cube scalasca

CubeGUI 4.5 | User Guide

Introduction in Cube GUI and its usage

May 2020
The Scalasca Development Team
scalasca@fz-juelich.de

JÜLICH FORSCHUNGSZENTRUM

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

2 Cube User Guide

2.1 Abstract

CUBE is a presentation component suitable for displaying performance data for parallel programs including MPI and OpenOpenMP 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*. Furthermore, the display is augmented with a source-code display that shows the position of a call site in the source code.

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

-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

-lastN, preload use given memory strategy. With the default value (skipped option) CubeGUI reads the data from the .cubex file by the first access to it. With the "preload" it will read all the data into the memory during the initialization. "lastN" strategy forces to keep in memory last N used data rows. N=1 will make CubeGUI load and drop the data after every calculation, which will lead to the minimal memory footprint, but to the I/O overhead. N is specified via environment variable 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 were 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.

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 lyfecycle of the Cube object.
2. **preload** - all data is loaded during the metric initialization and kept in memory to the end of lyfecycle 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.

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 Figure2.1, they represent call paths forming a call-tree. The nodes in the system dimension represent machines, nodes, processes, or threads from top to bottom.

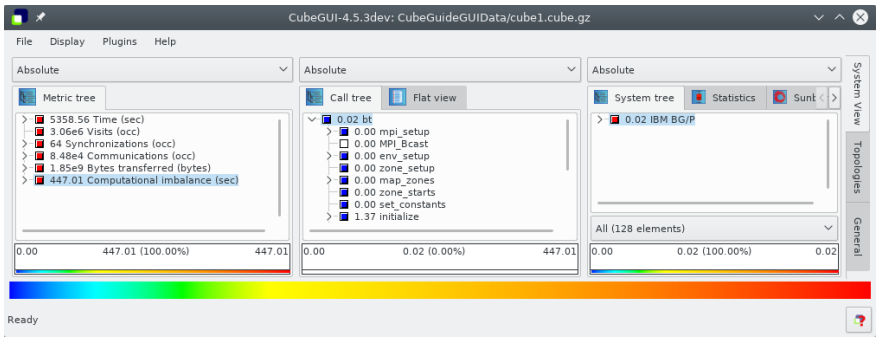

Figure 2.1: CUBE display window

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 identification of nodes of interest even in a large tree, whereas the numerical values enable

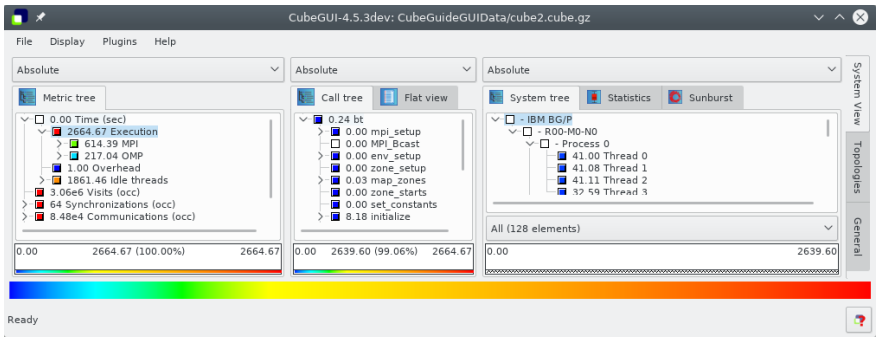

Figure 2.2: CUBE display window with expanded metric node ''Execution''

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 Figure2.1, 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.

2.5.1 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* ⇒ *Dimension order* in Section 2.5.1.1).

The metric pane provides only the metric-tree browser. The call pane offers a call-tree browser and a flat call profile. The system pane has a system-tree browser. Tree browsers also provide a context menu.

2.5.1.1 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) **Save as (Ctrl+S):** Offers a selection dialog to save a copy of a CUBE file. Opened CUBE file stays loaded in cube.
   c) **Close (Ctrl+W):** Closes the currently opened CUBE file. Disabled if no file is opened.
   d) **Open external:** Opens a file for the external percentage value mode (see Section 2.5.1.2).
   e) **Close external:** Closes the current external file and removes all corresponding data. Disabled if no external file is opened.
   f) **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.

g) **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.

h) **Quit (Ctrl+Q):** Closes the application.

i) **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 valuescan be presented, since they miss a reference metric; in this case values are specified to be undefined, denoted by a “-'' (minus) sign.

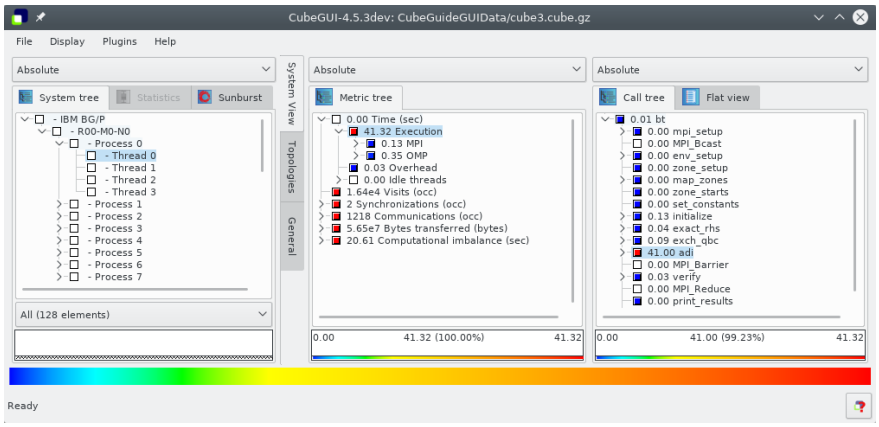

Figure 2.3: Modified pane order via the menu ''Display => Dimension order''

b) **General coloring:** 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 (2.6.3). At the top of the dialog you see a color legend with 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

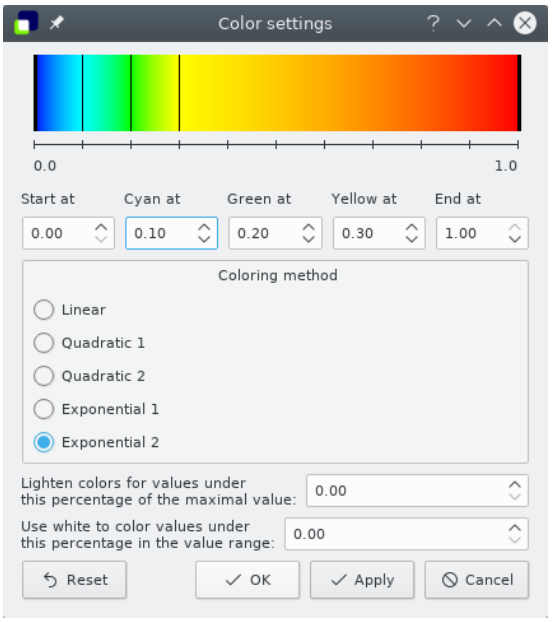

Figure 2.4: Configuration dialog of the default colormap which opened via the menu ''Display => Edit colormap''

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.

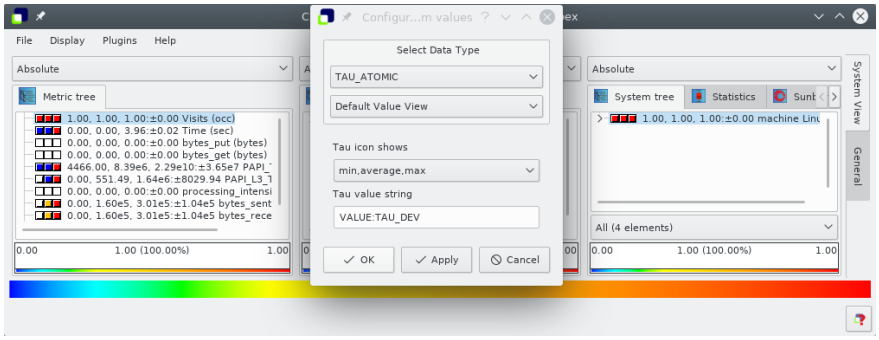

Figure 2.5: Value view config dialog for tau metrics

c) **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.

d) **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.

