documentation for the reference node is available, it is shown in a html widget below the information panels. For example, metrics might point to an online documentation explaining their semantics, or regions representing library functions might point to the corresponding library documentation.

QWebEngine is used for the HTML-rendering of the documentation, if the module is available. On some systems, problems with the graphics driver or OpenGL cause QWebEngine to display a blank window. For that reason it's possible to disable QWebEngine (*Display* \Rightarrow *QWebEngine*) and to show the documentation in a basic layout instead. This option is also saved as setting and used for the next start of CubeGUI.

Disabled, if not clicked over metric or call path item.

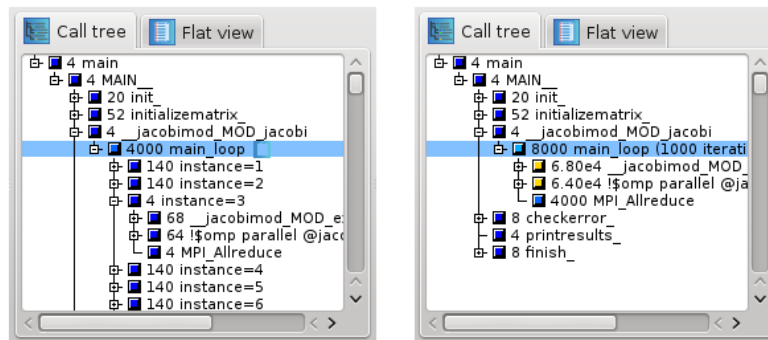


Figure 2.8: The item `main_loop` with 1000 iteration is marked as a loop. The aggregated view on the right is the result of selecting "Hide iterations".

15. **Hide iterations:** Only visible for calltree items that are recognized or manually defined as loop (see "Set as loop" below). By activating, all children of the loop are hidden. The grandchildren are shown and its values for the different iterations are aggregated (see Figure 2.8).
16. **Call site:** For call-trees only. Enabled only if there is a reference node. Offers information about the caller of the reference node.
 - a) **Location:** Displays information about the module and position within the module (line numbers) of the caller of the reference node.
 - b) **Set as loop:** Marks the selected tree item as loop. All subitems are treated as iterations. An additional context menu item "Hide iterations" appears.
17. **Called region:** For call-trees only. Enabled only if there is a reference node. Offers

information about the reference node.

- a) **Info:** Gives some short information about the reference node.
 - b) **Documentation:** Shows some (usually more extensive) online description for the reference node. Disabled if no online documentation is available.
 - c) **Location:** Displays information about the module and position within the module (line numbers) where the callee method of the reference node is defined.
18. **Min/max values:** Not for metric-trees. Here you can activate and deactivate the application of user-defined minimal and maximal values for the color extremes, i.e., the values corresponding to the left and right end of the color legend. If you activate user-defined values for the color extremes, you are asked to define two values that should correspond to the minimal and to the maximal colors. All values outside of this interval will get the color gray. Note that canceling any of the input windows causes no changes in the coloring method. If user-defined min/max values are activated, the selected value information widget (see Section 2.5.2.5) displays a (u) ' ' for user-defined" behind the minimal and maximal color values.
19. **Statistics:** Only available if a statistics file for the current CUBE file is provided. Displays statistical information about the instances of the selected metric in the form of a box plot. For an in-depth explanation of this feature see subsection 2.8.2.
20. **Max severity in trace browser:** Only available for metric and call-trees and only if a statistics file providing information about the most severe instance(s) of the selected metric is present. If CUBE is already connected to a trace browser (via *File* ⇒ *Connect to trace browser*), the timeline display of the trace browser is zoomed to the position of the occurrence of the most severe pattern so that the cause for the pattern can be examined further. For a more detailed explanation of this feature see subsection 2.8.3.
21. **Cut call tree/Cut selected call tree items** This context menu is enabled, if the right mouse button is pressed on a call tree item. If the mouse button is pressed and the item below the mouse pointer is part of a group of selected items, the action affects all selected items. Otherwise, only the item below the mouse item will be modified. The menu offers different modification possibilities:
- a) **Set as root:** Removes all call paths above the selected items and sets selected call paths as a root nodes.
 - b) **Prune element:** Removes the selected items and all their children. Their inclusive value will be added then to the exclusive value of their parents.
 - c) **Set as leaf:** Removes all children of the elements and shows the inclusive values.
 - d) **Undo** Undo last operation.
22. **Sort by inclusive/exclusive value (descending):** Sorts the nodes by their current values in descending order. The items will be automatically sorted, if the values change. If "Apply now" is selected, the values are only sorted once.
23. **Sort by name (ascending):** Sorts the nodes alphabetically by name in ascending order.
24. **Sort by name and trailing number (ascending):** For system tree only. Sorts the nodes alphabetically by name and the trailing rank in ascending order.

25. **Sort by order of definition:** Restores the original order.

2.5.2.5 Selected value info

Below each pane there is a selected value information widget. If no data is loaded, the widget is empty. Otherwise, the widget displays more extensive and precise information about the selected values in the tree above. This information widget and the topologies may have different precision settings than the trees, such that there is the possibility to display more precise information here than in the trees (see Section 2.5.2.1, menu *Display* \Rightarrow *Precision*).

The widget has a 3-line display. The first line displays at most 4 numbers. The left-most number shows the smallest value in the tree (or 0.0 in any percentage value mode for trees, or the user-defined minimal value for coloring if activated), and the right-most number shows the largest value in the tree (or 100.0 in any percentage value mode in trees, or the user-defined maximal value for coloring if activated). Between these two numbers the current value of the selected node is displayed, if it is defined. Additionally, in the absolute value mode it is followed by the percentage of the selected value on the scale between the minimal and maximal values, shown in brackets. Note that the values of expanded non-leaf system nodes and of nodes of trees on the left-hand side of the metric-tree are not defined. If the value mode is not the absolute value mode, then in the second line similar information is displayed for the absolute values in a light gray color.

In case of multiple selection, the information refers to the sum of all selected values. In case of multiple selection in system trees in the peer distribution and in the peer percent modes, this sum does not state any valuable information, but is displayed for consistency reasons.

If the widget width is not large enough to display all numbers in the given precision, then a part of the number displays get cut down and a "... " indicates that not all digits could be displayed.

Below these numbers, in the third line, a small color bar shows the position of the color of the selected node in the color legend. In case of undefined values, the legend is filled with a gray grid.

2.5.2.6 Color legend

By default, the colors are taken from a spectrum ranging from blue over cyan, green, and yellow to red, representing the whole range of possible values. You can change the color settings in the menu, *Display* \Rightarrow *Choose colormap*, see Section 2.5.2.1. Exact zero values are represented by the color white (in topologies you can decide whether you would like to use white or the minimal color, see Section 2.7.13, menu *Topology*).

2.5.2.7 Status Bar

The status bar displays some status information, like state of execution for longer procedures, hints for menus the mouse pointing at etc.

The status bar shows the most recent log message. By clicking on it, the complete log becomes visible.

2.6 Client-Server

2.6.1 Cube Server

cube_server is part of the cubelib installation.

```
cube_server [ -p N ] Bind socket on port N (default port: 3300)
```

Many hosts don't allow ports to be accessed from the outside. You may use SSH tunneling (also referred to as SSH port forwarding) to route the local network traffic through SSH to the remote host.

In the following example, cube_server is started with the default port 3300 on the remote server server.example.com. The traffic, which is sent to localhost:3000, will be forwarded to server.example.com on the same port.

```
[client]$ ssh -L 3300:server.example.com:3300 server.example.com
[server.example.com]$ cube_server
Cube Server: CubeLib-4.6.0 (external) [POSIX]
cube_server[5247] Waiting for connections on port 3300.
```

2.6.2 Cube Client

CubeGUI can also be used to open a cube file on a remote host which runs cube_server (see Figure 2.9). After selecting "Open Url..." a remote file dialog appears (see Figure 2.10).

The first line contains the URL to the remote cube server 2.6.1. After having changed this line, the reload-Button on the right has to be pushed to reconnect to the server.

2.7 Cube GUI Plugins

The features of cube can be extended using plugins. There is a set of predefined plugins which are described in the following sections. Before a cube file is loaded, the Plugin menu only contains the menu items "Configure plugin search path" and "Initial activation settings".

By Selecting the second item, a dialog is created which lists all available plugins (see Figure 2.12).

You may enable or disable all plugins, or select individual plugins that will be activated or deactivated. After loading a cube file, all suitable plugins are activated. Each plugin may add a submenu (see Figure 2.11) to the Plugins menu.

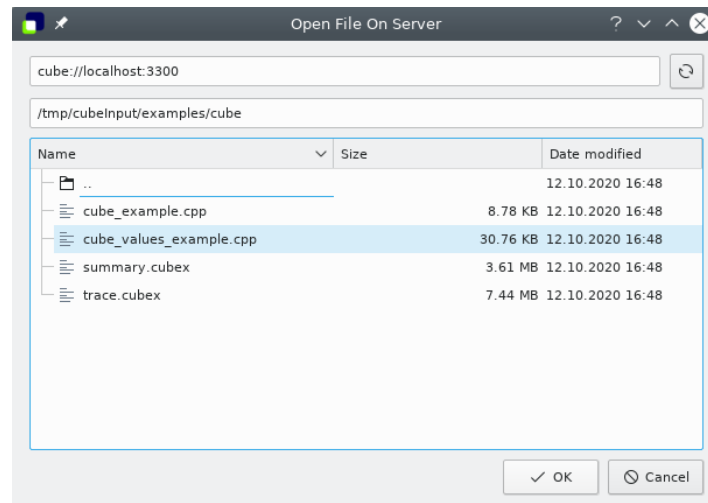


Figure 2.9: File menu

Cube searches for plugins in the directory "cube-plugins/" below the installation directory. This is the place where the predefined plugins are installed. If the environment variable CUBE_PLUGIN_DIR contains a colon or semicolon separated list of paths, these paths are prepended to the default search path.

Selecting "Configure plugin search path" of the plugin menu shows a dialog (see Figure 2.13), which allows to prepend additional search paths. The directory icon on the right opens a file browser whose selection is added to the input line on top and which is added to the path with the "add" button.

2.7.1 Detach Plugin Tabs

By clicking with the right mouse button on a plugin tab, the contents of the tab are moved to a separate window (see Figure 2.14). If the window is closed, the contents are moved to the tab widget again.

2.7.2 Context free plugins

Context free plugins are available via menu "File -> Start" as long no Cube is loaded in Cube GUI. Is one Cube file is loaded, one should close it using "File -> Close".

2.7.2.1 Plugin "Diff"

This plugin allows to perform algebra operation "difference" on two selected cubes and displays result in Gui.

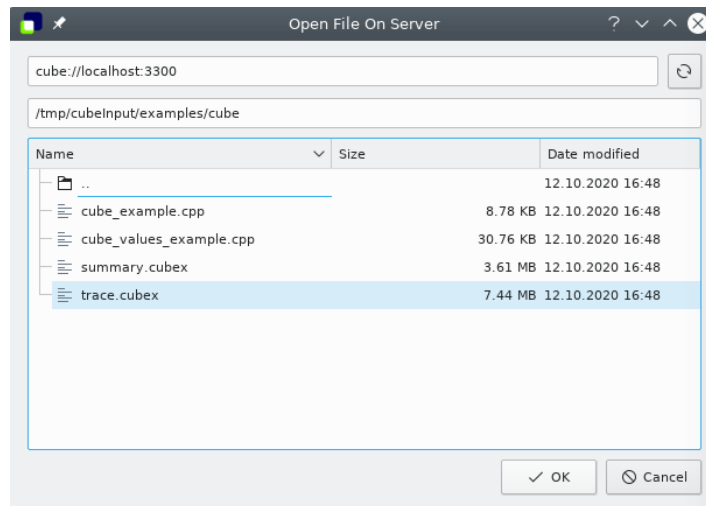


Figure 2.10: Remote file dialog

2.7.2.2 Plugin "Mean"

This plugin allows to perform algebra operation "mean" on selected cubes and displays result in Gui.

2.7.2.3 Plugin "Merge"

This plugin allows to perform algebra operation "merge" on selected cubes and displays result in Gui.

2.7.2.4 Plugin "Scaling"

This plugin allows user to do a simple scaling analysis. One selects a directory with the series of measurements. "Scaling" plugin creates a scaling profile, where metric and call-trees are identical (merged) with the input measurements, and the system-tree is an artificial scaling tree. Every entry in it corresponds to a single measurement. In couple with the "Jenga Fett" plugin (third party, www.scalasca.org) result is displayed as a series of stacked bars and allows the user to analysis the scaling behavior of the application.

2.7.2.5 Plugin "Tau2Cube"

This plugin allows user to open TAU Profile Directory using Cube Gui and explore it in casual way.

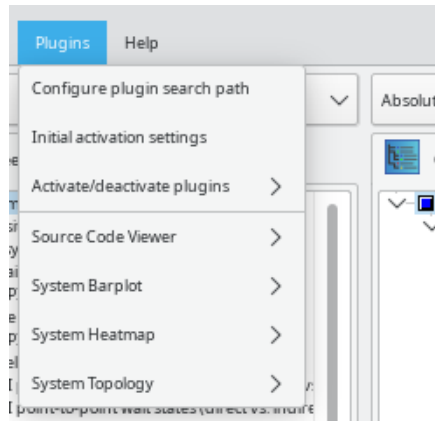


Figure 2.11: plugin menu

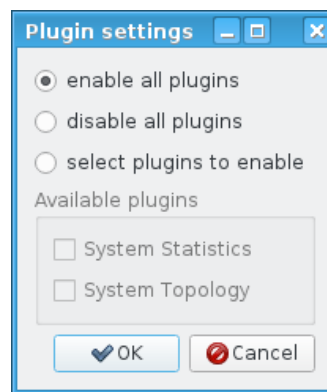


Figure 2.12: plugin settings dialog

2.7.2.6 Plugin "Measurement"

This plugin allows user to perform a Score-P measurement of own project and explore result in the CubeGUI. See [125](#)

2.7.3 Tree Item Marker

A plugin may define one or more tree item marker to tag items of interest.

Tree items are marked in different ways:

- Items with a colored background show that a plugin has set a marker
- Items with a colored frame indicate that a collapsed child has been marked.
- Items with a black frame indicate that there are several collapsed children with different marker.
- Items with a dotted frame show a dependency. A marked item of the right neighbor tree depends on

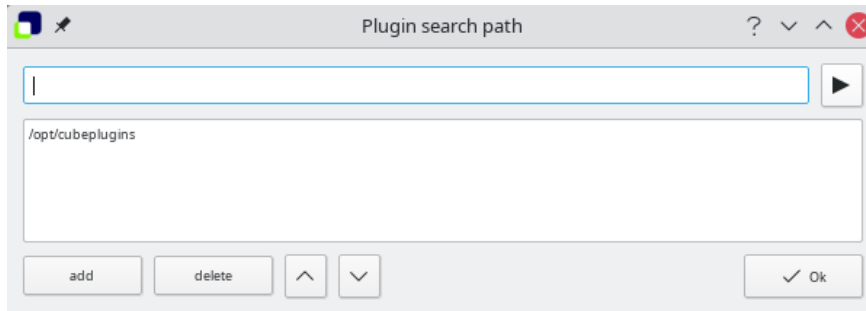


Figure 2.13: plugin search path dialog

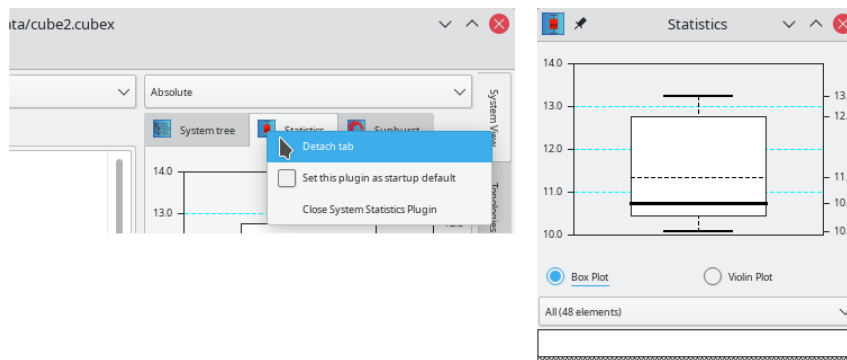


Figure 2.14: Boxplot plugin tab is detached

- Items can be grayed out. These items are either marked as unimportant by a plugin, or the user has chosen to gray out all items, for which no marker is set. this item. The dependent item is only marked, if the dotted item is selected.

The figure 2.16 shows two plugins which define marker. The Statistic Plugin marks all items with information about the most severe instances with a blue background and an icon. The Launch Plugin uses green marker and does not define an icon. Both of them use marker for items of the system-tree and for items of the call-tree that depend on items of the system-tree.

The Tree Item Marker dialog (see figure 2.11) allows the user to change the color of each marker, to disable the drawing of colors or icons and to emphasize the marked items by graying out the other items.

2.7.4 Advanced Color Map Plugin

Advanced Color Map Plugin provides additional color maps. The configuration dialogs are presented in Figure 2.17. For every color map, the plot allows for change of data accepted by color map and one can do that using left and right marker, by dragging the marker or providing exact position through a double click near the marker value (new dialog will appear). The default color for values out of range is grey.