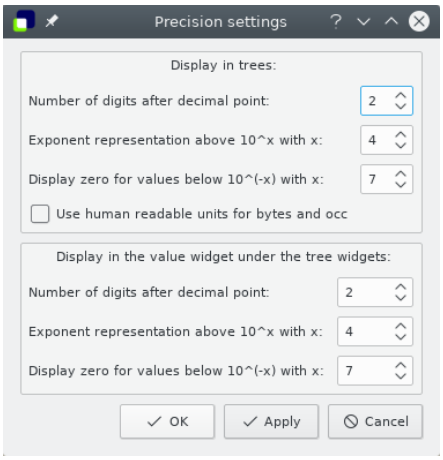

Figure 2.6: Display => Precision

e) **Trees:** This menu offers options to change the contents and the appearance of

the items of all trees.

i. **Font and colors:** Here you can specify the font and its size for the tree displays (see Figure 2.7). You may also change the background color of selected tree items. 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.

ii. **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 2.6.13).

iii. **Demangle Function Names** (only call tree and flat profile) If this option is enabled (default), cube tries to demangle function names.

iv. **Shorten Function Names** (only call tree and flat profile) This menu item opens a dialog in which you can hide parts of long function names. You may hide argument 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.

v. **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.

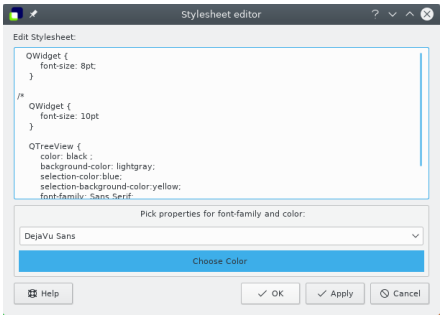

Figure 2.7: The dialog opened via the menu ''Display => Trees => Font and colors''

f) **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.

g) **Show synchronization toolbar** The synchronization of several cube instances is described in 2.7.4.

h) **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 currenly 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 *cubebasename*.ini.

3. **Plugins:** The plugin menu allows the user to define which plugins are laoded. 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.

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 2.6.7.1.

   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) **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.

   f) **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.

2.5.1.2 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'' in 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.

2.5.1.3 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.

2.5.1.4 Tree browsers

A tree browser displays different hierarchical data structures in form of trees. Currently supported tree types are metric-trees, call-trees, 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 2.5.1.1. The color icon reflects the position of the node's value between 0.0 and a maximal value. These maximal value is the maximal value in the tree for the absolute value mode, or 100.0 otherwise. See the menu item *Display* ⇒ *Choose colormap* in Section 2.5.1.1 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 2.8). 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.

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).

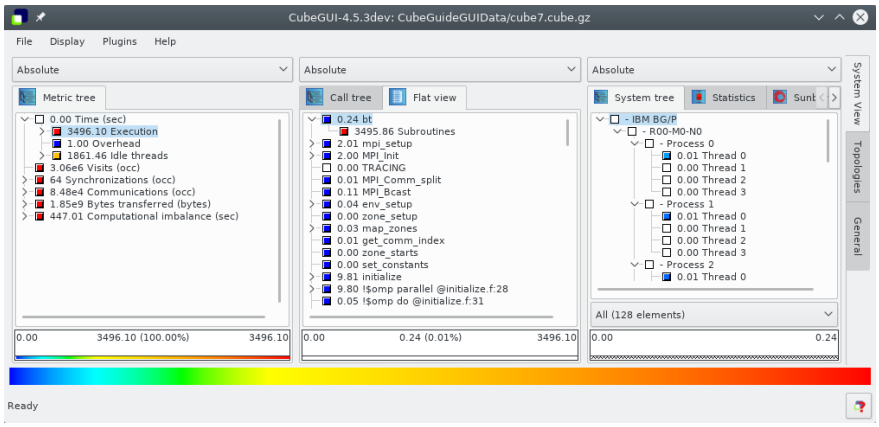

Figure 2.8: CUBE flat profile

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 refernce 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:** For all trees. Collapses all nodes in the tree.
2. **Collapse subtree:** For all trees. Enabled only if there is a reference node. It collapses all nodes in the subtree of the reference node (including the reference node).
3. **Collapse peers:** For system-trees only. Enabled only if there is a reference node. Collapses all peer nodes of the reference node, i.e., all nodes at the same hierarchy level.

4. **Expand all:** For all trees. Expands all nodes in the tree.
5. **Expand subtree:** For all trees. Enabled only if there is a reference node. Expands all nodes in the subtree of the reference node (including the reference node).
6. **Expand peers:** For system-trees only. Enabled only if there is a reference node. Expands all peer nodes of the reference node, i.e., all nodes at the same hierarchy level.
7. **Expand largest:** For all trees. Enabled only if there is a reference node. Starting at the reference node, expands its child with the largest inclusive value (if any), and continues recursively with that child until it finds a leaf. It is recommended to collapse all nodes before using this function in order to be able to see the path along the largest values.
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 an 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 begining 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 informataion 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.

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

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

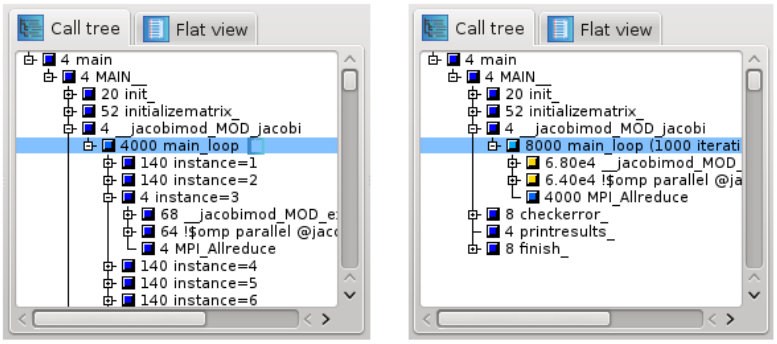

Figure 2.9: The item main_loop with 1000 iteration is marked as a loop. The aggregated view on the right is the result of selecting ''Hide iterations''.

aggregated (see Figure2.9).

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 2.5.1.5) displays a `(u)'' foruser-`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 2.7.2.
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 2.7.3.

21. **Cut all tree:** For call-trees only. Enabled only if clicked over item in call-tree. Offers different modification possibilities:
    a) **Set as root:** Removes all call path above the selected item and sets selected call path as a root node.
    b) **Prune element:** Removes the selected item and all its children. Its inclusive value will be added then to the exclusive value of its parent.
    c) **Set as leaf:** Removes all children of its element and add their inclusive values to its exclusive value.
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.

2.5.1.5 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 2.5.1.1, menu *Display* ⇒ *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.

2.5.1.6 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` ⇒ `Choose colormap`, see Section 2.5.1.1. 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 2.6.12, menu `Topology`).

2.5.1.7 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 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".

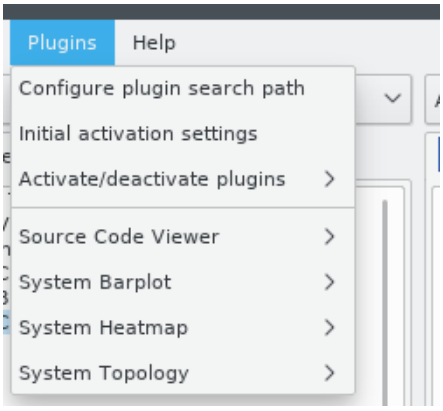

Figure 2.10: plugin menu

By Selecting the second item, a dialog is created which lists all available plugins (see Figure 2.11).

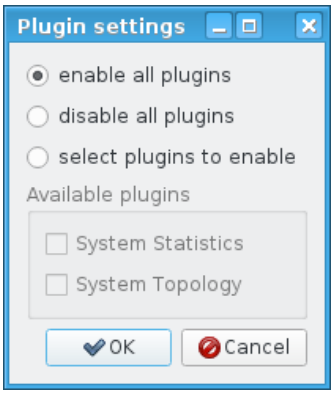

Figure 2.11: plugin settings dialog

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 2.10) to the Plugins 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 pathes, these pathes are prepended to the default search path.