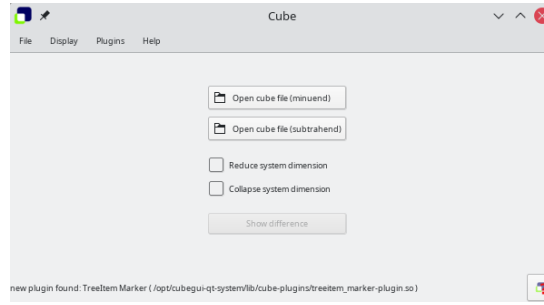


Figure 2.15: Plugin Diff

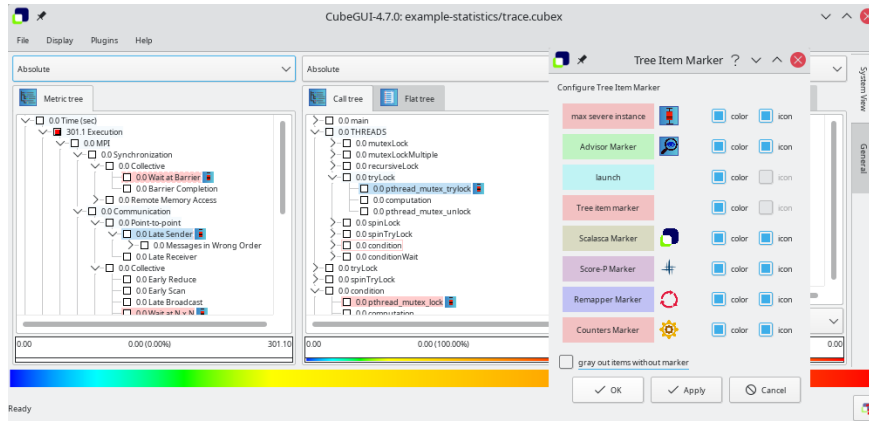


Figure 2.16: Tree item marker

One can change colors of scheme (for some color maps) and color for values out of range. Double mouse click on proper part of the plot opens a dialog with selection of RGB color. Additionally, one can adjust the plot marker or reset to default values through the context menu.

Currently the plugin adds four different sets of color maps:

1. **Sequential:** Scheme is defined by starting and ending color with linear or exponential interpolation between them. Predefined schemes provide simple interpolation from one color to pure white. Middle marker allows for subtle change of interpolation.
2. **Divergent:** This scheme is defined by an interpolation from starting to ending color, but with a critical value between them, depicted with the pure white. The position of critical point can be set with the middle marker.
3. **Cubehelix:** Scheme designed primarily for display of astronomical intensity images. The coloring is based on distribution from black to white, with R, G and B helixes giving additional deviations. Cubehelix is defined by four parameters:
Start colour - starting value for color, floating-point number between 0.0 and 3.0. R = 1, G = 2, B = 0
Rotations - floating-point number of R -> G -> B rotations from the start to the end. Negative value corresponds to negative direction of rotation.

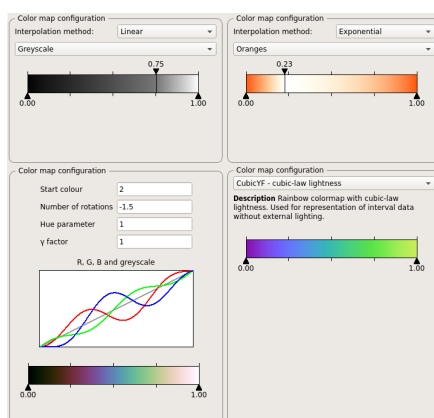


Figure 2.17: The examples of configuration for Advanced Color Maps. Upper row, starting from left: sequential, divergent; lower row, starting from left: cubehelix, improved rainbow.

Hue - non-negative value which controls saturation of the scheme, with pure greyscale for hue equal to 0.

Gamma factor - non-negative value which configures intensity of colours. Values below one emphasizes low intensity values and creates brighter color scheme. Values above one emphasizes high intensity values and generates darker color map.

Reference: Green, D. A., 2011, 'A colour scheme for the display of astronomical intensity images', Bulletin of the Astronomical Society of India, 39, 289.

4. **Improved rainbow colormap:** Set of color maps based on original jet (rainbow) scheme, but with different lightness distribution. The goal behind these schemes is to provide map with more balanced perception, which is poor for original jet, mainly because of sharp changes in lightness. These maps doesn't provide any possibility for configuration.

Reference: Perceptually improved colormaps, MATLAB Central

2.7.5 Metric Editor Plugin

The metric editor plugin allows to create derived metrics as root or child metrics. To create or edit such a metric, use the right mouse button to show the context menu of the metric-tree. Then select the menu item "Edit metric->Create derived metric". If the context menu is called on a tree item, the new metric may also be inserted as a child".

For detailed documentation of CubePL please see [?].

Some details about the fields in the dialog:

1. **Select metric from collection:** Provides a list of predefined derived metric, which might be helpful for the analysis. A new metric may be added to the collection with the plus button, existing user defined metrics may be updated that way.
2. **Derived metric type:** Selects the type of the derived metrics. Available are :

Figure 2.18: Create derived metric

Postderived metric, Prederived exclusive metric and Prederived inclusive metric.

3. **Display name:** Sets the display name of the metric in the metric-tree.
4. **Unique name:** Sets the unique name of the metric. There is no check done if another metric is present with the same unique name.
5. **Data type :** For derived metrics it is preselected and is always DOUBLE.
6. **Unit of measurement:** Selects a unit of measurement. It is a user defined string.
7. **URL:** Selects a URL with the documentation about this metric.
8. **Description:** Describes a metric.
9. **Calculation:** Field where one enters the CubePL expression for the derived metric. Automatic syntax check is done. If there is a syntax error, dialog highlights the place of the error and gives an error message.
10. **Calculation Init:** Field where one enters the initialisation CubePL expression for the derived metric, which is executed only once after metric creation.
Automatic syntax check is done. If there is a syntax error, dialog highlights the place of the error and gives an error message.
11. **Aggregaton "+":** Prederived metrics can specify an expression for the operator "+" in the aggregation formula. In this field one can redefine it.
Automatic syntax check is done. If there is a syntax error, dialog highlights the place

of the error and gives an error message.

12. **Calculation "-":** Prederived inclusive metric can specify an expression for the operator "-" in the aggregation formula. In this field one can redefine it.

Automatic syntax check is done. If there is a syntax error, dialog highlights the place of the error and gives an error message.

13. **Create metric** - This button is only enabled, if all required fields are set, the metric identifier is unique and the syntax is valid. First, the new metric is checked for undefined references. Other metrics, which are referenced by the new metric and which are part of the collection are inserted automatically. These automatically inserted metrics are hidden. If all references are resolved, the dialog is closed and a new metric with the given values is created.

14. **Cancel** - closes dialog without creating any metric.

15. **Share this metric with SCALASCA group** - Offers you to sent the metric definition via email to the SCALASCA group, so it might be included into the library of derived metrics in the future releases. Enabled only if definition of metric is valid.

To simplify the creation of a derived metric a little bit there is a way to fill the fields of this dialog automatically.

If one prepares a file with the following syntax one can select it and open "drop" on dialog via drag'n'drop, or copy its content into clipboard and paste in the dialog.

Example of a syntax of this file:

```
metric type: postderived
display name: Average execution time
unique name: kenobi
uom:sec
url: https://scalasca.org/documentation.html#kenobi
description:Calculates an average execution time
#
# Here is the Kenobi metric
#
cubepl expression: metric::time(i)/metric::visits(e)

cubepl init expression:

cubepl plus expression:  arg1 + arg2

cubepl minus expression: arg1 - arg2
```

metric type can have values: postderived, prederived_exclusive or prederived_inclusive.

1. **Remove metric** Removes metric from the metric-tree, if it is not used by other metrics.
2. **Edit metric** It offers a dialog to edit expressions (standard, initialisation, aggregation) of a derived metric. Enabled if selected metric is a derived metric. Window for editing is same like in "Create derived metric" case.

2.7.6 Metric Identification Plugin

Cube displays relatively many metrics in its "Metric" pane. These metrics have different origin or purpose. They can be generated by Score-P, Scalasca, Cube remapper or be

hardware counters. On order to support user to identify which metric origins from which tool, serves which purpose, Cube provides "Metric Identification Plugin" (see Figure 2.19)

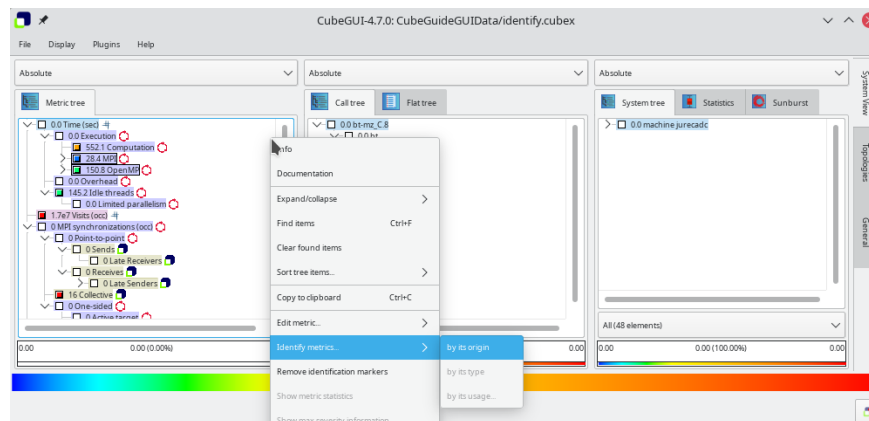


Figure 2.19: Metric identification

Tooltip displays help to every used environment variable with its possible values.

2.7.7 Score-P Configuration Plugin

This plugin (see Figure 2.20) presents the file "scorep.cfg", if found, in tabular way. Tooltip displays help to every used environment variable with its possible values.

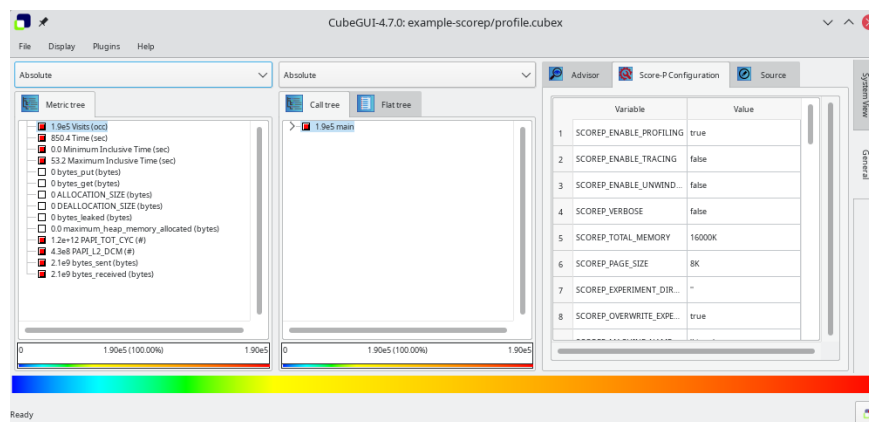


Figure 2.20: Score-P Configuration

2.7.8 Source Code Viewer

The Source code viewer plugin (see figure 2.21) displays the source code of the selected call-tree item. The file is opened in read-only mode per default. If you wish to edit the

text, please uncheck the Read only box in the plugin menu. The menu item "Set external editor" allows to open the source file with an external editor.

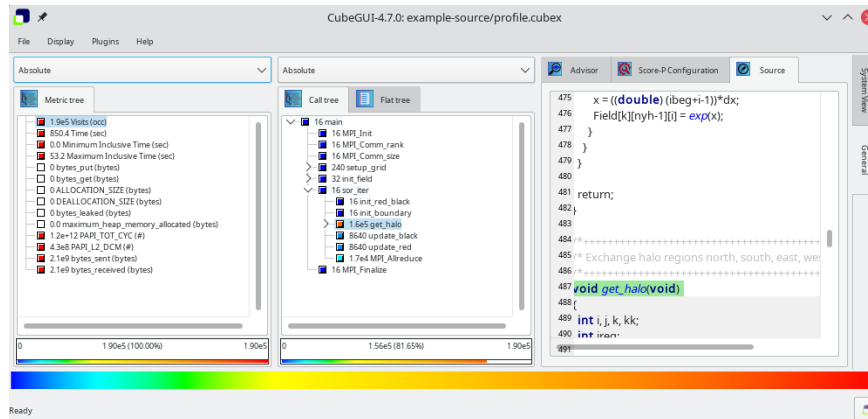


Figure 2.21: Source code viewer plugin

If CUBE doesn't find the file at its original location, a button to open a file dialog is displayed. The new location of the source files is saved in the global settings.

The context menu (right mouse button) shows following options:

1. **Copy** copies the selection to the clipboard
2. **Select All** selects the whole source file
3. **Show call site/function definition** if call site information is available, this item allows to switch between the call site and the function definition
4. **Find** adds an additional widget at the bottom of the viewer to search inside the source code
5. **Open in external editor** opens an external editor, after it is configured
6. **Reset user defined path** allows to select another path for the source code look-up (removes previous selection)

General options can be set in the plugin menu (Plugins->SourceCodeViewer).

1. **Set font** Change the default viewer font
2. **Read only** The default viewer mode is read only. You can enable editing here.
3. **Set external editor** This options allows to select one of the predefined external editors or define a new one.

2.7.8.1 Source Code Viewer Keyboard control

Control in read only mode:

Up Arrow	Move one line up
Down Arrow	Move one line down
Left Arrow	Scroll one character to the left (if horizontally scrollable)
Right Arrow	Scroll one character to the right (if horizontally scrollable)
Page Up	Move one (viewport) page up
PageDown	Move one (viewport) page down
Home	Move to the beginning of the text
End	Move to the end of the text
< scroll mouse-wheel >	Scroll the page vertically
Alt+< scroll mouse-wheel >	Scroll the page horizontally (if horizontally scrollable)
Ctrl+F	Find text
Ctrl+< scroll mouse-wheel >	Zoom the text
Ctrl+A	Select all text

Additionally for the read and write mode:

Left Arrow	Move one character to the left
Right Arrow	Move one character to the right
Backspace	Delete the character to the left of the cursor
Delete	Delete the character to the right of the cursor
Ctrl+C	Copy the selected text to the clipboard
Ctrl+Insert	Copy the selected text to the clipboard
Ctrl+K	Delete to the end of the line
Ctrl+V	Paste the clipboard text into text edit
Shift+Insert	Paste the clipboard text into text edit
Ctrl+X	Delete the selected text and copy it to the clipboard
Shift+Delete	Delete the selected text and copy it to the clipboard
Ctrl+Z	Undo the last operation
Ctrl+Y	Redo the last operation
Ctrl+Left arrow	Move the cursor one word to the left
Ctrl+Right arrow	Move the cursor one word to the right
Ctrl+Home	Move the cursor to the beginning of the text
Ctrl+End	Move the cursor to the end of the text
Hold Shift + some movement (e.g., Right arrow)	Select region

2.7.9 System Barplot Plugin

BARPLOT plugin is a CUBE plugin that plots vertical bar graph for the CUBE file which has iterations. Horizontal axis shows different iterations being compared and on vertical axis, several operations can be used to represent the value. The User can apply different metrics and call paths on the bar graph.

2.7.9.1 Basic Principles

As a start point, it should be mentioned that BARPLOT works only on a CUBE file that has iterations. For those files which have not, user would face the warning on the terminal : "No iterations for Barplot" and the plugin will not be shown.

By loading the plugin, on *system dimension*, the corresponding tab, *Barplot*, will be added. In the *Barplot* tab, the user can select different operations and assign desired color to them. Figure 2.22 displays a view of it.

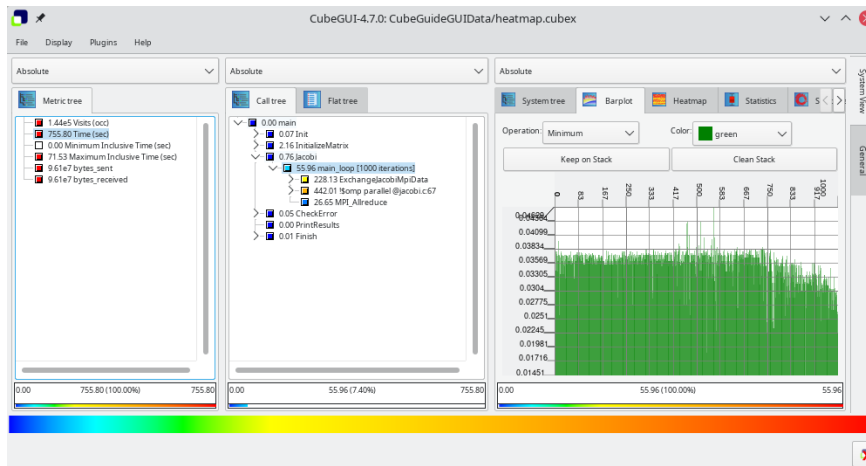


Figure 2.22: BARPLOT display window

User can select different metrics such as *Visits* and *Time*, by clicking on them in *metric dimension*. In addition, it is possible to get a BARPLOT for different *call paths* of iterations, via clicking on them. However, for *call paths* that are not located in iterations, like *input_in* in figure 2.23, no bar graph is displayed and user face the message "No data to display" on the window.

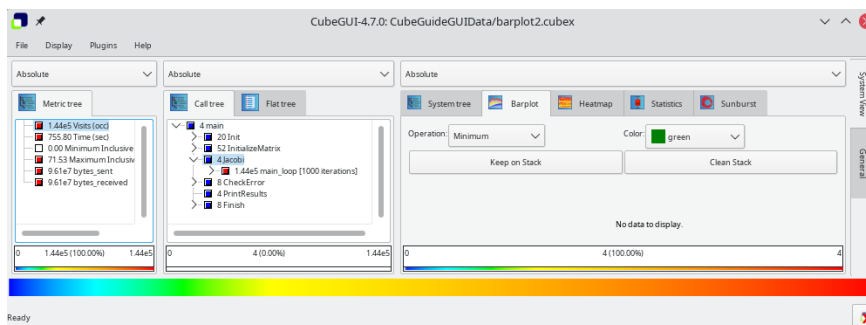


Figure 2.23: No data to display

Furthermore, the values on BARPLOT, can be evaluated in *Inclusive* and *Exclusive* manner. Therefore, user can easily collapse the tree on *call path* and click on the desired path to get the exclusive value of it.

Additionally, the exact calculated values can be seen by clicking left button of mouse on the desired position on the graph, a tooltip would display a value corresponding to the iteration.

In a situation that user needs to store the graph, it is just needed to do right click on a graph, and select "Save as image", then the *Save dialog* will be opened to specifying the path and name of the PNG file.

2.7.9.2 Toolbar

On the top of the Barplot space, there is a toolbar that allows user to specify the kind of an operation and its color(Figure 2.24).

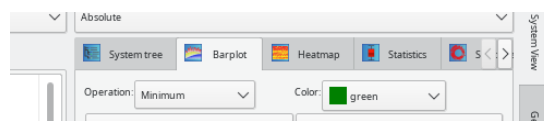


Figure 2.24: BARPLOT toolbar

By *operation* item, the user can select different operations, *Minimum*, *Maximum*, *Average*, *Median*, *1st Quartile* and *3rd Quartile* or the combination of *Maximum*, *Minimum* and *Average*. This provides the situation for the user to have different values for comparing at one time. These operations are done on all threads in each iterations. For instance, by *Minimum* operation, the minimum value among the existing threads for each iteration, is calculated and plotted. They are kind of statistical measurements.

Color item offers a color for an *operation*, however for each *operation*, a default color is assigned automatically. By changing the *operation*, corresponding color will be shown on *color* combo box. In a situation that different bar graphs are overlaid on each other, each graph will be shown by different color in order to distinguish various graphs.

In addition to above items, two buttons are also designed to manage the order of the bar graphs.

Keep on Stack: It is possible that user intends to compare different graphs by laying them on each other. For this matter, a push-button *keep on stack* is defined. Generally, by clicking on each *call path* or *metric*, a responding graph is replaced the previous one in the stack. In a situation, that the user intends to compare the next graph by the existing one, at one time, it is needed to click on the button *keep on the stack*, then the next graph will be added over the previous one, or in another words, it is overlaid on the last graph. If its values are less than the previous graph, user can see two graphs by different colors that help him/her in comparing, and in a situation that new values are greater than previous one, the new one will cover the previous with fresh color. Therefore, for keeping the top row of the stack, the user should click on the *keep the stack* button, otherwise the coming values will replace the last one.

Clean Stack: By clicking this button, all displayed graphs, are erased and the stack will be empty.

2.7.9.3 Menu Bar

Plugin menu offers the general function to enable or disable a plugin, and specific functions for each plugin. *Barplot plugin* provides the following functions in two areas, Measurement Customization and Threads Ruler Customization(Figure 2.25).

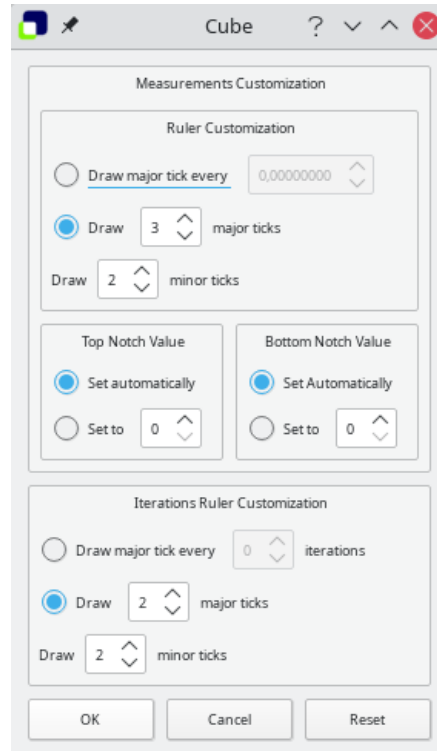


Figure 2.25: BARPLOT menu

Ruler Customization: User can modify the number of major and minor ticks of the ruler on vertical axis. For adjusting the major vertical ticks, user can set the drawing intervals or the number of ticks. By specifying the number of major ticks, the length of the vertical axis will be divided to the specified number and major ticks are drawn by length longer than minor ticks. Then in each divided length, if there is enough space, the specified number of minor ticks will be displayed. It is possible that the user set major ticks by interval. In order to do that, select the major ticks by interval option, and set the interval value. Therefore, after each interval, one major tick will be drawn.

Top Notch Value: The value of the top notch on a vertical axis can be altered by user as well as automatically. Therefore, due to scale issue, it can affect on the drawing of the graph.

Button Notch Value: The value of the button notch on a vertical axis can be altered by user as well as automatically. Therefore, due to scale issue, it can affect on the drawing of the graph.

Iterations Ruler Customization: User can modify the number of major and minor ticks of the ruler on horizontal axis. For adjusting the major horizontal ticks, user can set the

drawing intervals or the number of ticks. By specifying the number of major ticks, the width of the horizontal axis will be divided to the specified number and major ticks are drawn by length longer than minor ticks. Then in each divided length, if there is enough space, the specified number of minor ticks will be displayed. It is possible that the user set major ticks by interval of iterations. In order to do that, select the major ticks by interval option, and set the interval. Therefore, after each specified number of iterations, one major tick will be drawn.

2.7.10 System Heatmap Plugin

HEATMAP plugin is a CUBE plugin that represents the value of the thread in each iteration, as colors. The User can apply different metrics and call paths on heatmap graph.

2.7.10.1 Basic Principles

As a start point, it should be mentioned that HEATMAP works only on CUBE file that has iterations. For those files which have not, user would face the warning on the terminal : "No iterations for Heatmap" and the plugin will not be shown.

By loading the plugin, on *system dimension*, the corresponding tab, *Heatmap*, will be added. Figure 2.26 displays a view of it.

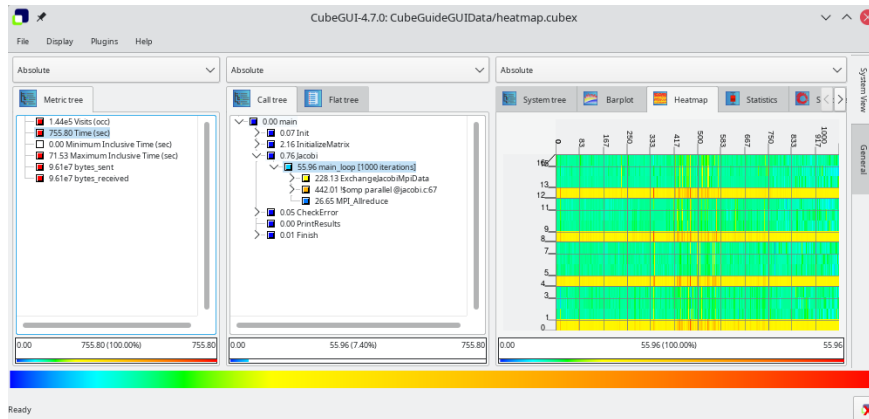


Figure 2.26: HEATMAP display window

User can select different metrics such as *Visits* and *Time*, by clicking on them in *metric dimension*. In addition, it is possible to get a HEATMAP for different *call paths* of iterations, via clicking on them. However, for *call paths* that are not located in iterations, like *input_in* in figure 2.27, no heatmap graph is displayed and user face the message "No data to display" on a window.

Furthermore, the values on HEATMAP, can be evaluated in *Inclusive* and *Exclusive* manner. Therefore, user can easily collapse the tree on *call path* and click on the desired path to get the exclusive value of it.

Additionally, the exact calculated values can be seen by clicking left button of mouse on

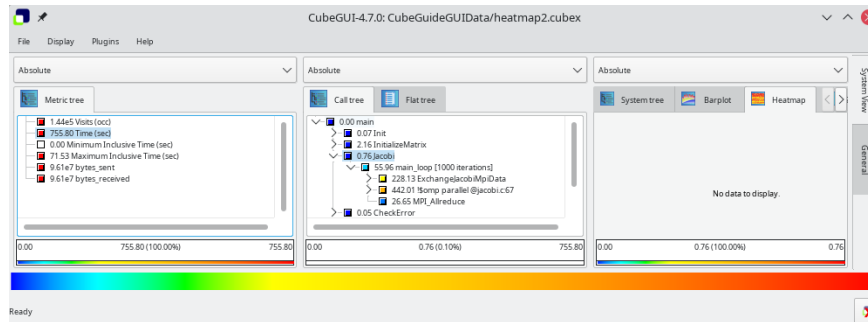


Figure 2.27: No data to display

the desired position on the graph, a tooltip would display a value corresponding to the iteration.

In a situation that user needs to store the graph, it is just needed to do right click on a graph, and select "Save as image", then the *Save dialog* will be opened to specifying the path and name of the PNG file.

2.7.10.2 Menu Heatmap

Plugin menu offers the general function to enable or disable a plugin, and specific functions for each plugin. *Heatmap plugin* provides the following functions in two areas, horizontal tick and vertical ticks (Figure 2.28).

Horizontal ticks: For adjusting the major horizontal ticks, user can set the drawing intervals or the number of ticks. By specifying the number of major ticks, the width of the horizontal axis will be divided to the specified number and major ticks are drawn by length longer than minor ticks. Then in each divided length, if there is enough space, the specified number of minor ticks will be displayed.

Also, it is possible that the user set major ticks by interval of iterations. In order to do that, select the major ticks by interval option, and set the interval. Therefore, after each specified number of iterations, one major tick will be drawn.

Vertical ticks: For adjusting the major vertical ticks, user can set the drawing intervals or the number of ticks. By specifying the number of major ticks, the length of the vertical axis will be divided to the specified number and major ticks are drawn by length longer than minor ticks. Then in each divided length, if there is enough space, the specified number of minor ticks will be displayed.

Also, it is possible that the user set major ticks by interval of threads. In order to do that, select the major ticks by interval option, and set the interval. Therefore, after each specified number of threads, one major tick will be drawn.

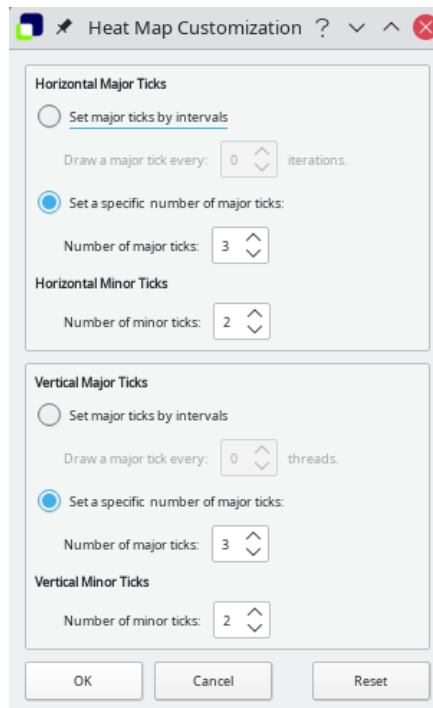


Figure 2.28: 'HEATMAP menu'

2.7.11 System Statistics Plugin

This plugin adds a statistics display tab next to the system-tree tab. It shows the value distribution either in a box plot or in a violin plot.

The box plot shows a box-and-whisker distribution of metric severity values for the currently active subset of system resources (typically threads). The active subset is changed via the combobox menu at the bottom of the pane, and the y-axis scale is adjusted via the display mode combobox at the top of the pane.

The vertical whisker ranges from the smallest value (minimum) and to the largest value (maximum), while the bottom and top of the box mark the lower quartile (Q1) and upper quartile (Q3). Within the box, the bold horizontal line represents the median (Q2) and the dashed line the mean value.

The violin plot is an alternative method of plotting statistical data, which additionally shows the distribution of the data. It is a box plot with a rotated kernel density plot on each side. The violin plot shows a thick black line for the median of the data, a dotted line for the mean, and red lines for quartiles.

To see the statistics as numeric values in a separate window, use <left-mouse click> inside the chart or use <right-mouse click> to show them in a tooltip. With <left-mouse drag>, an area is selected and the number of elements within this area is shown.

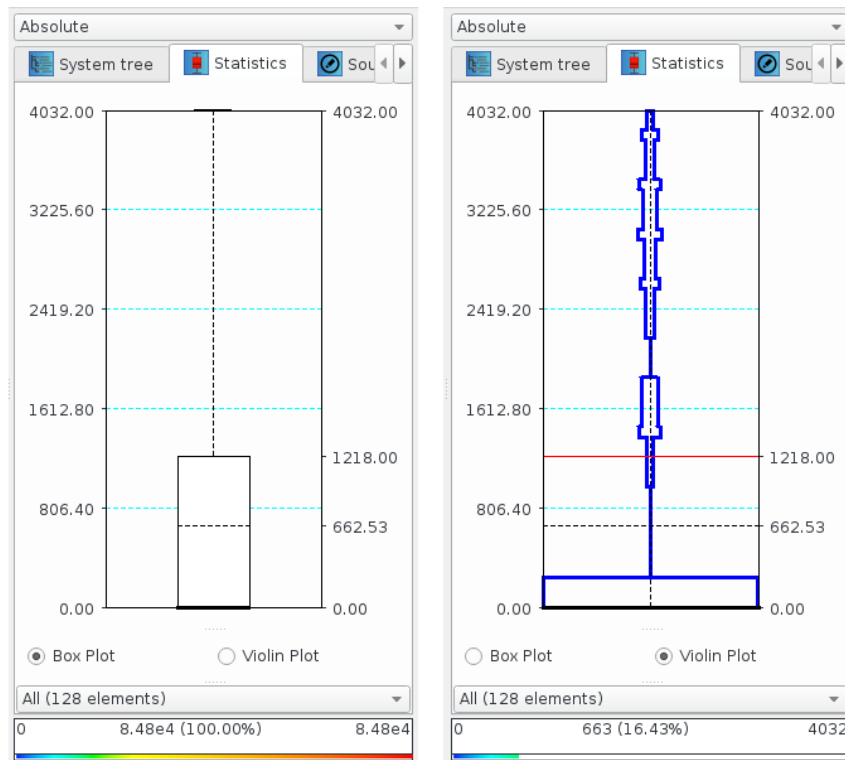


Figure 2.29: Statistical data shown in a box plot on the left side and violin plot on the right side

2.7.12 System Sunburst Plugin

This plugin adds a sunburst chart display tab to the system pane. The sunburst chart uses a radial tree to visualize the system-tree in a more compact form than the system-tree.

The sunburst chart and the system-tree are coupled, allowing the user to expand and collapse tree nodes in either widget with the changed state showing in the other widget. The arcs of the sunburst chart can be expanded and collapsed by <left-mouse click> on the outer edge of the arc. The edge is highlighted as shown in Figure 2.31 when hovering over it with the mouse cursor.

When expanded, the accumulated width of the child arcs is bounded by the width of their parent arc. To adjust the width of an arc, the user can expand its area by using Ctrl+<left-mouse drag> while clicking close to the side edge of the respective arc, as shown in Figure 2.32. The width of sibling arcs is adjusted automatically.

The standard interaction, next to expanding and collapsing arcs, is to rotate the sunburst chart, which is done via simple <left-mouse drag>. The user can zoom into and out of parts of the sunburst chart using the mouse wheel. The zoom behavior can be customized using the context menu. Furthermore, the user can move the visible canvas width Shift+<left-mouse drag>.

The user experience can be customized through flags set in the context menu via <right-

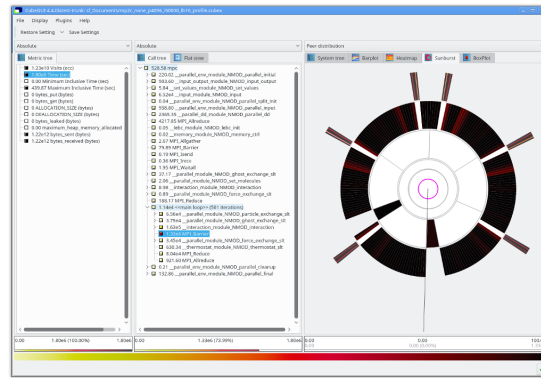


Figure 2.30: Expanded sunburst chart

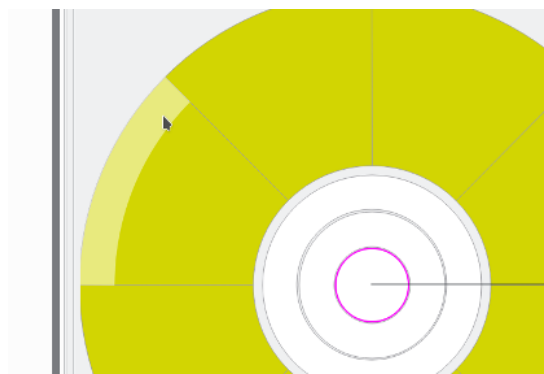


Figure 2.31: Arc edge highlighted when hovering over it

mouse click>. Furthermore, the context menu allows to reset specific or all interactions (e.g., rotation, arc width) with the chart to their default value.

The following table lists all available mouse interactions:

<left-mouse click>	<i>On arc:</i> Select arc <i>On arc edge:</i> Expand/collapse arc
<right-mouse click>	Context menu
<left-mouse drag>	Rotate chart
Ctrl+<left-mouse drag>	Change arc width
Shift+<left-mouse drag>	Move chart on canvas
< scroll mouse-wheel >	Zoom in/out

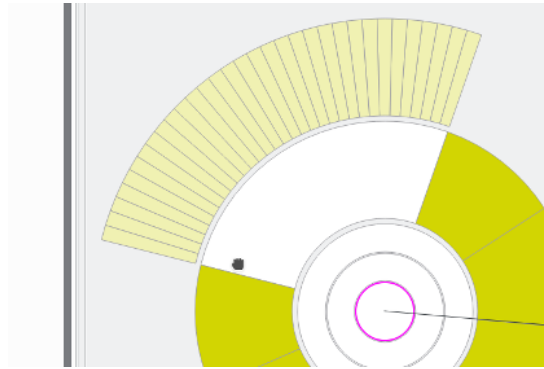


Figure 2.32: Adjusting the arc width using Ctrl+<left-mouse drag>

The following table lists all setting available via context menu:

Frame coloring	Adjust the frame color of arcs
Selection coloring	Adjust the frame color of selected arcs
Mark 0 degrees	Draw a line where the widget start the fan of arcs
Hide info tooltip	Do not show arc info in top left corner when hovering over a arc
Hide frame of small arcs	Avoid visual clutter by not drawing frames around thin arcs
Zoom towards the cursor	Instead of zooming into the chart origin, zoom towards the cursor
Invert zoom	Invert zoom direction when using the mouse wheel of track pad
Reset	Reset selected or all interactions (e.g., arc width, rotation,...) to default state

2.7.13 System Topology Plugin

In many parallel applications, each process (or thread) communicates only with a limited number of processes. The parallel algorithm divides the application domain into smaller chunks known as sub-domains. A process usually communicates with processes owning sub-domains adjacent to its own. The mapping of data onto processes and the neighborhood relationship resulting from this mapping is called *virtual topology*. Many applications use one or more virtual topologies specified as multi-dimensional Cartesian grids.