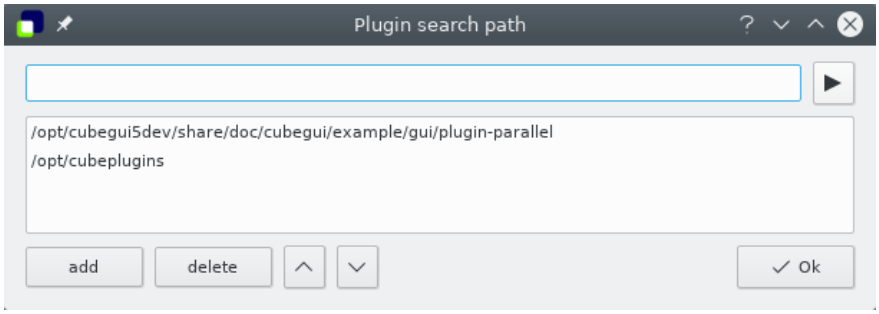

Figure 2.12: plugin search path dialog

Selecting "Configure plugin search path" of the plugin menu shows a dialog (see Figure 2.12), which allows to prepend additional search pathes. 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.

2.6.1 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 2.13). If the window is closed, the contents are moved to the tab widget again.

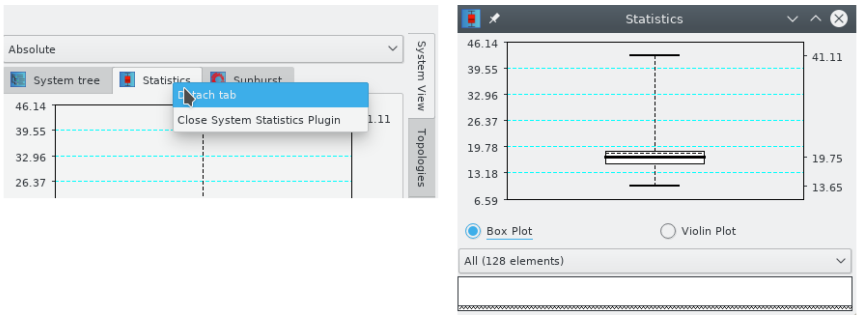

Figure 2.13: Boxplot plugin tab is detached

2.6.2 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".

2.6.2.1 Plugin "Diff"

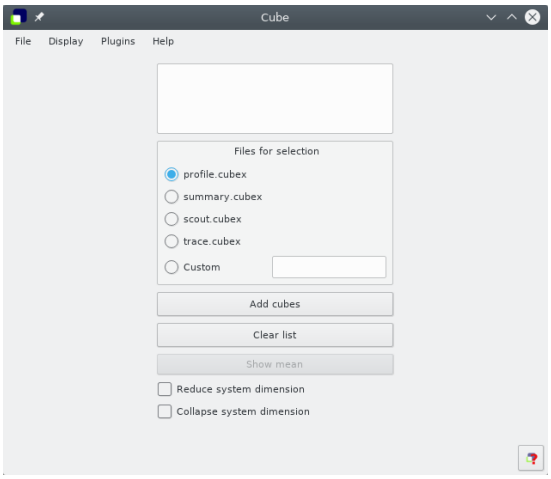

Figure 2.14: Plugin Diff

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

2.6.2.2 Plugin "Mean"

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

2.6.2.3 Plugin "Merge"

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

2.6.2.4 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 singe 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.

2.6.2.5 Plugin "Tau2Cube"

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

2.6.3 Advanced Color Map Plugin

Advanced Color Map Plugin provides additional color maps. The configuration dialogs are presented in Figure 2.15. 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.
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.
   Hue - non-negative value which controls saturation of the scheme, with pure greyscale

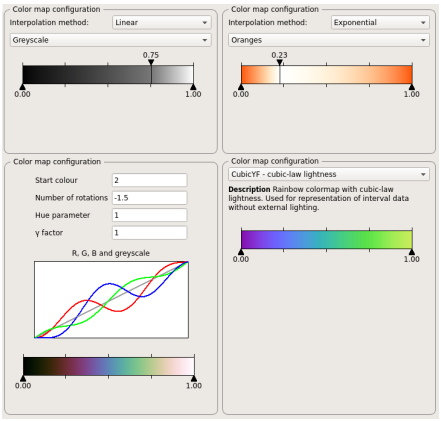

Figure 2.15: The examples of configuration for Advanced Color Maps. Upper row, starting from left: sequential, divergent; lower row, starting from left: cubehelix, improved rainbow.

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

2.6.4 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 : `Postderived metric`, `Prederived exclusive metric` and `Prederived inclusive`

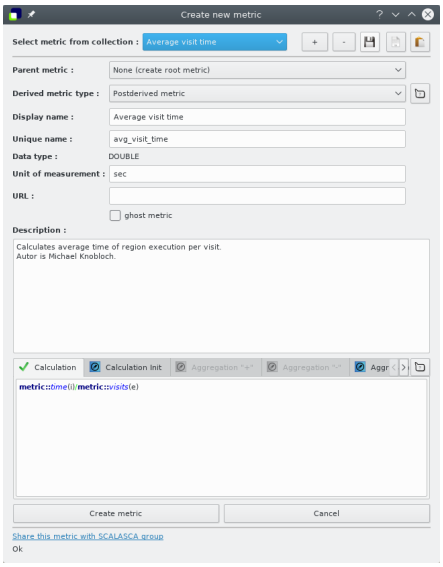

Figure 2.16: Create derived metric

`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.

2.6.5 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 2.17)

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

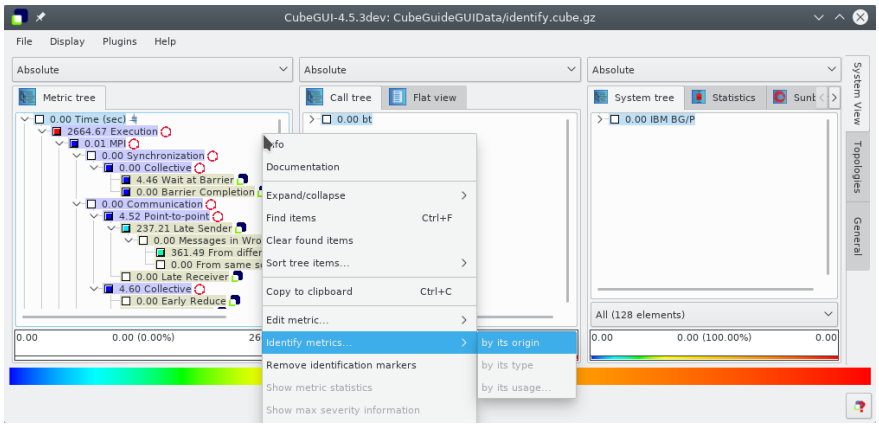

Figure 2.17: Metric identification

2.6.6 Score-P Configuration Plugin

This plugin (see Figure 2.18) presents the file "scorep.cfg", if found, in tabullar way. Tooltip displays help to every used environment variable with its possible values.

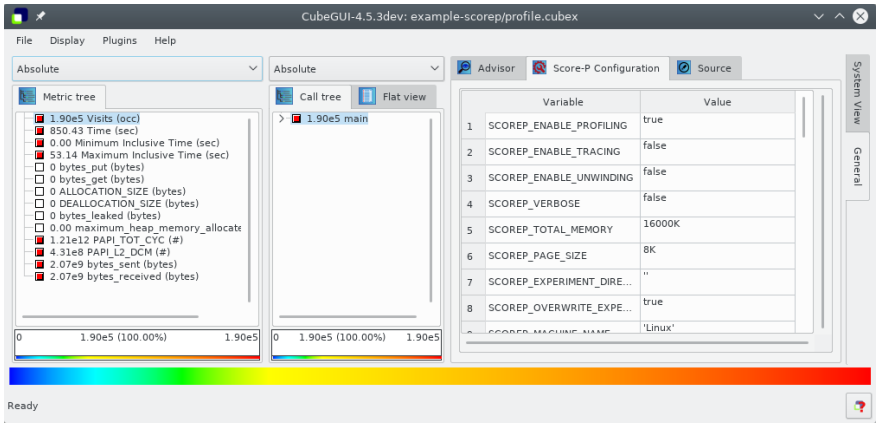

Figure 2.18: Score-P Configuration

2.6.7 Source Code Viewer

The Source code viewer plugin (see figure 2.19) 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.

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:

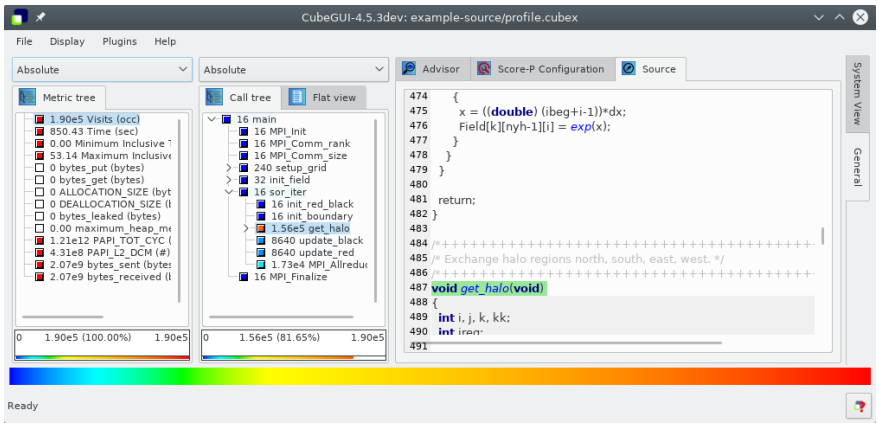

Figure 2.19: Source code viewer plugin

1. **Copy** copies the selection to the clipboard
2. **Select All** selects the whole source file
3. **Find** adds an additional widget at the bottom ot the viewer to search inside the source code
4. **Open in external editor** opens an external editor, after it is configured

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.

2.6.7.1 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

2.6.8 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.

2.6.8.1 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 2.20 displays a view of it.

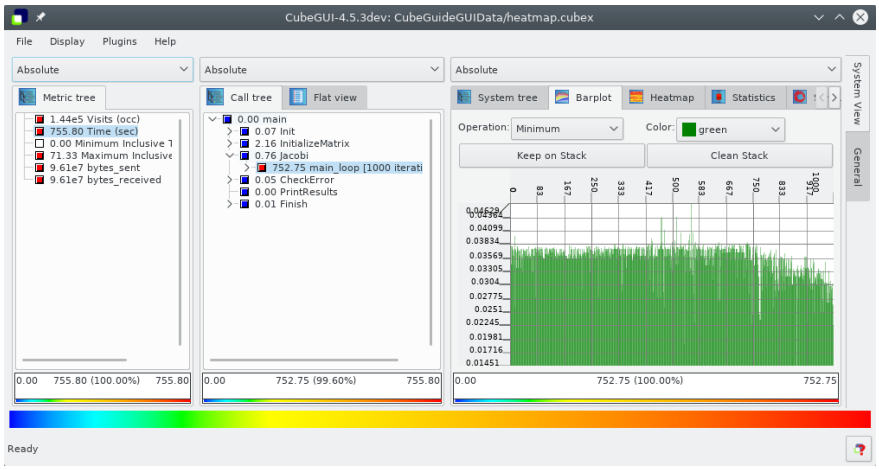

Figure 2.20: BARPLOT display window

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 2.21, no bar graph is displayed and user face the message `"No data to display"` on the window.

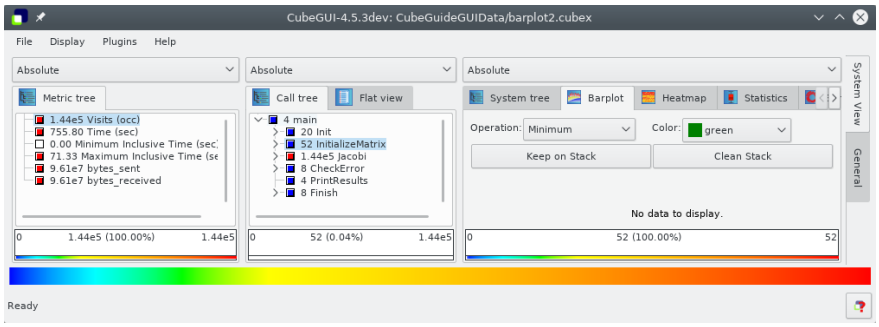

Figure 2.21: No data to display

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.

2.6.8.2 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 2.22).

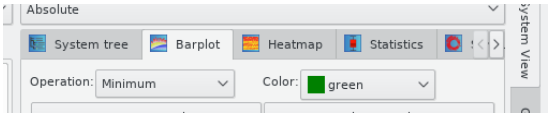

Figure 2.22: BARPLOT toolbar

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 intents 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.

2.6.8.3 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 2.23).

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

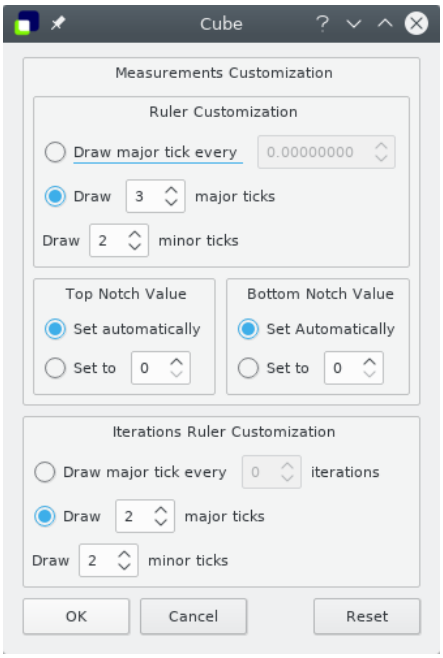

Figure 2.23: BARPLOT menu

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.

2.6.9 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.

2.6.9.1 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 2.24 displays a view of it.

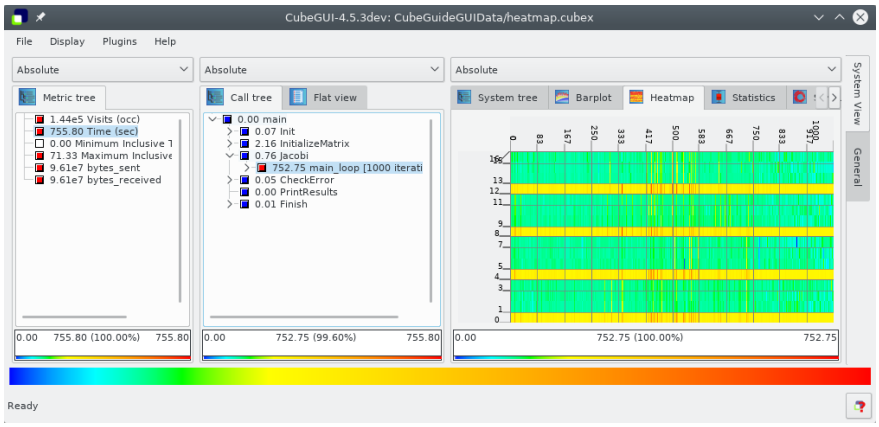

Figure 2.24: HEATMAP display window

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* figure 2.25, no heatmap graph is displayed and user face the message `"No data to display"` on a window.