Another sort of topologies are *physical topologies* reflecting the hardware structure on which the application was run. A typical three-dimensional physical topology is given by the (hardware) nodes in the first dimension, and the arrangement of cores/processors on nodes in further two dimensions.

The CUBE display supports multi-dimensional Cartesian grids, where grids with high dimensionality can be sliced or folded down to two or three dimensions for presentation. If the currently opened cube file defines one or more such topologies, separate tabs are available for each using the topology name when one is provided. The topology display

shows performance data mapped onto the Cartesian topology of the application. The corresponding grid is specified by the number of dimensions and the size of each dimension. Threads/processes are attached to the grid elements, as specified by the CUBE file. Not all system items have to be attached to a grid element, and not every grid element has a system item attached. An example of a two-dimensional topology is shown on Figure 2.33. Note that the topology toolbar is enabled when a topology is available to be displayed.

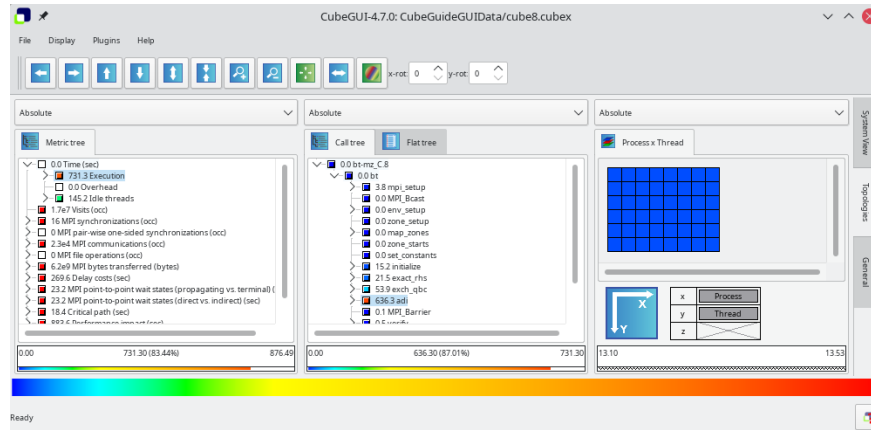


Figure 2.33: Topology Displays

The Cartesian grid is presented by planes stacked on top of each other in a three dimensional projection. The number of planes depends on the number of dimensions in the grid. Each plane is divided into tiles (typically shown as rombi). The number of tiles depends on the dimension size. Each tile represents a system resource (e.g., a process) of the application and has a coordinate associated with it.

The current value of each grid element (with respect to the selections on the left-hand side and to the current value mode) is represented by coloring the grid element. Coloring is based on a value scale from 0.0 to 100.0. Grid elements without having a system item attached to it are colored gray. See Section 2.5.2.1 (menu *Topology*) for further topology-specific coloring settings. For example, the upper topology in Figure 2.33 is drawn with black lines, the 2D topology in Figure 2.34 is drawn without lines.

If the selected system item occurs in the topology, it is marked by an additional frame and by additional lines at the side of the plane which contains the corresponding grid point, such that the selected item's position is also visible if the corresponding plane is not completely visible.

If zooming into planes is enabled, the plane containing the recently selected item is selected and the plane distance is adjusted to show this plane completely.

Selecting a collapsed tree in the system-tree selects all its children in the topology view.

Besides the functions offered by the topology toolbar (see 2.24), the following functionality is supported:

1. **Item selection:** You can change the current system selection by left-clicking on a grid element which has a system item assigned to it (resulting in the selection of that system item). Multiple items may be selected or deselected by holding down the Ctrl

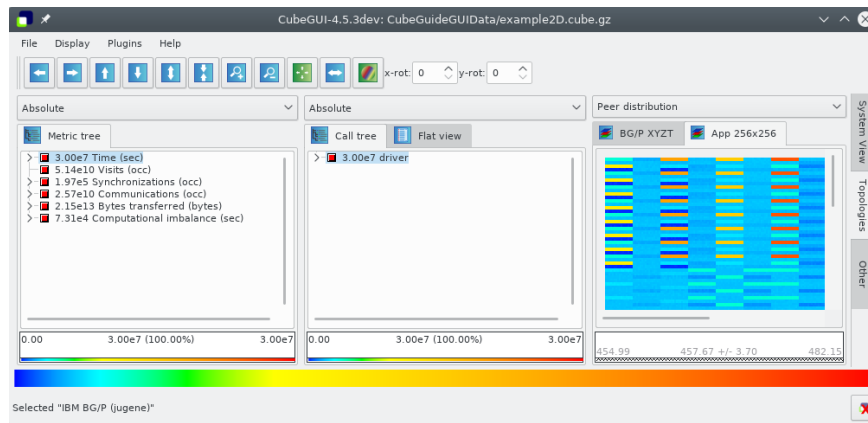


Figure 2.34: Topology Displays

key while clicking on an item.

2. **Info:** By right-clicking on a grid element, an information widget appears with information about the system item assigned to it. The information contains
 - the coordinate of the grid point in each topology dimension,
 - the hardware node to which the attached system item belongs to,
 - the system item's name,
 - its MPI rank,
 - its identifier,
 - and its value, followed by the percentage of this value on the scale between the minimal and maximal topology values.
3. **Rotation about the x and y axes:** can be done with left-mouse drag (click and hold the left-mouse button while moving the mouse).
4. **Increasing/decreasing the distance between the planes:** with Ctrl+<left-mouse drag>
5. **Moving the whole topology up/down/left/right:** with Shift+<left-mouse drag>

2.7.13.1 Topology mapping panel

If the number of topology dimensions is larger than three, the first three dimensions are shown and an additional control panel appears below the displayed topology. This panel allows rearranging topology dimensions on the x, y and z axes, as well as slicing or folding of higher dimensionality topologies for presentation in three or fewer dimensions.

Rearranging topology dimensions is achieved simply by dragging the topology dimension labels to the desired axis. When dragged on top of an existing topology dimension label, the two are exchanged.

When slicing, select up to three of the dimensions to display completely and choose one element of each of the remaining dimensions. The example in Figure 2.35 shows a

topology with 4 dimensions (32x16x32x4) labelled X, Y, Z and T. The first element of the 4th dimension (T) is automatically selected. By clicking on the button above the T, an index in this dimension from 0 to 3 can be chosen. If the index is set to *all*, the selection becomes invalid until an index of another dimension is selected.

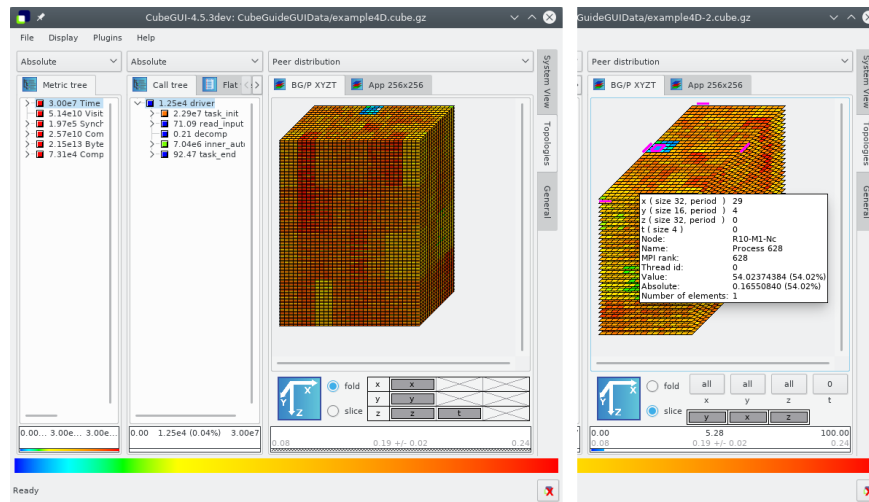


Figure 2.35: 4-dimensional example

Alternatively, the folding mode can be activated by clicking on the *fold* button. This mode is available for topologies with four to six dimensions and allows to display all elements by folding two dimensions into one. Every dimension appears in a box, with can be dragged into one of the three container boxes for the displayed dimensions x, y and z. In folding mode, the color of the inner borders is changed into gray. The black bordered rectangles show the element borders of each of the three displayed dimensions.

The right image in Figure 2.35 shows the folding of dimension Z with dimension T. One element with index (0,0,1,3) has been selected by clicking with the right mouse button into it. All elements inside the black rectangle around the selection belong to Z index one. The gray lines divide the rectangle into four elements which correspond to the elements of dimension T with index 0 to 3.

2.7.13.2 Topology plugin menu

- **Topology:** The topology menu offers the following functions related to the topology display described in Section 2.33 :
 1. **Item coloring:** Offers a choice how zero-valued system nodes should be colored in the topology display. The two offered options are either to use white or to use white only if all system leaf values are zero and use the minimal color otherwise.
 2. **Line coloring:** Allows to define the color of the lines in topology painting. Available colors are black, gray, white, or no lines.
 3. **Toolbar:** This menu item allows to specify if the topology toolbar buttons should be labeled by icons, by a text description, or if the toolbar should be hidden. For more information about the toolbar see Section 2.24 .

4. **Show also unused hardware in topology:** If not checked, unused topology planes, i.e., planes whose grid elements don't have any processes/threads assigned to, are hidden. Unused plane elements, if not hidden, are colored gray.
5. **Topology antialiasing:** If checked, anti-aliasing is used when drawing lines in the topologies.
6. **Zoom into current plane:** If checked, the plane containing the recently selected item is shown completely. It is never covered by a neighbor plane.

2.7.13.3 Toolbar

The system pane may contain topology displays if corresponding data is specified in the CUBE file. Basically, a topology display draws a two- or three-dimensional grid, in the form of some planes placed one above the other. Each plane consists of a two-dimensional grid of processes or threads.

The toolbar is enabled only if the system pane shows a topology display, and it offers functions to manipulate the display of the above grid planes. The toolbar can be labeled by icons, by text, or it can be hidden, see menu *Topology* \Rightarrow *Toolbar* in Section 2.5.2.1. The toolbar buttons have tool tips, i.e., a short description pops up if the toolbar is enabled and you move the mouse above a button.

The functions are the following, listed from the left to the right in the topology toolbar:

Move left Moves the whole topology to the left.

Move right Moves the whole topology to the right.

Move up Moves the whole topology upwards.

Move down Moves the whole topology downwards.

Increase plane distance Increase the distance between the planes of the topology.

Decrease plane distance Decrease the distance between the planes of the topology.

Zoom in Enlarge the topology.

Zoom out Scale down the topology.

Reset Reset the display. It scales the topology such that it fits into the visible rectangle, and transforms it into a default position.

Scale into window It scales the topology such that it fits into the visible rectangle, without transformations.

Set minimum/maximum values for coloring Similarly to the functions offered in the context menu of trees (see Section 2.5.2.4), you can activate and deactivate the application of user-defined minimal and maximal values for the color extremes, i.e., the values corresponding to the left and right end of the color legend. If you activate user-defined values for the color extremes, you are asked to define two values that should correspond to the minimal and to the maximal colors. All values outside of this interval will get the color gray. Note that canceling any of the input windows causes no changes in the coloring method. If user-defined min/max values are activated, the selected value information widget displays a (u) ' ' for user-defined" behind the minimal and maximal color values.

x-rotation Rotate the topology cube about the x-axis with the defined angle.

y-rotation Rotate the topology cube about the y-axis with the defined angle.

topolygy Allows to choose a topology from the list of defined topologies. If the topology is shown in the tab bar (default at startup), the corresponding tab will be selected. If the topology widget is detached, the widget will be shown on top of the main widget.

Using the grip at the left of the toolbar, it can be dragged to another position or detached entirely from the main window. The toolbar can also be closed after a right-click in the grip.

2.7.13.4 Topology keyboard and mouse control

<left-mouse click>	select item
<right-mouse click>	context information
Ctrl+<left-mouse drag>	increase plane distance
Shift+<left-mouse drag>	move topology
< scroll mouse-wheel >	zoom in/out
<left-mouse drag>	rotate topology
Up arrow	scroll one unit up
Down arrow	scroll one unit down
Page up	scroll one page up
Page down	scroll one page down

2.7.14 Tree Item Marker Plugin

This Plugin marks related items in the call, task and system tree. In the call tree, the correlation between a location group and its creator is shown. If an accelerator item is selected, this item and the related creator item are marked with a chain icon.

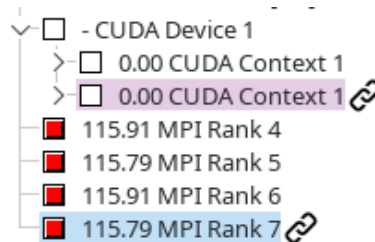


Figure 2.36: Related tree items in system tree

In the call tree, the relationship between an accelerator item and its call site is indicated by a marker. If the call site is located in the task tree, an additional marker is set to show that the item can be found in the next tab.

This plugins also adds an element to the context menu which allows to mark tree items manually. This is helpful to relocate the item after other selections have been done. The marked items are stored into the experiment specific settings.

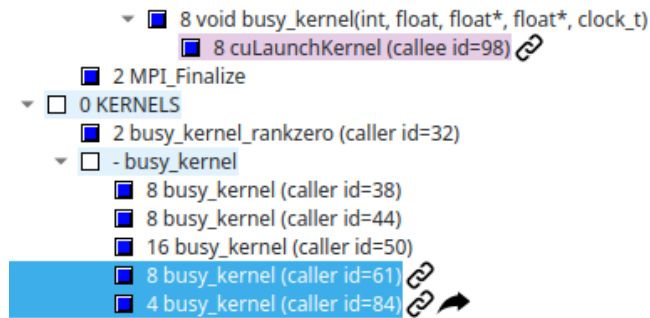


Figure 2.37: Related tree items in call tree

2.7.15 Launch Plugin

This Plugin allows one to create the connection between the performance measurement aka the cube file, and some another application via the simple assignment of the element in the profile, be it metric- or callpath- element and the to-be-executed command.

This connection is always established individually to the performance measurement. If a more general connection is needed, an additional CubeGUI plugin has to be developed.

2.7.15.1 .launch File

To achieve this one needs to create the file with the same name as the cube file and with the extension ".launch". For example, profile.cubex -> profile.launch, summary.cubex -> summary.launch etc.

This has to be located in the same directory as the corresponding .cubex file and can be created manually (also postfactum) or automatically.

2.7.15.1.1 Structure of the .launch file The content of the file has three main sections: INIT, MAIN, FINALIZE

- **INIT** - defines the command, which has to be executed before other commands of the launch get active. This might be used to initialize a third-party application. This command won't be executed automatically, but is only available in the plugin menu "Plugins".

Syntax:

```
[INIT]
<init menu title>
<command> <parameter>
```

The **init menu title** defines the menu item, which the user would need to select to execute the **command**

- **FINALIZE** - this command, if defined, is automatically executed when the cube file is closed

Syntax:

```
[FINALIZE]
<command> <parameter>
```

This section is "mute", means the **command** is executed without the additional (except "Close Cube") actions from the user.

- **MAIN** - this section defines commands for individual tree elements.

Syntax:

```
...
<metric uniq name>
<menu title in metric tree>
<metric tree command> <parameter>
- cnode <cnode id>
<menu title in call tree for given id>
<call tree command> <parameter>
...
```

In this section one defines metric wise the series of the entries. This means that for the selected metric **metric unit name**, the command **metric tree command** is defined, which is executed when **menu title in metric tree** is selected in the context menu in the metric tree. The prefix "- cnode" is used to define an ID (**cnode id**) of the call path, for which the command **call tree command** is specified. This command is executed when the menu item **menu title in call tree for given ID** is selected in the context menu of the call path.

One can specify multiple - cnode <cnode id> for the selected metric <metric uniq name> or one can specify <metric uniq name> for every cnode individually. The sequence of the entries is irrelevant.

One can choose instead of <metric uniq name> the placeholder "**", which makes the command available for every call path.

NOTICE Execution of the **command** is done in the "current directory", which is the directory in which "./cube" has been executed. Every command should be executable. This means that the executable is either found via the PATH variable or must be specified with an absolute path.

2.7.15.1.2 Parameters in the .launch file To establish the meaningful connection between the performance profile and the To-be-executed command one can specify the paramteres (placeholders) for the command. These will be replaces by the corresponding values. Available placeholders always start with the symbol "%" and are :

- **%f** - stands for the path to the .cubex file
- **%mn** - stands for the name of the selected metric
- **%mi** - stands for the id of the selected metric
- **%me** - stands for the expansion state of the selected metric
- **%m** - stands for the value in the metric tree of the selected metric
- **%mn** - defines the name of the selected call path
- **%cn** - stands for the name of the selected call path
- **%ci** - stands for the id of the selected call path
- **%ce** - stands for the expansion state of the selected call path

- **%c** - stands for the value in the call tree of the selected call path
- **%cn** - defines the name of the selected call path

2.7.15.1.3 Example of the .launch file

```
[INIT]
Init Visualisation
initialize.sh %f START

time
- cnode 4
display_timing_cnode.sh %cn %ci $ce %c

visits
Display calls in ParaView
display_calls_metric.sh %mn %mi $me %m
- cnode 3
Display this call in ParaView
display_calls_cnode.sh %cn %ci $ce %c
- cnode 10
Display this call in ParaView
display_calls_cnode.sh %cn %ci $ce %c

[FINALIZE]
finalize.sh %f DONE
```

.launch file doesn't support comments.

2.8 Other Features

2.8.1 Features enabled through statistic files

In this section we will explain two features – namely the display of statistical information about performance patterns which represent performance problems and the display of the most severe instances of these patterns in a trace browser – which both are only available if a statistic file for the currently opened CUBE file is present. Currently, such a statistic file can be generated by the SCOUT analyzer [?]. The file format of statistic files is described in the Appendix [124.1](#).

For CUBE to recognize the statistic file, it must be placed in the same directory as the CUBE file. The basename of the statistic file should be identical to that of the CUBE file, but with the suffix `.stat`. For example, when the CUBE file is called `trace.cubex`, the corresponding statistic file is called `trace.stat`.

2.8.2 Statistical information about performance patterns

If a statistic file is provided, you can view statistical information about one or multiple patterns (for example in order to compare them). This is done by selecting the desired

metrics in the metric-tree and then selecting the *Statistics* menu item in the context menu. This brings up the box plot window as shown in Figure 2.38.

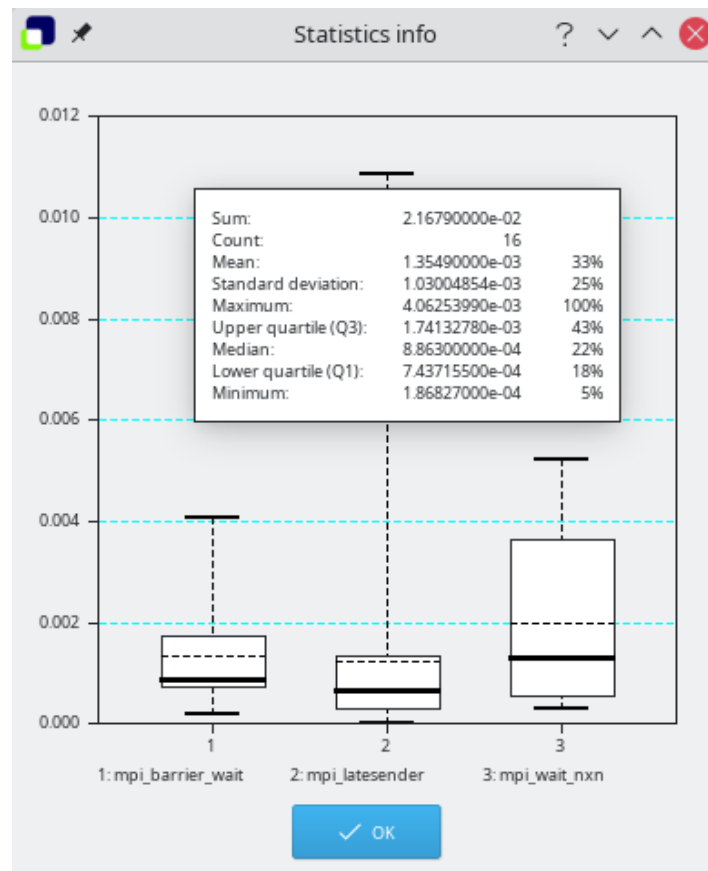


Figure 2.38: Screenshot of a box plot as shown by CUBE displaying statistical information about the selected patterns. The tooltip shows the exact values of the statistics.

The box plot shows a graphical representation of the statistical data of the selected patterns. The slender black lines on the top and the bottom designate the maximum and the minimum measured severity of the pattern, respectively. The lower and the upper borders of the white box indicate the values of the 25% and 75% quantile. The thick line inside the box represents the median of the values, while the dashed line indicates the mean.

There are two ways of interacting with the box plot. You can zoom to a certain interval on the y-axis by clicking on a position with the height of the desired maximal or minimal value and by consecutively dragging the mouse to a position with the height of the corresponding other extreme value. You can reset the view (i.e., to undo all zooming) by clicking the middle mouse button somewhere on the box plot.

If you are interested in more precise values for the severity statistics of a certain metric, you can click with the left mouse button somewhere in the column of the desired metric, which will yield a small window (as shown in the top right corner of Figure 2.38) displaying the exact values of the statistics. Clicking with the right mouse button shows the information in a tooltip.

2.8.3 Display of most severe pattern instances using a trace browser

If a statistic file also contains information about the most severe instances of certain patterns, CUBE can be connected to a trace browser (currently only Vampir [? ?] is supported) in order to view the state of the program being analyzed at the time this most severe pattern instance occurred. For collective operations, the most severe instance is the one with the largest *sum* of the waiting times of all processes, which is not necessarily the one with the largest maximal waiting time of each individual process.

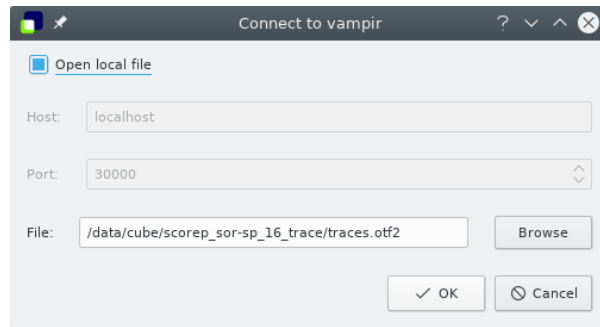


Figure 2.39: The dialog windows for a connection to a trace browser e.g. Vampir

To use this feature, you first have to connect to a trace browser by using the *Connect to* menu item of the *Vampir Plugin* submenu of the *Plugin* menu. This will open one of the two dialog windows shown below.

For Vampir, you have to specify the host name and port of the Vampir server you want to connect to and the path of the trace file you want to load. This will launch the Vampir client (if it is correctly configured) and load the specified trace file. To configure Vampir so that it can be started automatically by CUBE, a service file `com.gwt.vampir.service`, describing the path to your Vampir client executable must be placed under `(/usr/share/dbus-1/service)` or `${HOME}/.local/share/dbus-1/services`. This service file must be exactly as shown below, with the exception that `Exec` should point to your Vampir client executable.

```
[D-BUS Service]
Name=com.gwt.vampir
Exec=/private/utils/bin/vng
```

An example of the `com.gwt.vampir.service` file

Once CUBE is connected to a trace browser you can select the *Max severity in trace browser* menu item of the metric-tree so that all connected trace browsers are zoomed to the (globally) most severe instance of the selected pattern.

A more sophisticated feature of CUBE is the ability to zoom to the most severe instance of a pattern in a selected call path. This can be done by selecting a metric in the metric-tree

which will highlight the most severe call paths in the call-tree. You can then use the context menu of the call tree to select the *Max severity in trace browser* menu item which will then zoom all connected trace browsers to the most severe instance of the selected pattern with respect to the chosen call path (see Figure 2.40).

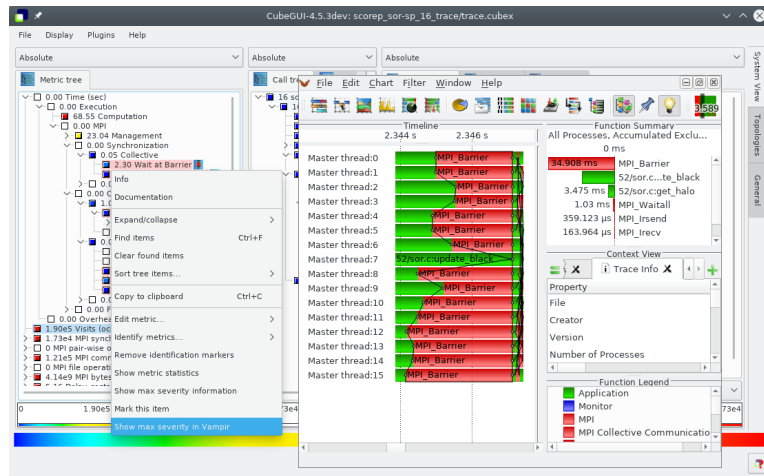


Figure 2.40: Context menu called on the metric "Wait at Barrier", showing the maximum severity in trace browser, which results in the location of the worst instance shown in the timeline display of Vampir.

2.8.3.1 Troubleshooting

1. In some D-BUS configurations Vampir does not start automatically. In this case it might solve the problem to have Vampir already running (with explicitly enabled D-BUS subsystem)

```
user@host: vampir --dbus&
```

2. On some HPC system it might be helpful to extend your environment. Add to your .bashrc file following code snippet:

```
## test for an existing bus daemon, just to be safe
if test -z "$DBUS_SESSION_BUS_ADDRESS" ; then
    ## if not found, launch a new one
    eval `dbus-launch --sh-syntax`
    echo "D-Bus per-session daemon address is: $DBUS_SESSION_BUS_ADDRESS"
fi
```

2.8.4 Synchronization of several cube instances

The current state of a cube instance (selections, expanded tree items, ...) can be synchronized with other cube instances on the same or on different machines. The synchronization function uses the clipboard to exchange data, so no network protocol is required. Synchronization can be useful e.g. for following tasks:

- Comparison of several runs of the same program with different number of processes or threads.
- Examination of different metrics at the same time.

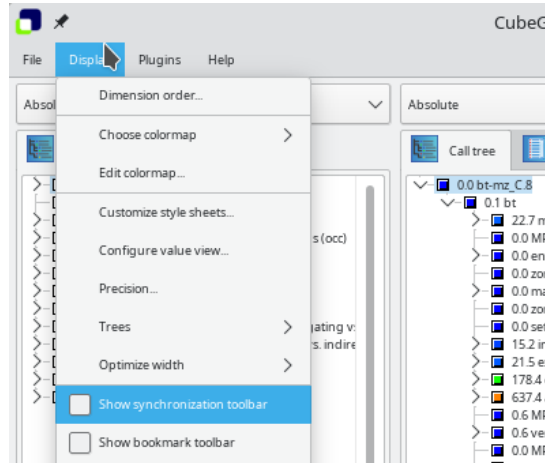


Figure 2.41: Enable Synchronization

To enable Synchronization, the corresponding toolbar has to be enabled (Figure 2.41). Press the toolbar button with the red outgoing arrow to enable sending of status information. The current state is sent when the button is activated and after every change while the button is checked. To receive status information press the button with the white incoming arrow. If activated, cube listens for changed status information.

Tree items are identified by their label, not by the position in the tree. This might lead to unexpected selections, if a tree item has multiple children with the same label.

With the "Synchronize state" menu, you can select the information that is sent and received. By default, this is the state of the trees. If you want to show different metrics in each cube instance, but synchronize the selected callpath and system-tree, you have to disable "Metric tree" (Figure 2.42).

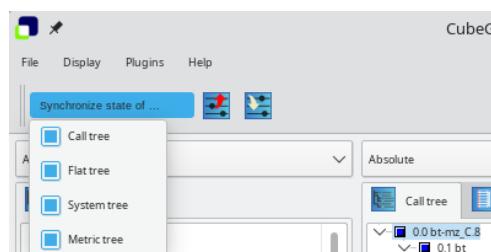


Figure 2.42: Synchronization toolbar

2.9 Keyboard and mouse control

Shift+F1	Help: What's this?
Ctrl+O	Shortcut for menu File ⇒ Open
Ctrl+W	Shortcut for menu File ⇒ Close
Ctrl+Q	Shortcut for menu File ⇒ Quit
<left-mouse click>	<i>over menu/tool bar</i> : activate menu/function <i>over value mode combo</i> : select value mode <i>over tab</i> : switch to tab <i>in tree</i> : select/deselect/expand/collapse items
<right-mouse click>	<i>in tree</i> : context menu
Ctrl+<left-mouse click>	<i>in tree</i> : multiple selection/deselection
<left-mouse drag>	<i>over scroll bar</i> : scroll
Up arrow	<i>in tree</i> : move selection one item up (+Shift: multiple selection)
Down arrow	<i>in tree</i> : move selection one item down (+Shift: multiple selection)
Left arrow	<i>in scroll area</i> : scroll to the left
Right arrow	<i>in scroll area</i> : scroll to the right
Ctrl+F	find tree item
F3	move to next search result
Shift+F3	move to previous search result

For keyboard shortcuts in different plugins, see the corresponding sections:

- [2.7.8](#)
- [2.7.13.4](#)

3 Cube Advisor Plugin

Advisor is a standard plugin and is available as long as the measurement contains a *Time* metric. The main goal of the Advisor plugin is to provide a user a fast access to the various performance evaluations of the performance of their HPC application.

3.1 Getting Started with Advisor

If measurement contains metric *Time*, CubeGUI will enable the Advisor plugin in the "General" tab in the plugins section.

Some [3.2](#) can be disabled due to missing performance properties, e.g. missing PAPI counters. In such cases one potential solution is to merge original measurement with measurement which includes missing properties and run analysis again. Measurement merging can be done with one of the context-free plugins [2.7.2.3](#) or [2.7.2.2](#).

Moreover, some assessments are hidden (e.g. [3.2.2](#) and [3.2.5](#)) and can be available in "expert" mode (see [2.3](#)).

3.2 Supported Assessments

Advisor supports various performance assessments, such as

- [3.2.1](#)
- [3.2.2](#)
- [3.2.3](#)
- [3.2.4](#)
- [3.2.5](#)

3.2.1 Only-MPI Assessment

Attempting to optimize the performance of a parallel code can be a daunting task, and often it is difficult to know where to start. For example, we might ask if the way computational work is divided is a problem? Or perhaps the chosen communication scheme is inefficient? Or does something else impact performance? To help address this issue, POP has defined a methodology for analysis of parallel codes to provide a quantitative way of measuring relative impact of the different factors inherent in parallelization. This article introduces these metrics, explains their meaning, and provides insight into the thinking behind them.

A feature of the methodology is, that it uses a hierarchy of [3.2.1](#), each metric reflecting a common cause of inefficiency in parallel programs. These metrics then allow a comparison

of the parallel performance (e.g. over a range of thread/process counts, across different machines, or at different stages of optimization and tuning) to identify which characteristics of the code contribute to the inefficiency.

The first step to calculating these metrics is to use a suitable tool (e.g. Score-P or Extrae) to generate trace data whilst the code is executed. The traces contain information about the state of the code at a particular time, e.g. it is in a communication routine or doing useful computation, and also contains values from processor hardware counters, e.g. number of instructions executed, number of cycles.

The [3.2.1](#) are then calculated as efficiencies between 0 and 1, with higher numbers being better. In general, we regard efficiencies above 0.8 as acceptable, whereas lower values indicate performance issues that need to be explored in detail. The ultimate goal then for POP is rectifying these underlying issues by the user. Please note, that [3.2.1](#) can be computed only for **inclusive** callpaths, as they are less meaningful for exclusive callpaths. Furthermore, [3.2.1](#) are not available in "Flat view" mode.

The approach outlined here is applicable to various parallelism paradigms, however for simplicity the [3.2.1](#) presented here are formulated in terms of a distributed-memory message-passing environment, e.g., MPI. For this the following values are calculated for each process from the trace data: time doing useful computation, time in communication, number of instructions & cycles during useful computation. Useful computation excludes time within the overhead of parallel paradigms ([117.1](#)).

At the top of the hierarchy is **Global Efficiency (GE)**, which we use to judge overall quality of parallelization. Typically, inefficiencies in parallel code have two main sources:

- Overhead imposed by the parallel nature of a code
- Poor scaling of computation with increasing numbers of processes

and to reflect this we define two sub-metrics to measure these two inefficiencies. These are the **Parallel Efficiency** and the **Computation Efficiency**, and our top-level GE metric is the product of these two sub-metrics:

$$GE = \text{102.1} \cdot \text{119.1}$$

Note:

Computation Efficiency can be computed only at scale with multiple measurements and currently is not supported by Advisor.

We sincerely hope this methodology will be adopted by our users and others and will form part of the project's legacy. If you would like to know more about the POP metrics and the tools used to generate them please check out the rest of the Learning Material on our website, especially the document on POP Metrics

3.2.2 Multiplicative Hybrid Assessment

Note:

[3.2.2](#) is available only in "expert" mode (see [2.3](#)).

This is one approach to extend POP metrics for hybrid (MPI+OpenMP) applications. In this approach [4.1](#) split into two components:

- [6.1](#) shows the inefficiencies on MPI level, and can be broken down into [8.1](#) and [10.1](#)
- [16.1](#) shows the inefficiencies on OpenMP level, and can be broken down into [18.1](#) and [19.1](#)

In this analysis **Parallel Efficiency (PE)** can be computed as a product of these two sub-metrics:

$$PE = 6.1 \cdot 16.1$$

3.2.3 Additive Hybrid Assessment

This is one approach to extend POP metrics for hybrid (MPI+OpenMP) applications. In this approach [29.1](#) split into two components:

- [31.1](#) shows the inefficiencies on MPI level, and can be broken down into [39.1](#) and [33.1](#).
- [41.1](#) shows the inefficiencies on OpenMP level, and can be broken down into [43.1](#) and [44.1](#)

In this analysis **Parallel Efficiency (PE)** can be computed directly or as a sum of these two sub-metrics minus one:

$$PE = 31.1 + 41.1 - 1$$

This scheme has two advantages: each hybrid efficiency measures absolute cost of the issue(s) under consideration, i.e. relative to the runtime; additive method gives more freedom in defining child metrics.

3.2.4 BSC Hybrid Assessment

This is one approach to extend POP metrics for hybrid (MPI+OpenMP) applications. It provides three types of efficiencies, i.e.:

- [54.1](#) reveals the inefficiency in processes and threads utilization and can be broken down into [56.1](#) and [58.1](#)
- [60.1](#) reveals the inefficiency in MPI processes and can be broken down into [62.1](#) and [64.1](#)
- [66.1](#) reveals the inefficiency in OpenMP threads and can be broken down into [68.1](#) and [70.1](#)

3.2.5 JSC Hybrid Assessment

Note:

[3.2.5](#) is available only in "expert" mode (see [2.3](#)).