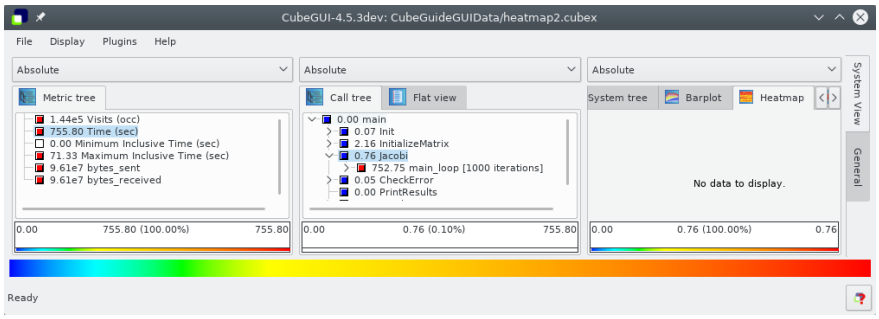

Figure 2.25: No data to display

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 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.

2.6.9.2 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 2.26).

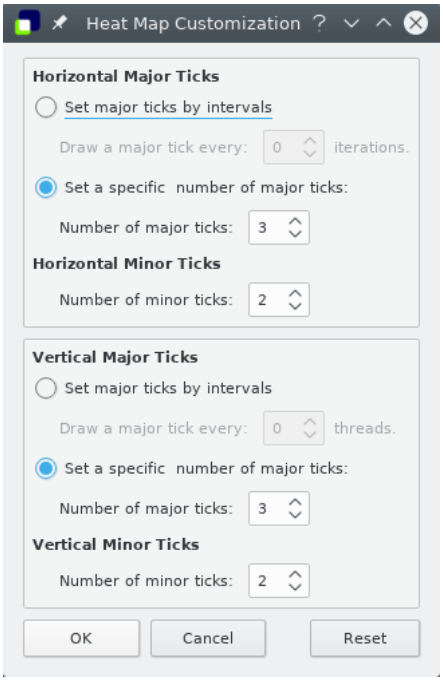

Figure 2.26: 'HEATMAP menu'

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.

2.6.10 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.

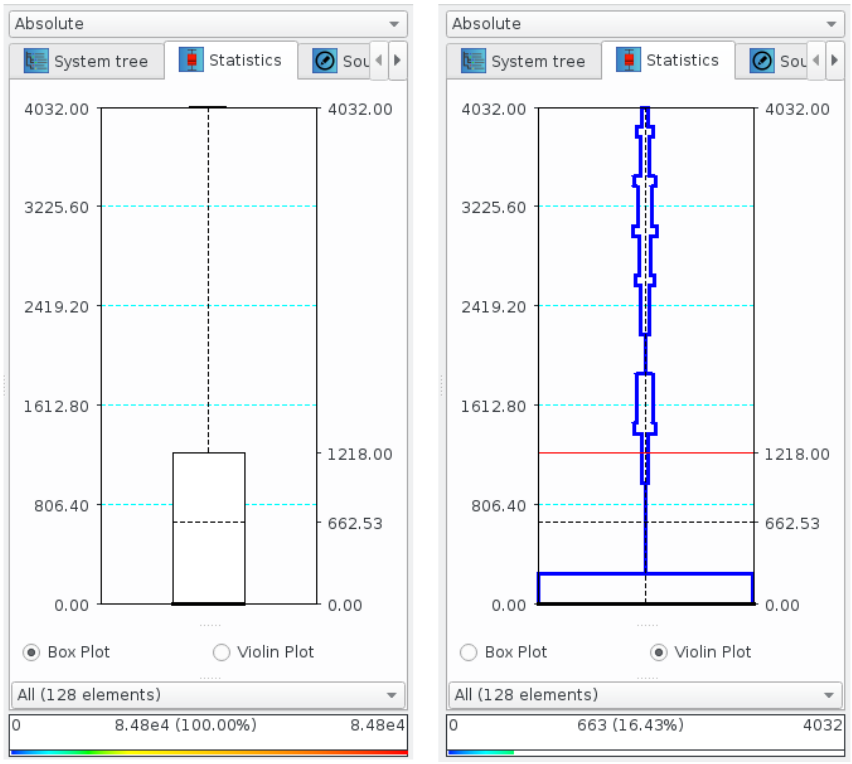

Figure 2.27: Statistical data shown in a box plot on the left side and violin plot on the right side

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. Zooming into the boxplot is done with <left-mouse drag> from top to bottom, and reset with a <middle-mouse click> inside the chart.

2.6.11 System Sunburst Plugin

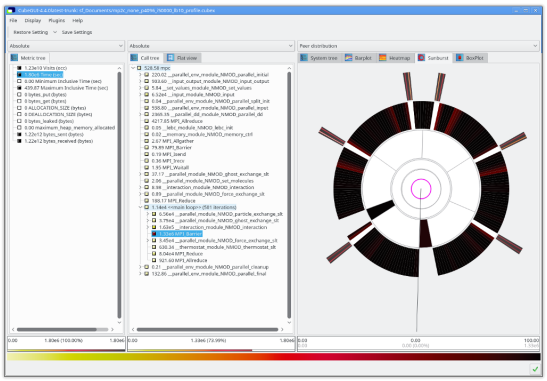

Figure 2.28: Expanded sunburst chart

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 2.29 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 2.30. 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-mouse click>. Furthermore, the context menu allows to reset specific or all interactions

Figure 2.29: Arc edge highlighted when hovering over it

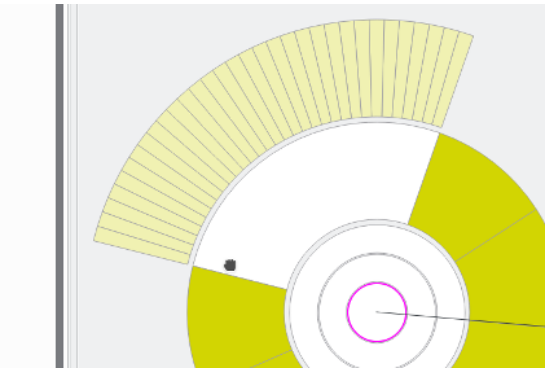

Figure 2.30: Adjusting the arc width using Ctrl+<left-mouse drag>

(e.g., rotation, arc width) with the chart to their default value.

The following table lists all available mouse interactions:

<left-mouse click>	*On arc:*Select arc *On arc edge:*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

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

2.6.12 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 three-dimensional topology is shown on Figure 2.31. Note that the topology toolbar is enabled when a topology is available to be displayed.

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

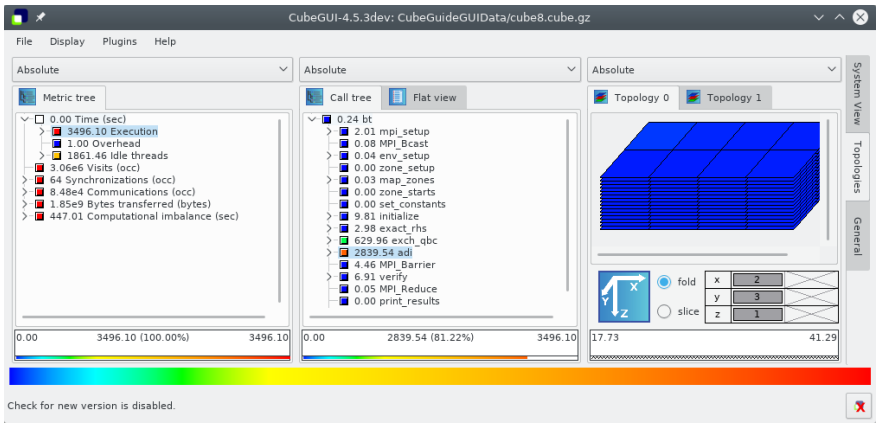

Figure 2.31: Topology Displays

attached to it are colored gray. See Section 2.5.1.1 (menu *Topology*) for further topology-specific coloring settings. For example, the upper topology in Figure 2.31 is drawn wit black lines, the 2D topology in Figure 2.32 is drawn without lines.