This is JSC spin-off of POP metrics for hybrid (MPI+OpenMP) applications. In this approach there are two sets of metrics, i.e.:

- metrics describing inefficiencies in MPI: [80.1](#) and [82.1](#)

- metrics describing inefficiencies in OpenMP: [88.1](#) and [90.1](#) and [92.1](#)

There are two peculiarities for this model

- this model considers only MPI and OpenMP and doesn't evaluate parallel behaviour in general
- additionally to a single metric user can explore statistics over metrics (for some metrics), i.e. MIN/AVG/MAX values, which can help to identity execution anomalies

4 AdvisorPOPHybridTestsParallel_efficiency

4.1 Parallel Efficiency

Parallel Efficiency (PE) reveals the inefficiency in processes and threads utilization. These are measured with **Process Efficiency** and **Thread Efficiency**, and PE can be computed directly or as a product of these two sub-metrics:

$$PE = \frac{avg(comp)}{max(runtime)}$$

$$= 6.1 \cdot 16.1$$

5 AdvisorPOPHybridTestsMissing_parallel_efficie

5.1 Missing Parallel Efficiency?

[4.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

6 AdvisorPOPHybridTestsProcess_efficiency

6.1 Process Efficiency

Process Efficiency completely ignores thread behavior, and evaluates process utilization via two components:

- Workload across processes
- Communication across processes

These two can be measured with **Computation Load Balance** and **Communication Efficiency** respectively. **Process Efficiency** can be computed directly or as a product of these two sub-metrics:

$$PE = \frac{avg(time\ in\ OpenMP) + avg(serial\ computation)}{max(runtime)}$$

$$= 8.1 \cdot 10.1.$$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

7 AdvisorPOPHybridTestsMissing_process_efficie

7.1 Missing Process Efficiency?

[6.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

8 AdvisorPOPHybridTestsLoad_balance

8.1 Computation Load Balance

Computation Load Balance can be evaluated directly by following formula:

$$\text{Computation Load Balance} = \frac{\text{avg}(\text{time in OpenMP}) + \text{avg}(\text{serial computation})}{\text{max}(\text{time in OpenMP} + \text{serial computation time})}$$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

9 AdvisorPOPHybridTestsMissing_load_balance

9.1 Missing Computation Load Balance?

[8.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

10 AdvisorPOPHybridTestsCommunication_efficie

10.1 MPI Communication Efficiency

MPI Communication Efficiency (CommE) can be evaluated directly by following formula:

$$CommE = \frac{\max(\text{time in OpenMP} + \text{serial computation time})}{\max(\text{runtime})}$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using [12.1](#) and [14.1](#). Combination of these two sub-metrics gives us **Communication Efficiency**:

$$CommE = \text{12.1} \cdot \text{14.1}$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

11 AdvisorPOPHybridTestsMissing_communicatio

11.1 Missing Communication Efficiency?

[10.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

12 AdvisorPOPHybridTestsSerialisation_efficiency

12.1 Serialisation Efficiency

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

13 AdvisorPOPHybridTestsMissing_serialisation_c

13.1 Missing Serialisation Efficiency?

[12.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

14 AdvisorPOPHybridTestsTransfer_efficiency

14.1 Transfer Efficiency

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \text{maximum across processes} \left(\frac{\text{total runtime on ideal network}}{\text{maximum across processes}(\text{total measured runtime})} \right)$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

15 AdvisorPOPHybridTestsMissing_transfer_effic

15.1 Missing Transfer Efficiency?

[14.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

16 AdvisorPOPHybridTestsThread_efficiency

16.1 Thread Efficiency

Thread Efficiency considers two sources of inefficiency:

- Serial computation on the master outside OpenMP, i.e. reflects Amdahl's law
- Inefficiencies within threads, e.g. serialisation across threads

These two can be measured with **Amdahl's Efficiency** and **OpenMP region Efficiency** respectively. **Thread Efficiency** can be computed directly or as a product of these two sub-metrics:

$Thread\ Efficiency = \frac{avg(computation\ time)}{avg(time\ in\ OpenMP) + avg(serial\ computation)}$
$= 18.1 \cdot 19.1$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

17 AdvisorPOPHybridTestsMissing_thread_efficie

17.1 Missing Thread Efficiency?

[16.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

17.2 Missing Amdahl's Efficiency?

[18.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

18 AdvisorPOPHybridTestsAmdahl_efficiency

18.1 Amdahl's Efficiency

Amdahl's Efficiency indicates serial computation and can be computed as follows:

$$\text{Amdahl's Efficiency} = \frac{\text{avg}(\text{computation time})}{\text{avg}(\text{time in useful computation within OpenMP}) + \text{avg}(\text{serial computation})}$$

Where average serial computation computed as **weighted arithmetic mean**. If number of threads is equal across processes average serial computation can be computed as **ordinary arithmetic mean**.

19 AdvisorPOPHybridTestsOmpRegion_efficiency

19.1 OpenMP Region Efficiency

OpenMP Region Efficiency indicates inefficiencies within threads, and can be computed as follows:

$$\text{OpenMP Region Efficiency} = \frac{\text{avg}(\text{time in useful computation within OpenMP}) + \text{avg}(\text{serial computation})}{\text{avg}(\text{time in OpenMP}) + \text{avg}(\text{serial computation})}$$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

20 AdvisorPOPHybridTestsMissing_omp_region_e

20.1 Missing OpenMP Region Efficiency?

[19.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

21 AdvisorPOPHybridTestsIpc

21.1 IPC (only computation)

IPC indicates number of instructions executed by CPU per clock cycle. The higher the value the better the CPU performance. It is computed as the ratio of total instructions in user code to total cycles spent in user code.

- If PAPI counters are available, use **PAPI_TOT_INS / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use **instructions / cycles**.

22 AdvisorPOPHybridTestsMissing_ipc

22.1 Missing IPC?

21.1 metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_TOT_INS** and **PAPI_TOT_CYC**,
- or the Perf counters **instructions** and **cycles**.

How to do it see Score-P manual

23 AdvisorPOPHybridTestsStalled_resources

23.1 Stalled resources (only computation)

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources.

- If PAPI counters are available, use **PAPI_RES_STL / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use one of
 - **(stalled-cycles-backend+stalled-cycles-frontend) / cycles**,
 - **stalled-cycles-backend / cycles**,
 - **stalled-cycles-frontend / cycles**,depending on what is available.

24 AdvisorPOPHybridTestsMissing_stalled_resou

24.1 Missing "Resource stall cycles"?

[23.1](#) metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_RES_STL** and **PAPI_TOT_CYC**,
- or the Perf counter **cycles** and at least one of **stalled-cycles-backend**, **stalled-cycles-frontend** .

How to do it see Score-P manual

25 AdvisorPOPHybridTestsNoWaitINS_efficiency

25.1 Instructions (only computation)

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the [27.1](#).

- If PAPI counters are available, use **PAPI_TOT_INS**.
- Otherwise, if Perf counters are available, use **instructions**.

26 AdvisorPOPHybridTestsMissingNoWaitINS_eff

26.1 Missing Instructions (only computation)?

25.1 metric is available only, if the measurement has collected

- either the PAPI counter **PAPI_TOT_INS**,
- or the Perf counter **instructions**.

How to do it see Score-P manual

27 AdvisorPOPHybridTestsComputationTime

27.1 Computation time

Computation time indicated total time spend in the computation call path **without** MPI, OpenMP, POSIX threads, std::threads, CUDA, OpenCL, OpenACC, SHMEM. With another words, it is the user code.

28 AdvisorPOPHybridTestsMissingComputationT

28.1 Missing Computation time?

[27.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

29 AdvisorPOPHybridAddTestsParallel_efficiency

29.1 Parallel Efficiency

Parallel Efficiency (PE) reveals the inefficiency in processes and threads utilization. These are measured with **Process Efficiency** and **Thread Efficiency**, and PE can be computed directly or as a sum of these two sub-metrics minus one:

$PE = \frac{avg(computation\ time)}{max(runtime)}$
= 31.1 + 41.1 - 1

30 AdvisorPOPHybridAddTestsMissing_parallel_e

30.1 Missing Parallel Efficiency?

[29.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

31 AdvisorPOPHybridAddTestsProcess_efficiency

31.1 Process Efficiency

Process Efficiency completely ignores thread behavior, and evaluates process utilization via two components:

- Workload across processes
- Communication across processes

These two can be measured with **Computation Load Balance** and **Communication Efficiency** respectively. **Process Efficiency** can be computed directly or as a sum of these two sub-metrics minus one:

$Process\ Efficiency = \frac{avg(time\ in\ OpenMP) + avg(serial\ computation)}{max(runtime)}$
$= 39.1 + 33.1 - 1$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

32 AdvisorPOPHybridAddTestsMissing_process_e

32.1 Missing Process Efficiency?

[31.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

33 AdvisorPOPHybridAddTestsCommunication_efficiency

33.1 MPI Communication Efficiency

MPI Communication Efficiency (CommE) can be evaluated directly by following formula:

$$CommE = \frac{\max(\text{time in OpenMP} + \text{serial computation time})}{\max(\text{runtime})}$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using [35.1](#) and [37.1](#). Combination of these two sub-metrics gives us **Communication Efficiency**:

$$CommE = 35.1 \cdot 37.1$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

34 AdvisorPOPHybridAddTestsMissing_communic

34.1 Missing Communication Efficiency?

[33.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

35 AdvisorPOPHybridAddTestsSerialisation_effic

35.1 Serialisation Efficiency

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

36 AdvisorPOPHybridAddTestsMissing_serialisat

36.1 Missing Serialisation Efficiency?

[35.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

37 AdvisorPOPHybridAddTestsTransfer_efficiency

37.1 Transfer Efficiency

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{\text{maximum across processes}(\text{total runtime on ideal network})}{\text{maximum across processes}(\text{total measured time})}$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

38 AdvisorPOPHybridAddTestsMissing_transfer_e

38.1 Missing Transfer Efficiency?

[37.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

39 AdvisorPOPHybridAddTestsLoad_balance

39.1 Computation Load Balance

Computation Load Balance can be evaluated directly by following formula:

$$\text{Computation Load Balance} = \frac{\max(\text{runtime}) - \max(\text{time in OpenMP} + \text{serial computation time}) + \text{avg}(\text{time in OpenMP})}{\max(\text{runtime})}$$

Where average time in OpenMP and average serial computation are computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as **ordinary arithmetic mean**.

40 AdvisorPOPHybridAddTestsMissing_load_bala

40.1 Missing Computation Load Balance?

[39.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

41 AdvisorPOPHybridAddTestsThread_efficiency

41.1 Thread Efficiency

Thread Efficiency considers two sources of inefficiency:

- Serial computation on the master outside OpenMP, i.e. reflects Amdahl's law
- Inefficiencies within threads, e.g. serialisation across threads

These two can be measured with **Amdahl's Efficiency** and **OpenMP region Efficiency** respectively. **Thread Efficiency** can be computed directly or as a sum of these two sub-metrics minus one:

$$\text{Thread Efficiency} = \frac{\max(\text{runtime}) - \text{avg}(\text{time in OpenMP}) + \text{avg}(\text{time in useful computation within OpenMP})}{\max(\text{runtime})}$$
$$= 43.1 + 44.1 - 1$$

Where **average idling time of OpenMP threads** considers that threads are idling if only master thread is working and can be computed by following formula

$$\text{average idling time of OpenMP threads} = \sum_{\text{process}=0}^{\text{num of processes}} \frac{\text{serial computation} \cdot (\text{number of threads per process} - 1)}{\text{number of all available threads}}$$

Moreover, average time in OpenMP computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP can be computed as **ordinary arithmetic mean**.

42 AdvisorPOPHybridAddTestsMissing_thread_ef

42.1 Missing Thread Efficiency?

[41.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

42.2 Missing Amdahl's Efficiency?

[43.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

43 AdvisorPOPHybridAddTestsAmdahl_efficiency

43.1 Amdahl's Efficiency

Amdahl's Efficiency indicates serial computation and can be computed as follows:

$$\text{Amdahl's Efficiency} = \frac{\text{max(runtime)} - \text{avg(idling time of OpenMP threads)}}{\text{max(runtime)}}$$

where **average idling time of OpenMP threads** considers that threads are idling if only master thread is working and can be computed by following formula

$$\text{average idling time of OpenMP threads} = \sum_{\text{process}=0}^{\text{num of processes}} \frac{\text{serial computation} \cdot (\text{number of threads per process} - 1)}{\text{number of all available threads}}$$

44 AdvisorPOPHybridAddTestsOmpRegion_efficie

44.1 OpenMP Region Efficiency

OpenMP Region Efficiency indicates inefficiencies within threads, and can be computed as follows:

$$OpenMP\ Region\ Efficiency = \frac{max(runtime) - avg(time\ in\ OpenMP) + avg(time\ in\ useful\ computation\ within\ OpenMP)}{max(runtime)}$$

Where average time in OpenMP is computed as **weighted arithmetic mean**. If number of threads is equal across processes average time in OpenMP can be computed as **ordinary arithmetic mean**.

45 AdvisorPOPHybridAddTestsMissing_omp_region

45.1 Missing OpenMP Region Efficiency?

[44.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

46 AdvisorPOPHybridAddTestsIpc

46.1 IPC (only computation)

IPC indicates number of instructions executed by CPU per clock cycle. The higher the value the better the CPU performance. It is computed as the ratio of total instructions in user code to total cycles spent in user code.

- If PAPI counters are available, use **PAPI_TOT_INS / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use **instructions / cycles**.

47 AdvisorPOPHybridAddTestsMissing_ipc

47.1 Missing IPC?

46.1 metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_TOT_INS** and **PAPI_TOT_CYC**,
- or the Perf counters **instructions** and **cycles**.

How to do it see Score-P manual

48 AdvisorPOPHybridAddTestsStalled_resources

48.1 Stalled resources (only computation)

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources.

- If PAPI counters are available, use **PAPI_RES_STL / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use one of
 - **(stalled-cycles-backend+stalled-cycles-frontend) / cycles**,
 - **stalled-cycles-backend / cycles**,
 - **stalled-cycles-frontend / cycles**,

depending on what is available.

49 AdvisorPOPHybridAddTestsMissing_stalled_re

49.1 Missing Stalled resources?

48.1 metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_RES_STL** and **PAPI_TOT_CYC**,
- or the Perf counter **cycles** and at least one of **stalled-cycles-backend**, **stalled-cycles-frontend** .

How to do it see Score-P manual

50 AdvisorPOPHybridAddTestsNoWaitINS_efficien

50.1 Instructions (only computation)

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the [52.1](#).

- If PAPI counters are available, use **PAPI_TOT_INS**.
- Otherwise, if Perf counters are available, use **instructions**.

51 AdvisorPOPHybridAddTestsMissingNoWaitINS

51.1 Missing Instructions (only computation)?

50.1 metric is available only, if the measurement has collected

- either the PAPI counter **PAPI_TOT_INS**,
- or the Perf counter **instructions**.

How to do it see Score-P manual

52 AdvisorPOPHybridAddTestsComputationTime

52.1 Computation time

Computation time indicated total time spend in the computation call path **without** MPI, OpenMP, POSIX threads, std::threads, CUDA, OpenCL, OpenACC, SHMEM. With another words, it is the user code.

53 AdvisorPOPHybridAddTestsMissingComputation

53.1 Missing Computation time?

[52.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

54 AdvisorBSPOPHybridTestsParallel_efficiency

54.1 Hybrid Parallel Efficiency

Hybrid Parallel Efficiency (HPE) reveals the inefficiency in processes and threads utilization. These are measured with **Hybrid Load Balance Efficiency** and **Hybrid Communication Efficiency**, and HPE can be computed directly or as a product of these two sub-metrics:

$HPE = \frac{avg(computation\ time)}{max(runtime)}$
= 56.1 · 58.1

55 AdvisorBSPOPHybridTestsMissing_parallel_eff

55.1 Missing Hybrid Parallel Efficiency?

[54.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

56 AdvisorBSPOPHybridTestsLoadBalance_efficie

56.1 Hybrid Load Balance Efficiency

Hybrid Load Balance Efficiency can be computed as follows:

$$\text{Hybrid Load Balance Efficiency} = \frac{\text{avg}(\text{computation time})}{\text{max}(\text{computation time})}$$

57 AdvisorBSPOPHybridTestsMissing_loadbalanc

57.1 Missing Hybrid Load Balance Efficiency?

[56.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

58 AdvisorBSPOPHybridTestsCommunication_eff

58.1 Hybrid Communication Efficiency

Hybrid Communication Efficiency can be evaluated directly by following formula:

$$\text{Hybrid Communication Efficiency} = \frac{\max(\text{computation time})}{\max(\text{runtime})}$$

This metric identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations.

59 AdvisorBSPOPHybridTestsMissing_communica

59.1 Missing Hybrid Communication Efficiency?

[58.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

60 AdvisorBSPOPHybridTestsMPIParallel_efficien

60.1 MPI Parallel Efficiency

MPI Parallel Efficiency (MPE) reveals the inefficiency in MPI processes. MPE can be computed directly or as a product of [62.1](#) and [64.1](#) :

$MPE = \frac{avg(time\ outside\ of\ MPI)}{max(runtime)}$
$= \text{62.1} \cdot \text{64.1}$

61 AdvisorBSPOPHybridTestsMissing_MPIparalle

61.1 Missing MPI Parallel Efficiency?

[60.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

62 AdvisorBSPOPHybridTestsMPILoad_balance_e

62.1 MPI Load Balance Efficiency

MPI Load Balance Efficiency can be computed as follows:

$$MPI\ Load\ Balance\ Efficiency = \frac{avg(time\ outside\ of\ MPI)}{max(time\ outside\ of\ MPI)}$$

63 AdvisorBSPOPHybridTestsMissing_MPIload_ba

63.1 Missing MPI Load Balance Efficiency?

[62.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

64 AdvisorBSPOPHybridTestsMPICommunication

64.1 MPI Communication Efficiency

MPI Communication Efficiency can be evaluated directly by following formula:

$$MPI\ Communication\ Efficiency = \frac{max(time\ outside\ of\ MPI)}{max(runtime)}$$

This metric identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations.

65 AdvisorBSPOPHybridTestsMissing_MPIcommu

65.1 Missing MPI Communication Efficiency?

[64.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

66 AdvisorBSPOPHybridTestsOMPParallel_efficie

66.1 OpenMP Parallel Efficiency

OpenMP Parallel Efficiency (OMPE) reveals the inefficiency in OpenMP threads. OMPE can be computed directly or as a division [54.1](#) by [60.1](#) :

$$OMPE = \frac{avg(computation\ time)}{avg(time\ outside\ of\ MPI)}$$

$$= 54.1 / 60.1$$

67 AdvisorBSPOPHybridTestsMissing_OMPparallel

67.1 Missing OpenMP Parallel Efficiency?

[66.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

68 AdvisorBSPOPHybridTestsOMPLoadBalance_e

68.1 OpenMP Load Balance Efficiency

OpenMP Load Balance Efficiency can be computed as follows:

$$\text{OpenMP Load Balance Efficiency} = 56.1 / 62.1$$

69 AdvisorBSPOPHybridTestsMissing_OMPlloadba

69.1 Missing OpenMP Load Balance Efficiency?

[68.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

70 AdvisorBSPOPHybridTestsOMPCommunication

70.1 OpenMP Communication Efficiency

OpenMP Communication Efficiency can be evaluated directly by following formula:

$\text{OpenMP Communication Efficiency} = \frac{\max(\text{computation time})}{\max(\text{time outside of MPI})}$
= 58.1 / 64.1

71 AdvisorBSPOPHybridTestsMissing_OMPcomm

71.1 Missing OpenMP Communication Efficiency?

[70.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

72 AdvisorBSPOPHybridTestsIpc

72.1 IPC (only computation)

IPC indicates number of instructions executed by CPU per clock cycle. The higher the value the better the CPU performance. It is computed as the ratio of total instructions in user code to total cycles spent in user code.

- If PAPI counters are available, use **PAPI_TOT_INS / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use **instructions / cycles**.

73 AdvisorBSPOPHybridTestsMissing_ipc

73.1 Missing IPC?

[72.1](#) metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_TOT_INS** and **PAPI_TOT_CYC**,
- or the Perf counters **instructions** and **cycles**.

How to do it see Score-P manual

74 AdvisorBSPOPHybridTestsStalled_resources

74.1 Stalled resources (only computation)

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources.

- If PAPI counters are available, use **PAPI_RES_STL / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use one of
 - **(stalled-cycles-backend+stalled-cycles-frontend) / cycles**,
 - **stalled-cycles-backend / cycles**,
 - **stalled-cycles-frontend / cycles**,depending on what is available.

75 AdvisorBSPOPHybridTestsMissing_stalled_res

75.1 Missing Stalled resources?

[74.1](#) metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_RES_STL** and **PAPI_TOT_CYC**,
- or the Perf counter **cycles** and at least one of **stalled-cycles-backend**, **stalled-cycles-frontend** .

How to do it see Score-P manual

76 AdvisorBSPOPHybridTestsNoWaitINS_efficiency

76.1 Instructions (only computation)

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the [78.1](#).

- If PAPI counters are available, use **PAPI_TOT_INS**.
- Otherwise, if Perf counters are available, use **instructions**.

77 AdvisorBSPOPHybridTestsMissingNoWaitINS_

77.1 Missing Instructions (only computation)?

76.1 metric is available only, if the measurement has collected

- either the PAPI counter **PAPI_TOT_INS**,
- or the Perf counter **instructions**.

How to do it see Score-P manual

78 AdvisorBSPOPHybridTestsComputationTime

78.1 Computation time

Computation time indicated total time spend in the computation call path **without** MPI, OpenMP, POSIX threads, std::threads, CUDA, OpenCL, OpenACC, SHMEM. With another words, it is the user code.

79 AdvisorBSPOPHybridTestsMissingComputation

79.1 Missing Computation time?

[78.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

80 AdvisorJSCTestsLoad_balance

80.1 MPI computation Load Balance

MPI computation Load Balance can be evaluated directly by following formula:

$$MPI\ Computation\ Load\ Balance = \frac{avg(computation\ time)}{max(runtime)}$$

81 AdvisorJSCTestsMissing_load_balance

81.1 Missing MPI computation Load Balance?

[80.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

82 AdvisorJSTestsCommunication_efficiency

82.1 MPI communication Efficiency

MPI communication Efficiency (MPI CommE) can be evaluated directly by following formula:

$$MPICommE = \frac{max(computation\ time)}{max(runtime)}$$

MPI CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. **MPI CommE** is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using [84.1](#) and [86.1](#). Combination of these two sub-metrics gives us **Communication Efficiency**:

$$CommE = 84.1 \cdot 86.1$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

83 AdvisorJSCTestsMissing_communication_effici

83.1 Missing MPI communication Efficiency?

[82.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

84 AdvisorJSCTestsSerialisation_efficiency

84.1 Serialisation Efficiency

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

85 AdvisorJSCTestsMissing_serialisation_efficiency

85.1 Missing Serialisation Efficiency?

[84.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

86 AdvisorJSCTestsTransfer_efficiency

86.1 Transfer Efficiency

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{\text{maximum across processes}(\text{total runtime on ideal network})}{\text{maximum across processes}(\text{total measured runtime})}$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

87 AdvisorJSCTestsMissing_transfer_efficiency

87.1 Missing Transfer Efficiency?

[86.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

88 AdvisorJSTestsAmdahl_efficiency

88.1 OpenMP Amdahl's Efficiency

OpenMP Amdahl's Efficiency indicates serial computation and can be computed as follows:

$$Amdahl's\ Efficiency = \frac{parallel\ execution\ time}{total\ runtime}$$

where **parallel execution time** is a time spent in OpenMP parallel regions. Amdahl's Efficiency computed per MPI rank, significant difference between MAX and MIN values indicate that some process have bigger serialisation part than others.

89 AdvisorJSCTestsMissingAmdahl_efficiency

89.1 Missing OpenMP Amdahl's Efficiency?

[88.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

90 AdvisorJSTestsOmpLoad_balance

90.1 OpenMP Load Balance Efficiency

OpenMP Load Balance Efficiency reveals the inefficiency in OpenMP parallel regions caused by imbalance and can be computed as follows:

$$\text{OpenMP Load Balance Efficiency} = \frac{\text{avg}(\text{computation time in OpenMP})}{\text{max}(\text{computation time in OpenMP})}$$

This metric computed per MPI rank and average across MPIs is shown. Moreover, MIN and MAX values are also available by click on metric bar. Statistics shows if there is considerable difference in workload across MPI ranks.

91 AdvisorJSCTestsMissing_omp_load_balance

91.1 Missing OpenMP Load Balance Efficiency?

[90.1](#) metric is enabled if application has OpenMP part.

92 AdvisorJSCTestsOmpSerialisation_efficiency

92.1 OpenMP Serialisation Efficiency

OpenMP Serialisation Efficiency indicates serialisation within OpenMP regions across MPI ranks (e.g. time in barriers, critical sections, atomics, etc.) and can be computed as follows:

$$\text{OpenMP Serialisation Efficiency} = \frac{\max(\text{runtime in OpenMP without serialisation})}{\max(\text{runtime in OpenMP without management overhead})}$$

If **management overhead** is not known due to missing Scalasca analysis, **total runtime without management overhead** is equal to total runtime. **OpenMP Serialisation Efficiency** computed per MPI rank and average across MPIs is shown. Moreover, MIN and MAX values are also available by click on metric bar. Statistics shows if there is considerable difference in serialisation across MPI ranks.

93 AdvisorJSCTestsMissing_omp_serialisation_eff

93.1 Missing OpenMP Serialisation Efficiency?

[92.1](#) metric is enabled if application has OpenMP part.

94 AdvisorJSCTestsIpc

94.1 IPC (only computation)

IPC indicates number of instructions executed by CPU per clock cycle. The higher the value the better the CPU performance. It is computed as the ratio of total instructions in user code to total cycles spent in user code.

- If PAPI counters are available, use **PAPI_TOT_INS / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use **instructions / cycles**.

95 AdvisorJSCTestsMissing_ipc

95.1 Missing IPC?

94.1 metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_TOT_INS** and **PAPI_TOT_CYC**,
- or the Perf counters **instructions** and **cycles**.

How to do it see Score-P manual

96 AdvisorJSCTestsStalled_resources

96.1 Stalled resources (only computation)

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources.

- If PAPI counters are available, use **PAPI_RES_STL / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use one of
 - **(stalled-cycles-backend+stalled-cycles-frontend) / cycles**,
 - **stalled-cycles-backend / cycles**,
 - **stalled-cycles-frontend / cycles**,depending on what is available.

97 AdvisorJSCTestsMissing_stalled_resources

97.1 Missing Stalled resources?

96.1 metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_RES_STL** and **PAPI_TOT_CYC**,
- or the Perf counter **cycles** and at least one of **stalled-cycles-backend**, **stalled-cycles-frontend** .

How to do it see Score-P manual

98 AdvisorJSCTestsNoWaitINS_efficiency

98.1 Instructions (only computation)

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the [100.1](#).

- If PAPI counters are available, use **PAPI_TOT_INS**.
- Otherwise, if Perf counters are available, use **instructions**.

99 AdvisorJSCTestsMissingNoWaitINS_efficiency

99.1 Missing Instructions (only computation)?

98.1 metric is available only, if the measurement has collected

- either the PAPI counter **PAPI_TOT_INS**,
- or the Perf counter **instructions**.

How to do it see Score-P manual

100 AdvisorJSTestsComputationTime

100.1 Computation time

Computation time indicated total time spend in the computation call path **without** MPI, OpenMP, POSIX threads, std::threads, CUDA, OpenCL, OpenACC, SHMEM. With another words, it is the user code.

101 AdvisorJSTestsMissingComputationTime

101.1 Missing Computation time?

[100.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

102 AdvisorPOPTestsParallel_efficiency

102.1 Parallel Efficiency

Parallel Efficiency (PE) reveals the inefficiency in splitting computation over processes and then communicating data between processes. As with GE, PE is a compound metric whose components reflects two important factors in achieving good parallel performance in code:

- Ensuring even distribution of computational work across processes
- Minimizing time communicating data between processes

These are measured with **Load Balance Efficiency** and **Communication Efficiency**, and PE is defined as the product of these two sub-metrics:

$$PE = 104.1 \cdot 106.1$$

103 AdvisorPOPTestsMissing_parallel_efficiency

103.1 Missing Parallel Efficiency?

[102.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

104 AdvisorPOPTestsLoad_balance

104.1 Load Balance

Load Balance (LB) is computed as the ratio between average useful computation time (across all processes) and maximum useful computation time (also across all processes):

$$LB = \frac{avg(computation\ time)}{max(computation\ time)}$$

Thus it shows how big is a difference between average and maximal computation.

105 AdvisorPOPTestsMissing_load_balance

105.1 Missing Load Balance?

[104.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

106 AdvisorPOPTestsCommunication_efficiency

106.1 Communication Efficiency

Communication Efficiency (CommE) is the maximum across all processes of the ratio between useful computation time and total run-time:

$$CommE = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime}} \right)$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using [108.1](#) and [110.1](#). Combination of these two sub-metrics gives us **Communication Efficiency**:

$$CommE = 108.1 \cdot 110.1$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

107 AdvisorPOPTestsMissing_communication_eff

107.1 Missing Communication Efficiency?

[106.1](#) metric is a basic POP metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another tool than Score-P/Scalasca, it might have missing metric **Time**. In this case POP analysis is not possible.

108 AdvisorPOPTestsSerialisation_efficiency

108.1 Serialisation Efficiency

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

109 AdvisorPOPTestsMissing_serialisation_efficie

109.1 Missing Serialisation Efficiency?

[108.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

110 AdvisorPOPTestsTransfer_efficiency

110.1 Transfer Efficiency

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{\text{maximum across processes}(\text{total runtime on ideal network})}{\text{maximum across processes}(\text{total measured runtime})}$$

where **total run-time on ideal network** is a runtime without detected by Scalasca waiting time and MPI I/O time.

111 AdvisorPOPTestsMissing_transfer_efficiency

111.1 Missing Transfer Efficiency?

[110.1](#) metric is available only, if MPI wait states have been detected and measured. Hence it is only available for trace analysis results of Scalasca such as **scout.cubex** or **trace.cubex**

112 AdvisorPOPTestsIpc

112.1 IPC (only computation)

IPC indicates number of instructions executed by CPU per clock cycle. The higher the value the better the CPU performance. It is computed as the ratio of total instructions in user code to total cycles spent in user code.

- If PAPI counters are available, use **PAPI_TOT_INS / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use **instructions / cycles**.

113 AdvisorPOPTestsStalled_resources

113.1 Stalled resources (only computation)

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources.

- If PAPI counters are available, use **PAPI_RES_STL / PAPI_TOT_CYC**.
- Otherwise, if Perf counters are available, use one of
 - **(stalled-cycles-backend+stalled-cycles-frontend) / cycles**,
 - **stalled-cycles-backend / cycles**,
 - **stalled-cycles-frontend / cycles**,depending on what is available.

114 AdvisorPOPTestsMissing_stalled_resources

114.1 Missing Stalled resources?

[113.1](#) metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_RES_STL** and **PAPI_TOT_CYC**,
- or the Perf counter **cycles** and at least one of **stalled-cycles-backend**, **stalled-cycles-frontend** .

How to do it see Score-P manual

115 AdvisorPOPTestsNoWaitINS_efficiency

115.1 Instructions (only computation)

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the [117.1](#).

- If PAPI counters are available, use **PAPI_TOT_INS**.
- Otherwise, if Perf counters are available, use **instructions**.

116 AdvisorPOPTestsMissingNoWaitINS_efficiency

116.1 Missing Instructions (only computation)?

115.1 metric is available only, if the measurement has collected

- either the PAPI counter **PAPI_TOT_INS**,
- or the Perf counter **instructions**.

How to do it see Score-P manual

117 AdvisorPOPTestsComputationTime

117.1 Computation time

Computation time indicated total time spend in the computation call path **without** MPI, OpenMP, POSIX threads, std::threads, CUDA, OpenCL, OpenACC, SHMEM. With another words, it is the user code.

118 AdvisorPOPTestsMissingComputationTime

118.1 Missing Computation time?

[117.1](#) metric is a basic Cube metric and is available for every Score-P/Scalasca measurement. If Cube Report was produced by another than Score-P/Scalasca, it might have missing metric **Time**.

119 AdvisorPOPComputation_efficiency

119.1 Computation Efficiency

Computation Efficiency is a ratio of total time in useful computation summed over all processes. For strong scaling (i.e. problem size is constant) it is the ratio of total time in useful computation for a reference case (e.g. on 1 process or 1 compute node) to the total time as the number of processes (or nodes) is increased. For **Computation Efficiency** to have a value of 1 this time must remain constant regardless of the number of processes.

Insight into possible causes of poor computation scaling can be investigated using metrics devised from processor hardware counter data. Two causes of poor computational scaling are:

- Dividing work over additional processes increases the total computation required
- Using additional processes leads to contention for shared resources

and we investigate these using [120.1](#) and [121.1](#).

120 AdvisorPOPInstruction_efficiency

120.1 Instruction Efficiency

Instruction Efficiency is the ratio of total number of useful instructions for a reference case (e.g. 1 processor) compared to values when increasing the numbers of processes. A decrease in Instruction Efficiency corresponds to an increase in the total number of instructions required to solve a computational problem.

121 AdvisorPOPTestsIpc_efficiency

121.1 IPC Efficiency

IPC Efficiency compares IPC to the reference, where lower values indicate that rate of computation has slowed. Typical causes for this include decreasing cache hit rate and exhaustion of memory bandwidth, these can leave processes stalled and waiting for data.

122 AdvisorPOPTestsMissing_ipc

122.1 Missing IPC?

[121.1](#) metric is available only, if the measurement has collected

- either the PAPI counters **PAPI_TOT_INS** and **PAPI_TOT_CYC**,
- or the Perf counters **instructions** and **cycles**.

How to do it see Score-P manual

123 Customization with Qt Stylesheets

Style Sheet Editor

Qt Style Sheets allow the user to customize the appearance of widgets. Qt Style Sheets are similar to HTML Cascading Style Sheets (CSS) but adapted to widgets. To define style sheets, open the Editor with *Display* ⇒ *Customize style sheet*.

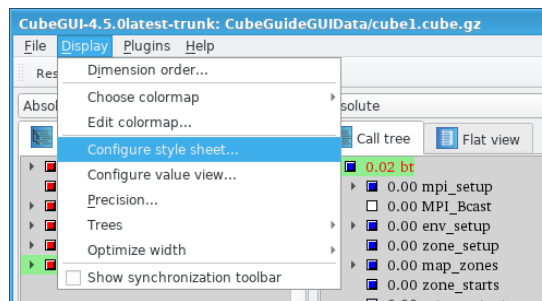


Figure 123.1: start style sheet editor

The following example customizes the appearance of the three tree views. The tree items are drawn in black, selected tree items in red. The background color of the tree items is set to lightgray, the background color of selected items to green. To draw the tree items, the font family "Bitstream Charter" with 10 point size is used.

```
QTreeView {
    color: black;
    background-color: lightgray;
    selection-color:red;
    selection-background-color:lightgreen;
    font-family: Bitstream Charter;
    font-size: 10pt
}
```

For further information, refer to the Qt style sheet reference:

- List of widgets which can be customized
- List of properties
- Style sheet syntax

124 Appendix

124.1 File format of statistics files

Statistic files (for an example see [124.1](#)) are simply text files which contain the necessary data. The first line is always ignored but should look similar to that in the example as it simplifies the understanding for the human reader. *All values in a statistic file are simply separated by an arbitrary number of spaces.* For each pattern there is a line which contains

PatternName	MetricID	Count	Mean	Median	Minimum	Maximum	Sum	Variance	Quartil25	Quartil75
LateBroadcast	6	4	0.010	0.000031	0.000004	0.042856	0.042	0.000459		
- cnode: 5 enter: 0.245877 exit: 0.256608 duration: 0.042856										
WaitAtBarrier	18	20	0.018	0.006477	0.000002	0.065293	0.369	0.000698	0.000040	0.047409
- cnode: 14 enter: 0.192332 exit: 0.192378 duration: 0.000100										
- cnode: 12 enter: 0.326120 exit: 0.335651 duration: 0.065293										
BarrierCompletion	17	20	0.000	0.000005	0.000002	0.000018	0.000	0.000000	0.000003	0.000009
- cnode: 14 enter: 0.192332 exit: 0.192378 duration: 0.000009										
- cnode: 12 enter: 0.159321 exit: 0.165005 duration: 0.000018										
WaitAtBarrier	27	144	0.001	0.000027	0.000001	0.028451	0.212	0.000028	0.000002	0.000437
- cnode: 11 enter: 0.297292 exit: 0.297316 duration: 0.000057										
- cnode: 10 enter: 0.322577 exit: 0.332093 duration: 0.028451										

Figure 124.1: An example of a statistic file

at least the pattern name (as plain text *without spaces*), its corresponding metric id in the CUBE file (integer as text) and the count – i.e., how many instances of the pattern exist (also as integer). If more values are provided, there have to be the mean value, median, minimum and maximum as well as the sum (all as floating point numbers in arbitrary format). If one of these values is provided, all have to. The next optional value is the variance (also as a floating point number). The last two optional values of which both or none have to be provided are the 25% and the 75% quantile, also as floating point numbers.