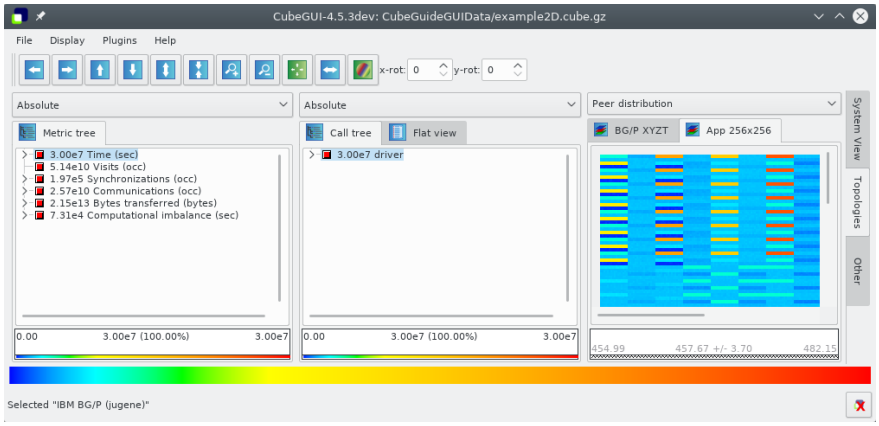

Figure 2.32: Topology Displays

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 complely.

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 2.22), 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 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>

2.6.12.1 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 Figure2.33 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.

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 Figure2.33 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 rectancle around the selection belong to Z index one. The gray lines devide the rectangle into four elements which correspond to the elements of

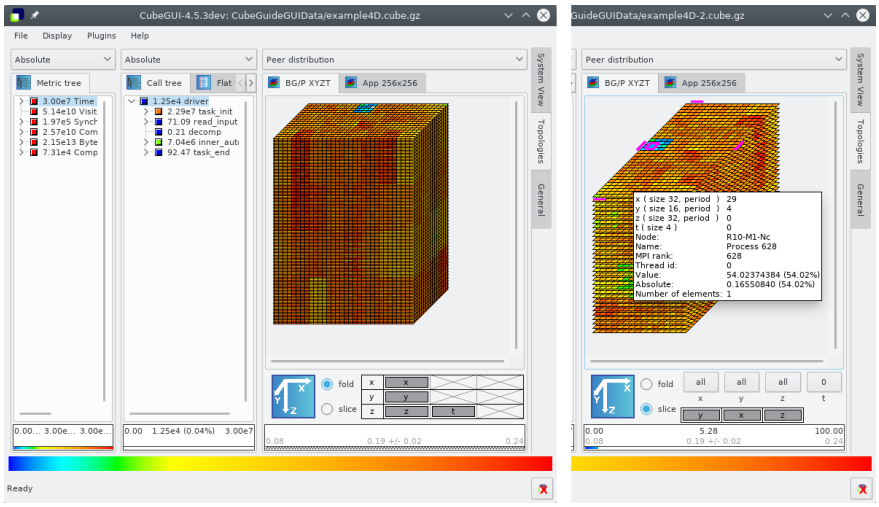

Figure 2.33: 4-dimensional example

dimension T with index 0 to 3.

2.6.12.2 Topology plugin menu

- **Topology:** The topology menu offers the following functions related to the topology display described in Section 2.31 :
  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 2.22 .
  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.

2.6.12.3 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* ⇒ *Toolbar* in Section 2.5.1.1. 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 2.5.1.4), 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)'' foruser-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.

Dimension order for the topology displays This button no longer exists, but formerly allowed the order of topology dimensions to be adjusted: this is now done with the control panel at the bottom of the topology pane.

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.

2.6.12.4 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

2.6.13 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
- Items can be grayed out. These items are either marked as unimportant by a plugin, or the user has choosen 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.

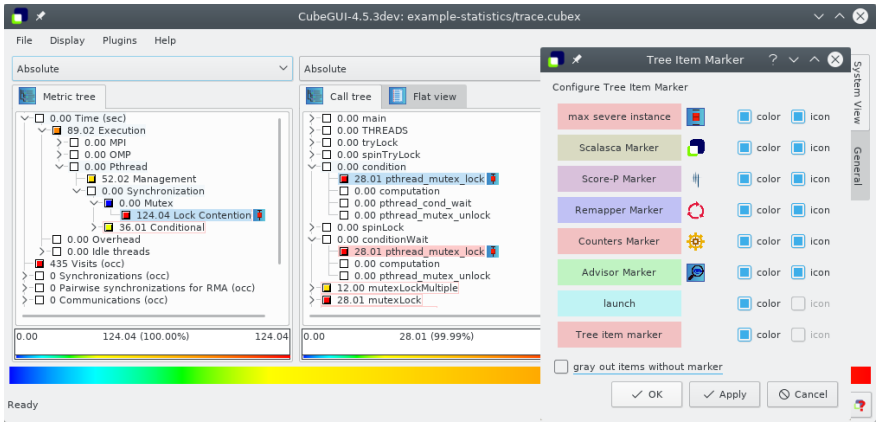

Figure 2.34: tree item marker

The figure 2.34 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 2.10) 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.

This plugins 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.

2.7 Other Features

2.7.1 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 42.1.

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`.

2.7.2 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 2.35.

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 2.35) displaying the exact values of the statistics. Clicking with the right mouse button shows the information in a tooltip.

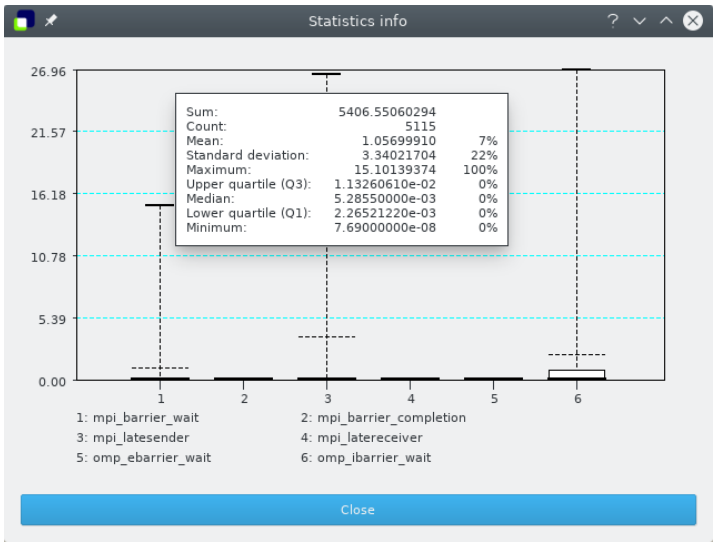

Figure 2.35: Screenshot of a box plot as shown by CUBE displaying statistical information about the selected patterns. The tooltip shows the exact values of the statistics.

2.7.3 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.

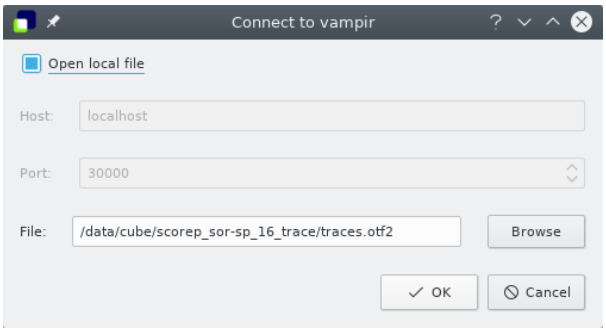

Figure 2.36: The dialog windows for a connection to a trace browser e.g. Vampir

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 2.37).

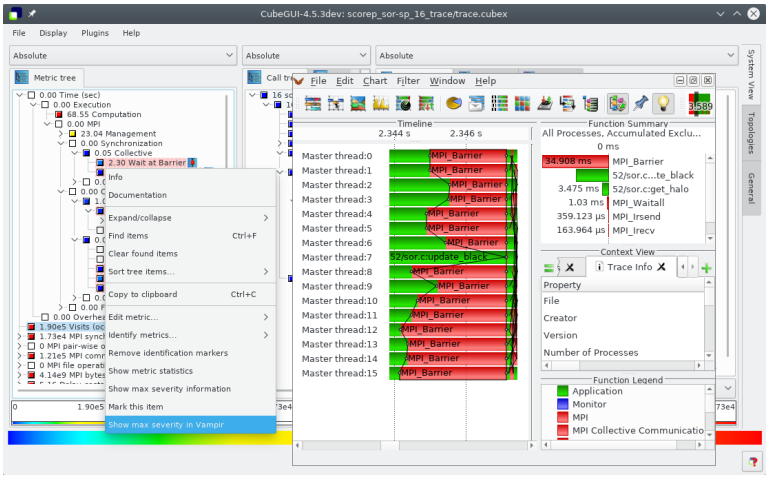

Figure 2.37: Context menu called on the metric "Wait at Barrier", showing the maximum severity in trace browser, which results in the location of the worst instance shown in the timeline display of Vampir.