If any of these values is omitted, all following values have to be omitted, too. If for example the variance is not provided, the lower and the upper quartile must not be provided either.

In the subsequent lines (there can be an arbitrary number), the information of the most severe instances is provided. Each of these lines has to begin with a minus sign (-). Then the text *cnode:*, followed by the cnode id of this instance in the CUBE file (integer as text) is provided. The same holds for enter, exit and duration (floats as text).

The begin of the next pattern is indicated by a blank line.

125 Getting Started with Plugin "Measurement"

125.1 Start the Plugin

To start using the measurement plugin, first open CubeGUI and navigate to the "Measurement Plugin" option under the "Open context-free plugin" menu. If you are unable to locate the plugin in the menu, please ensure that you have followed all the steps listed in the section [2.7](#).

If you're still having difficulty locating the plugin, one solution is to configure the plugin search path. You can do this by accessing the "Plugins" menu, selecting "Configure Plugin Search Paths," and adding the path to the project's .so file. Once you have done this, restart CubeGUI to activate the plugin.

Alternatively, you can start CubeGUI in verbose mode by using the command line option `-verbose`, which will list all the plugins that CubeGUI is able to locate. For additional information, please refer to the CubeGUI User Guide.

125.2 Step-by-Step example

In this section, we will guide you through an example of how to use the Measurement plugin. Once the Measurement plugin is launched, the user is prompted to either start a new measurement or load a recent one. For this example, we choose to start a new measurement. After selecting "Start New Measurement", the user will be taken to the [127](#) tab. Here, the user can specify the MPI and Compiler used to build the program they want to measure. This information is required to select an appropriate Score-P version, since the Score-P version must be installed with the same MPI and compiler. In our example, the plugin has identified a matching Score-P version in our path. We select this version and proceed to the [128](#) step. In the [128](#) tab, we select the "jacobi" executable using the [128.1](#). Since the executable has not been instrumented yet, we choose to prepare a new instrumentation and specify our preferred [128.3.1](#), which in this case is the makefile.

The plugin provides us with two options: to [128.3.2.1](#) for the makefile or to open a makefile that has been [128.3.2.2](#) by the plugin. Since the plugin has detected our makefile, we choose to open it. This displays a tab where we can edit the makefile. The plugin opens a window to edit the makefile. At the top of this window, we find instructions on how to apply the correct compiler wrapper to enable instrumentation. In this example, we changed the compiler wrapper from `CC=mpicc` to `CC=scorep-mpicc`. After making the changes, we save them to return to the [128](#) tab window. In the next step, a dialog box appears for specifying the build command of the program. In our case, the plugin automatically suggests the correct command. However, it is possible to customize this command if necessary. After

entering the build command, we click the "Build your application" button to execute the command. The build output is displayed in the [126.3](#). If the build is successful, a message is displayed, and the [128.5](#) is enabled.

We click the [128.5](#) to proceed with the [129](#) tab. At this point, we select to perform an initial run, which is the first run that can be performed. We choose to run four processes and two threads. The plugin suggests a directory name for the experiment directory, where the resulting profile.cubex file will be located. Additionally, the plugin suggests the run command. The commands executed in the background and the required variable settings can be viewed in the [126.3](#).

The plugin offers two options to perform the measurement: a job can be submitted, or the measurement can be executed locally. In this example, we perform the measurement locally. The measurement was performed successfully, and a corresponding message is displayed. The user now has the option to either take another measurement or to view the results of the measurement. For more information, please refer to [129](#).

As next, see [126](#)

126 Overview

126.1 Layout

The layout of the Measurement Plugin is mainly divided into two sections. On the left side, there is the layout part the user can interact with. On the right side, a [126.3](#) is displayed. The interactive layout consists of three [126.2](#). At the top of each tab, users get an overview of the steps that need to be completed before proceeding to the next tab.

1. [126.2](#)
2. Tab-specific steps
3. [126.3](#)
4. Hide/Show virtual Console

126.2 Tabs

The Measurement plugin includes three tabs: Setup, Instrumentation, and Measurement. Each tab has a specific purpose and offers different functionalities to the user.

127: This tab allows the user to choose the Score-P installation for their measurement.

128: In this tab, the user can select their source code and specify the compilation command.

129: This tab is where the user can configure their measurement by selecting predefined settings.

126.3 Virtual Console

On the right side of the Measurement Plugin GUI, there is a virtual console that logs all the commands executed in the background when the user interacts with the GUI on the left side. This console provides users with valuable information on what is happening during their measurement and shows Score-P specific commands. It is possible to hide the console by pressing (4).

As next, see [127](#)

127 Setup

When the plugin is launched, the setup tab is displayed. There, the user can decide whether to load a former measurement or to start a new one. It is important to note that the plugin is only usable when there exists an installed Score-P version, and the plugin must be started in the build environment of the application the user wants to measure.

1. [127.1.1](#)
2. [127.2](#)
3. Job ID of [127.1.2](#)
4. [127.1.2.1](#) of [127.1.2](#)
5. Load measurement of [127.1.2](#)
6. Delete saved settings of [127.1.2](#)

127.1 Load Measurement

127.1.1 Load recent Measurement Button

When the user clicks on the "Load Recent Measurement" button, the most recently saved measurement will be loaded. This will return the user to the measurement at the same point where they exited the plugin during the previous session.

127.1.2 Submitted Jobs

The plugin lists all measurements that were submitted as a job during previous usage of the measurement. It displays the job ID, the status of the job and options to load or delete the specific measurement. If the user selects the option to load the measurement, they will be taken to the [129](#) tab, where they can access various options depending on the [127.1.2.1](#).

127.1.2.1 Job Status

The status of a job can be one of the following:

- COMPLETED: The job was successfully completed. The user can view the results of the job.
- FAILED: The job failed to complete. The user can try starting the job again, but no results are available for this job.
- RUNNING: The job is currently running.

- **STATUS NOT AVAILABLE:** The status of the job is not available. This could indicate that the job was not properly submitted or that an error occurred while trying to retrieve its status.

If the status of a job is not available, the user should try resubmitting the job or checking for any errors that may have occurred.

127.2 Start new Measurement

When the user selects the "Start New Measurement" button, the setup tab layout will change to show the steps required to select a Score-P installation.

The plugin offers multiple ways to select a Score-P version. If the user has a Score-P version available on their system's path, it will be automatically detected and listed in the [127.2.2](#) section. Alternatively, if the user wants to load a Score-P module, they can use the [127.2.3](#) to search for and select the desired version. Finally, if the Score-P version is not automatically detected or available as a module, the user can manually browse the system directory using the [127.2.4](#) to select the appropriate version.

The plugin lists all detected Score-P versions and provides additional information about each version, including whether PAPI counter and libunwind are available and whether it is a [127.2.2.1](#). A usable configuration implies that the MPI and/or the compiler used to build the Score-P version matches the MPI and compiler specified for building the user's application.

1. Select [127.2.1.1](#)
2. Select [127.2.1.2](#)
3. [127.2.2](#)
4. [127.2.3](#)
5. [127.2.5](#)
6. [127.2.6](#)

127.2.1 Select Compiler version and MPI

There are two drop down menus to select the MPI and Compiler used to build the program being measured.

127.2.1.1 Compiler

The user can choose between the following options:

- gcc
- ibm
- intel
- pgi
- studio

- clang

127.2.1.2 MPI

The user can choose between the following options:

- mpich2
- impi
- openmpi

127.2.2 Score-P version found in PATH

If a Score-P installation was found in the user's PATH variable, the plugin lists it with providing additional information. It gives information whether PAPI counter and libunwinded are available in this installation and if it is a [127.2.2.1](#).

127.2.2.1 Usable Configuration Status

The usable configuration status indicates whether the selected Score-P version is compatible with the MPI and compiler used to build the user's application. The status can have one of three values:

- "Usable configuration": Indicates that the selected Score-P version is compatible with the user's MPI and compiler, and can be used to measure their application without issues.
- "Possibly usable configuration": Indicates that the selected Score-P version may be compatible with the user's MPI and compiler, but it was not specified during installation. Therefore, the plugin cannot confirm if it is a usable configuration.
- "Not usable configuration": Indicates that the selected Score-P version is not compatible with the user's MPI and compiler, and cannot be used to measure their application.

Next to the status message, an info icon is displayed. If the user hovers over this icon with their mouse, a tooltip will appear containing information about the MPI and compiler used to build the selected Score-P installation.

127.2.3 Find Score-P versions Button

This button allows the plugin to search for Score-P modules and display all available versions. If no Score-P module is found, an error message is displayed to the user.

127.2.4 Browse Score-P Button

When the "Browse Score-P" button is pressed, a file dialog box will appear for the user to select an installed Score-P version. Only visible after [127.2.3](#) is clicked.

127.2.5 Proceed Button

When the user has selected an appropriate Score-P installation, the "Proceed" button becomes enabled. Clicking this button will move the user to the [128](#) tab and the selected Score-P version is used for the measurement.

127.2.6 Help Button

If the user needs assistance in detecting a Score-P version, they can click on the "Help" button. This opens a pop-up window with information on how to detect a Score-P version.

As next, see [128](#)

128 Instrumentation

The "Instrumentation" tab guides the user through the process of instrumenting their application. To begin instrumentation, the user must first select the executable they wish to instrument. They can then choose whether to create a new instrumentation setup or use an existing one. If the user chooses to create a new instrumentation setup, they must select how the application was built and adjust the build settings as necessary. They can then rebuild the application to instrument their application.

1. [128.1](#)
2. Select whether to use an [128.2.1](#) or to create a [128.2.2](#)
3. [128.3.1](#)
4. [128.3.2.1](#)
5. [128.3.2.2](#)
6. [128.4.1](#)
7. [128.4.2](#)
8. [128.5](#)

128.1 Browse Executable File Button

When the "Browse executable file" button is pressed, a file dialog box will appear for the user to select the executable file. Selecting an executable file is required to ensure that the user's application can be built without instrumentation. Once a file is selected, the program will check if the file is executable. If it is not executable, an error message will be displayed, and the user will not be able to proceed.

The program will also check if the selected file is already instrumented. If it is, the [128.2.1](#) option will be enabled. If the user selects a new executable file that is different from the previous selection, any further steps will be hidden, and any previously selected options will be unchecked.

128.2 Select Instrumentation Box

In the Select Instrumentation box, the user has the option to choose between using a [128.2.1](#) or preparing a [128.2.2](#). If the user's application is already instrumented, the option to use the [128.2.1](#) is enabled. This allows the user to continue with the existing instrumentation instead of preparing a new one. On the other hand, if the user wants to prepare a new instrumentation, they can select the option to do so. This will lead to further

steps where the user can select how the application was built, adjust the build settings, and rebuild the application to perform the instrumentation.

128.2.1 Use Former Instrumentation

The "Use Former Instrumentation" option is only enabled if the selected executable is already instrumented with Score-P. In this case, the [128.5](#) is enabled, and the user can skip the rest of the instrumentation step and directly continue with the [129](#) step.

128.2.2 Prepare New Instrumentation

If "Prepare New Instrumentation" is selected, the user is displayed with the further step to select their [128.3.1](#) in order to prepare a new instrumentation.

128.3 Adapt Build System

128.3.1 Select Build System Box

In the Select Build System Box, the user has the option to choose their build system for the executable they selected in [128.1](#) step. For now, only makefile based build systems are supported.

128.3.1.1 Adjust Makefile

When the user selects to adjust their makefile they are shown the next step [128.3.2](#) where they are asked to adjust their makefile.

128.3.2 Open Makefile for Editing

There are two options to open a makefile. The user can browse their directory for their makefile by clicking the [128.3.2.1](#) or the plugin is able to detect the makefile and it can be opened by clicking the [128.3.2.2](#). When a makefile is selected, a layout is displayed for editing. The plugin displays the required changes which the user has to add to their makefile and opens the makefile for editing. Once the user has applied the required changes, they can press the "Save changes" button to save the adapted makefile or the "Discard changes" button to revert to the original makefile. In both cases, the user returns to the layout shown before. But only when the "Save changes" button is pressed, the next step to [128.4](#) is shown.

128.3.2.1 Browse Makefile Button

When the user selects to browse the makefile manually, a file dialog box will be displayed for the user to select the makefile.

128.3.2.2 Open detected Makefile Button

This option is only enabled when the plugin was able to locate a makefile. The plugin searches the executable file's directory for a file that is named "Makefile", "makefile", or "MAKEFILE". If the file is named differently or located elsewhere, the user has to [128.3.2.1](#) manually.

128.4 Rebuild Application

In order to instrument the program, it is required to rebuild it with the adapted build settings.

128.4.1 Build Command Box

In the Build Command Box, the user can adjust the build command for their application. The plugin displays a recommended build command for the user's application, but they can also edit it if needed.

128.4.2 Build Application Button

When the user has made the necessary changes to their build settings and command, they can press the "Build Application" button to rebuild the application to perform the Score-P instrumentation. If the build is successful, the [128.5](#) will be enabled.

128.5 Continue with Analysis Button

When the user has successfully instrumented their application, the "Continue with Analysis" button becomes enabled. Clicking this button will move the user to the [129](#) tab, where they can configure the measurement settings for their instrumented application.

As next, see [129](#)

129 Measurement

The "Measurement" tab provides guidance to the user on measuring their application. To begin measurement, the user must first select the [129.2](#) they wish to perform. They can choose whether to do an [129.2.1](#), [129.2.2](#), custom, or detailed run. Next, the user needs to specify some [129.1](#) which are the same for all runs. This step is followed by selecting run-specific settings. Afterward, the user needs to run their application and will be offered options on how to proceed.

1. Select run
2. Specify [129.1.1](#)
3. Specify [129.1.2](#)
4. Specify [129.1.3](#)
5. Specify [129.3](#) command
6. Prepare [129.3.1](#) button
7. [129.3](#) local button

129.1 Presettings

There are a few options that are the same for each run. These settings are specified in this section.

129.1.1 Number of Processes

This combo box allows the user to specify the number of processes to be used for running the program. The value will be included as a command line argument in the mpirun or mpiexec command and will also be part of the suggested name for the [129.1.3](#).

129.1.2 Number of Threads

This combobox enables the user to specify the number of threads to be used for program execution. Upon selection, the "OMP_NUM_THREADS" variable will be exported with the corresponding value.

129.1.3 Experiment Directory Name

This line edit allows the user to specify the name of the experiment directory. The directory is where the profile.cubex file containing the measurement results will be stored after the measurement is completed.

129.2 Runs

129.2.1 Initial Run

The initial run creates the first measurement, which serves as the basis for a more accurate measurement. It is important to note that the initial run may not provide optimal results, but it is necessary to conduct it before performing a more accurate measurement.

The initial run does not require any run-specific settings.

129.2.2 Finetuned Run

The finetuned run is available once the first measurement of the application exists. This run employs a [129.2.2.1](#) to enhance the measurement. However, it does not improve the application itself.

129.2.2.1 Filter

In the finetuned run, the user is required to use a filter file. Please refer to the Score-P User Guide for more information on filter files. The plugin provides two options for using a filter file: using an existing filter file by specifying its path, or creating a new filter file within the plugin. If the user chooses to use an existing filter file, a window will appear allowing them to browse the directory and select the filter file. If the user chooses to create a filter file, the plugin provides two options: creating a filter file manually or generating an initial filter file. If the user chooses to create a filter file manually, the plugin displays a window allowing the user to create their own filter file. The plugin also provides hints on how to create the filter file.

If the user chooses to generate an initial filter file, the plugin will prompt the user to select a measurement that will serve as the basis for creating the filter. This is because the plugin utilizes Score-P's scorep-score tool. In the following step, the user can customize scorep-score's command line options. When the user has selected a filter file in any of the described ways, the step to [129.3](#) will be visible. Additionally, the SCOREP_FILTERING_FILE variable will be exported with the path to the selected filter file. The user will also have the option to inspect the filter. Clicking on this button will open a window displaying the selected filter file, allowing the user to edit it.

129.3 Run the program

To create the measurement, it is necessary to execute the instrumented program. The plugin provides two options for running the program: locally or by submitting a jobscript.

If the user chooses to run the program locally, the command specified in the run command box will be executed. It should be noted that the full path to the command used must be specified in the command line. Nevertheless, the plugin typically suggests this.

The plugin waits for the program to finish running, and during that time, the screen may appear frozen. Therefore, it is recommended to use the jobscript option for larger programs.

129.3.1 Prepare job Script

The plugin provides the option to open either a job script generated by the plugin itself, or the user's own job script.

129.3.1.1 Open generated job Script

If the user selects to open a generated job script, the plugin creates a sbatch job script containing all the necessary information specified earlier in this tab. The user can review and modify the script before submitting it to the job scheduler.

129.3.1.2 Open own job Script

If the user selects to open their own job script, the plugin opens an edit window displaying the script. The user is asked to modify it in order to apply the required changes corresponding to the options selected earlier. The plugin provides appropriate suggestions that should be added to the job script, which could look like the following: Once the user has made the required changes, they can save the modified script and submit it. To submit the job, the plugin uses the sbatch scheduler.

After the job is submitted, the user can wait for it to finish within the plugin or close the plugin and return later. The job is added to the list of [127.1.2](#) which is displayed when the plugin is started. From there, the user can easily check the status of the job and return when it is finished.

129.4 Options after running the Program

After the program has been successfully run, the user has the option to save a shell script which can be used to reconstruct the measurement they have just performed. The user is also offered the option to view the measurement results or to perform another measurement with the same executable. If the user selects to view the measurement results, the plugin is closed and CubeGUI opens the created profile.cubex file containing the measurement profile. If the user selects to perform another measurement, all the options selected in the [129](#) tab are reset, but there is no need to redo the [127](#) and [128](#) steps.

To return, see [2.7.2.6](#)