2.7.3.1 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
```

2.7.4 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:

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

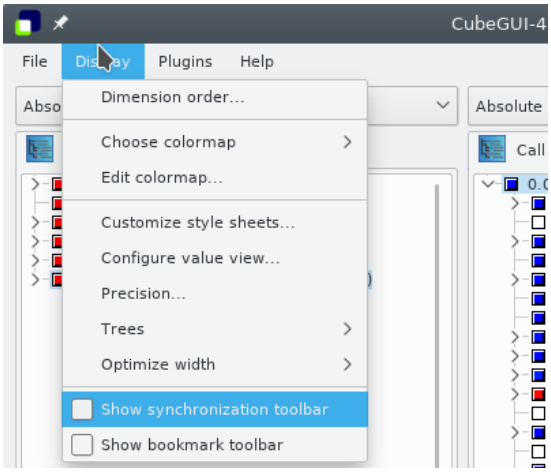

Figure 2.38: Enable Synchronization

To enable Synchronization, the corresponding toolbar has to be enabled (Figure 2.38). 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.

With the "Synchronize state" menu, you can select the information that is sent and received. By default, this is the state of the four 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 2.39).

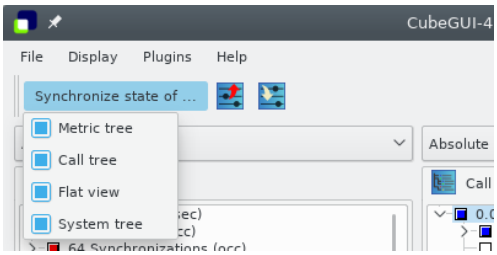

Figure 2.39: Synchronization toolbar

2.8 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>	*over menu/tool bar:* activate menu/function *over value mode combo:* select value mode *over tab:* switch to tab *in tree:* select/deselect/expand/collapse items
<right-mouse click>	*in tree:* context menu
Ctrl+<left-mouse click>	*in tree:* multiple selection/deselection
<left-mouse drag>	*over scroll bar:* scroll
Up arrow	*in tree:* move selection one item up (+Shift: multiple selection)
Down arrow	*in tree:* move selection one item down (+Shift: multiple selection)
Left arrow	*in scroll area:* scroll to the left
Right arrow	*in scroll area:* 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:

- 2.6.7
- 2.6.12.4

3 Cube Advisor Plugin

Advisor is a standard plugin and is available as long the measurement contains a *Time* metric. The main goal of the Advisor plugin is to provide a user the fast access to the various performance evaluations of the performance of their HPC application, such as 4, 25 or 26.

3.1 Getting Started with Advisor

As long the measurement contains metric *Time* CubeGUI will enable the Advisor plugin and it is available in "Others" tab in the plugins section.

If one would like to merge various measurements first and to analyze the result one starts with one of the context-free plugins, 2.6.2.3 or 2.6.2.2, and analyzes the result with the Advisor.

4 POP analysis

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 4.1, 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 4.1 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, e.g. by the user, or as part of a POP Proof-of-Concept activity.

The approach outlined here is applicable to various parallelism paradigms, however for simplicity the 4.1 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 (**??**).

4.1 POP Performance metrics

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 = **??** $*$ **??**

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`

5 AdvisorPOPTestsParallel_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 = **??** $*$ **??**

6 AdvisorPOPTestsLoad_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 = average computation time / maximum computation time

7 AdvisorPOPTestsCommunication_efficiency

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

CommE = maximum computation time / total run-time

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. serialization)
- Processes transferring large amounts of data relative to the network capacity

These are measured using **??** and **??**. In obtaining these two sub-metrics we first calculate (using the Dimemas simulator) how the code would behave if run on an idealized network where transmission of data takes zero time.

These two sub-metrics combine to give **Communication Efficiency**:

CommE = **??** $*$ **??**

8 AdvisorPOPTestsSerialization_efficiency

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

SerE = maximum computation time on ideal network / total run-time on ideal network

9 AdvisorPOPTestsTransfer_efficiency

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

TE = total run-time on ideal network / total run-time on real network

10 AdvisorPOPTestsIpc_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.

11 AdvisorPOPTestsStalled_resources

Stalled resources indicates number of computational cycles, where processor has been waiting for some resources to the total number of cycles. It is calculates as a ration of **PAPI_RES_STL** and **PAPI_TOT_CYC**

12 AdvisorPOPTestsComputationTime

Computation time indicated total time spend in the computation call path. Computation call path is a not one of the following call paths: MPI, OpenMP, pthreads, std::threads, CUDA, OpenCL, OpenACC, SchMEM. With another words, it is the user code.

Computation time indicated total time spend in the computation call path. Computation call path is a not one of the following call paths: MPI, OpenMP, pthreads, std::threads, CUDA, OpenCL, OpenACC, SchMEM. With another words, it is the user code.

13 AdvisorPOPTestsNoWaitINS_efficiency

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the **??**.

Instructions (only computation) indicates the number of CPU instructions executed in the computation code, which contributes to the **??**.

14 AdvisorPOPComputation_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 **??** and **??**.

15 AdvisorPOPInstruction_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.

16 AdvisorPOPTestsMissing_parallel_efficiency

?? 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 **parallel_efficiency**. In this case POP analysis is not possible.

17 AdvisorPOPTestsMissing_communication_effic

?? 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 **communication_efficiency**. In this case POP analysis is not possible.

18 AdvisorPOPTestsMissing_load_balance

?? 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 **missing_load_balance**. In this case POP analysis is not possible.

19 AdvisorPOPTestsMissing_ipc_efficiency

?? metric is available only, if one has collected **PAPI_TOT_INS** and **PAPI_TOT_CYC** counters while measurement. How to do it see `Score-P manual`

20 AdvisorPOPTestsMissing_serialization_efficien

?? 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**

21 AdvisorPOPTestsMissing_stalled_resources

?? metric is available only, if one has collected **PAPI_STL_RES** and **PAPI_TOT_CYC** counters while measurement. How to do it see `Score-P manual`

22 AdvisorPOPTestsMissing_transfer_efficiency

?? 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**

23 AdvisorPOPTestsMissingComputationTime

?? 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 **computation_time**.

24 AdvisorPOPTestsMissingNoWaitINS_efficiency

?? metric is available only, if one has collected **PAPI_TOT_INS** counters while measurement. How to do it see `Score-P manual`

25 KNL Vectorization analysis

We investigate loops with regard to their degree of vectorization and offer suggestions for optimization candidates. This required hardware counter measurements, obtained in multiple runs, due to the limited num ber of available counter registers. In the context of counter measurements this is not unusual for the Score-P work-flow. The suggestion of specific optimization candidates on the other hand is a deviation from the standard Score-P metric semantics.

The Score-P metric concept operates on the actual value of a metric (in absolute or relative terms) and analysis sometimes requires implicit information, e.g. if a higher value is worse than a small value. This approach leaves the decision about the rel- evance of a metric value of a certain call-path to the user. They need to judge the severity of an issue based on the knowledge of the hardware architecture, the source code, the input data, the use case, or even external parameters. Providing a generic set of thresholds, deciding if a metric value is problematic, is a hard problem in general, as too many parameters are involved, some outside the scope of the perfor- mance analysis tool.

In the case of vectorization assistance we used the cooperation with Intel R to investigate the use of explicit knowledge about the architecture for providing such thresholds in that limited context. In the following we describe the metrics we focused on and the challenges they pose for the Score-P work-flow and analysis.

25.1 KNL Vectorization metrics

We focus on the three metrics. The first metric calculates the computational density, i.e. the number of operations performed on average for each piece of loaded data. The **L1 compute to data access ratio** can be used to judge how suitable an application is to run on the KNL architecture. Ideally, operations should be vectorized and each datum fetched from L1 cache should be used for multiple operations.

Similar to this, the **L2 compute to data access ratio** is calculated as the number of vector operations against the loads that initially miss the L1 cache. While the L1 metric is critical in esti- mating a codes general suitability, the L2 metric is an indicator whether the code is operating efficiently.

The thresholds are considered the limits where an investigation into the code section?s vectorization would be useful. These limits are based on recommendations of Intel R [**?**] for the KNL architecture and while these hold true for most applications running on KNL, they are only guide- lines and should be applied with care.

An additional metric, the **VPU intensity**, offers a rule of thumb on how well a loop is vectorized, calculating the proportion of vectorized operations on total arithmetic operations. This metric should be applied only to small pieces of code and certain non-arithmetic operations, such as mask manipulation instructions, are counted as vector operations,

which can skew this ratio. One defines the metrics as ratios of hardware counters provided by the KNL architecture. These can be accessed in Score-P through the `PAPI metrics interface`

1. Metric: L1 Compute to data access ratio
   Threshold: < 1

```
UOPS RETIRED.PACKED SIMD/ MEM UOPS RETIRED.ALL LOADS
```

2. Metric: L2 Compute to data access ratio
   Threshold: < 100? L1 Compute to data access ratio

```
UOPS RETIRED.PACKED SIMD/ MEM UOPS RETIRED.L1 MISS LOADS
```

3. Metric: VPU intensity
   Threshold: < 0.5

```
UOPS RETIRED.PACKED SIMD/ (UOPS RETIRED.PACKED SIMD + UOPS RETIRED.SCALAR SIMD)
```

and can measured at a call-path level on each thread. To calculate all derived metrics, multiple native hardware counters have to be recorded. Since the KNL architecture provides only two general purpose counters per thread, multiple measurements have to be used to obtain the full set of counters required.

26 KNL Memory usage analysis

With Score-P, we measure the bandwidth values per code-region outside of OpenMP parallel regions, due the given uncore counter restrictions. Depending on the application, there might be a lot of code regions that show a high band- width value. To find the most bandwidth sensitive candidates among these regions, we need to sort them by their last-level cache-misses (LLC). This gives us the MCDRAM candidate metric per code region, as shown in Figure 4. We derive the MCDRAM candidate metric, i.e., we sort the high bandwith callpaths by their last- level cache misses, in the Cube plugin KNL advisor (see also 5.2). As input we use the PAPI-measured access counts for each DDR4 memory channel and the PAPI-SCIPHI Score-P and Cube extensions for Intel Phi measured LLC counts. We take care of measuring the memory accesses only per- process while running exclusively on a single KNL node. As Score-P and Cube purely work on code regions, the MCDRAM candidates are also code regions. As a drawback, if a candidate code region accesses several data structures, we cannot point to the most bandwidth sensitive structure. Vtune [1], HPCToolkit [3][12] or ScaAnalyzer [13] might provide more detailed insight. In addition to this drawback, the above approach is not generally applicable for tools as accessing counters from the uncore requires priviledged access to a ma- chine, either by setting the paranoia flag or by providing a special kernel module. On production machines, this access is, for security reasons, often not granted. This does not only apply to memory accesses, but to all uncore counters.

27 Memory analysis for KNL

In order to analyse quality of the computational code Advisor requires that at least the following metrics are collected:

1. LLC_MISSES
2. knl_unc_imc0::UNC_M_CAS_COUNT:ALL:cpu=0,
3. knl_unc_imc1::UNC_M_CAS_COUNT:ALL:cpu=0,
4. knl_unc_imc2::UNC_M_CAS_COUNT:ALL:cpu=0,
5. knl_unc_imc3::UNC_M_CAS_COUNT:ALL:cpu=0,
6. knl_unc_imc4::UNC_M_CAS_COUNT:ALL:cpu=0 and
7. knl_unc_imc5::UNC_M_CAS_COUNT:ALL:cpu=0

How to do it see `Score-P manual`

28 Vectorization analysis for KNL

In order to analyse quality of the computational code Advisor requires that at least following metrics are collected:

1. MEM_UOPS_RETIRED:L1_MISS_LOADS,
2. MEM_UOPS_RETIRED:L2_MISS_LOADS
3. UOPS_RETIRED:PACKED_SIMD,
4. UOPS_RETIRED:SCALAR_SIMD

How to do it see `Score-P manual`

29 Memory transfer

Number of transferred bytes.

30 Missing Memory transfer?

29 metric is available only, if one has collected **knl_unc_imc0::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc1::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc2::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc3::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc4::UNC_M_CAS_COUNT:ALL:cpu=0 and knl_unc_imc5::UNC_M_CAS_COUNT:ALL:cpu=0** counters while measurement.
How to do it see `Score-P manual`

31 Memory bandwidth

Number of transferred bytes per runtime of the call path.

32 Missing Memory bandwidth?

31 metric is available only, if one has collected **knl_unc_imc0::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc1::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc2::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc3::UNC_M_CAS_COUNT:ALL:cpu=0, knl_unc_imc4::UNC_M_CAS_COUNT:ALL:cpu=0 and knl_unc_imc5::UNC_M_CAS_COUNT:ALL:cpu=0** counters while measurement.
How to do it see `Score-P manual`

33 LLC Miss metric

Displays number of misses in last level cache.

34 Missing LLC Miss metric?

33 metric is available only, if one has collected **LLC_MISSES** counters while measurement. How to do it see `Score-P manual`

35 VPU Intensity

VPU intensity offers a rule of thumb on how well a loop is vectorized, calculating the proportion of vectorized operations on total arithmetic operations. This metric should be applied only to small pieces of code and certain non-arithmetic operations, such as mask manipulation instructions, are counted as vector operations, which can skew this ratio.

```
UOPS RETIRED.PACKED SIMD/ (UOPS RETIRED.PACKED SIMD + UOPS RETIRED.SCALAR SIMD)
```

---

36 Missing VPU Intensity?

35 metric is available only, if one has collected **UOPS_RETIRED.PACKED_SIMD, UOPS_RETIRED.SCALAR_SIMD** counters while measurement. How to do it see `Score-P manual`

37 L1 to Computation ratio

The **L1 compute to data access ratio** can be used to judge how suitable an application is to run on the KNL architecture. Ideally, operations should be vectorized and each datum fetched from L1 cache should be used for multiple operations.

38 Missing L1 to Computation ratio

37 metric is available only, if one has collected **UOPS_RETIRED.PACKED_SIMD, MEM_UOPS_RETIRED.ALL_LOADS** counters while measurement. How to do it see `Score-P manual`

39 L2 to L1 ratio

Similar to this, the **L2 compute to data access ratio** is calculated as the number of vector operations against the loads that initially miss the L1 cache. While the L1 metric is critical in esti- mating a codes general suitability, the L2 metric is an indicator whether the code is operating efficiently.

40 Missing L2 to L1

39 metric is available only, if one has collected **UOPS_RETIRED.PACKED_SIMD, MEM_UOPS_RETIRED.L1_MISS_LOADS** counters while measurement. How to do it see `Score-P manual`

41 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*.

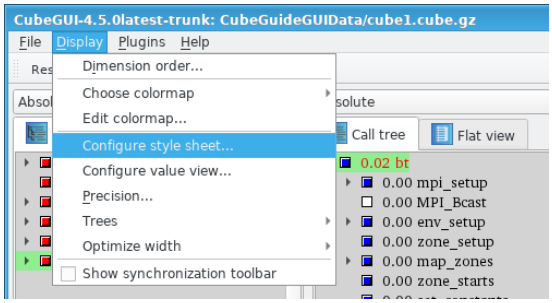

Figure 41.1: start style sheet editor

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

42 Appendix

42.1 File format of statistics files

Statistic files (for an example see 42.1) 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

WaitAtIBarrier     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 42.1: 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.

www.scalasca.org