

# The hpcviewer User Interface

The HPCToolkit Project Team

February 12, 2009

## Table of Contents

1. [Introduction](#)
2. [Menus](#)
3. [Views](#)
4. [Panes](#)
5. [Understanding Metrics](#)
6. [Derived Metrics](#)
7. [For Convenience Sake](#)
8. [Limitation](#)

## Introduction

HPCToolkit provides the **hpcviewer** browser for interactive examination of performance databases. The viewer can be launched from a command line (Linux/Unix platform) or by clicking the hpcviewer icon (for Windows, Mac OS X and Linux/Unix platform). The command line syntax is as follows:

```
hpcviewer [<database-directory>]
```

<database-directory> is an optional argument to load a database automatically. Without this option, the viewer will prompt the location of a database.

## Menus

hpcviewer provides three main menus:

### File

This menu includes several menu items for controlling basic viewer operations.

- **New window**. Open a new hpcviewer window that is independent from the existing one.
- **Open database ...**. Load a performance database into the current hpcviewer window.
- **Preferences ...**. Display the settings dialog box.
- **Exit**. Quit the hpcviewer application.

### Debug

This menu enables one to request display of HPCToolkit's raw XML representation for performance data. (This operation is intended primarily for tool developer use.)

## Help

This menu displays information about the viewer. The menu contains two items:

- **About**. Displays brief information about the viewer, including used plug-ins and error log.
- **hpcviewer help**. This document.

Figure 1 shows a screenshot of an hpcviewer browser window with panes and key controls labeled. The browser supports three different views of performance data. The browser window is divided into three panes. We first explain the views of performance data and then the role of the different panes.

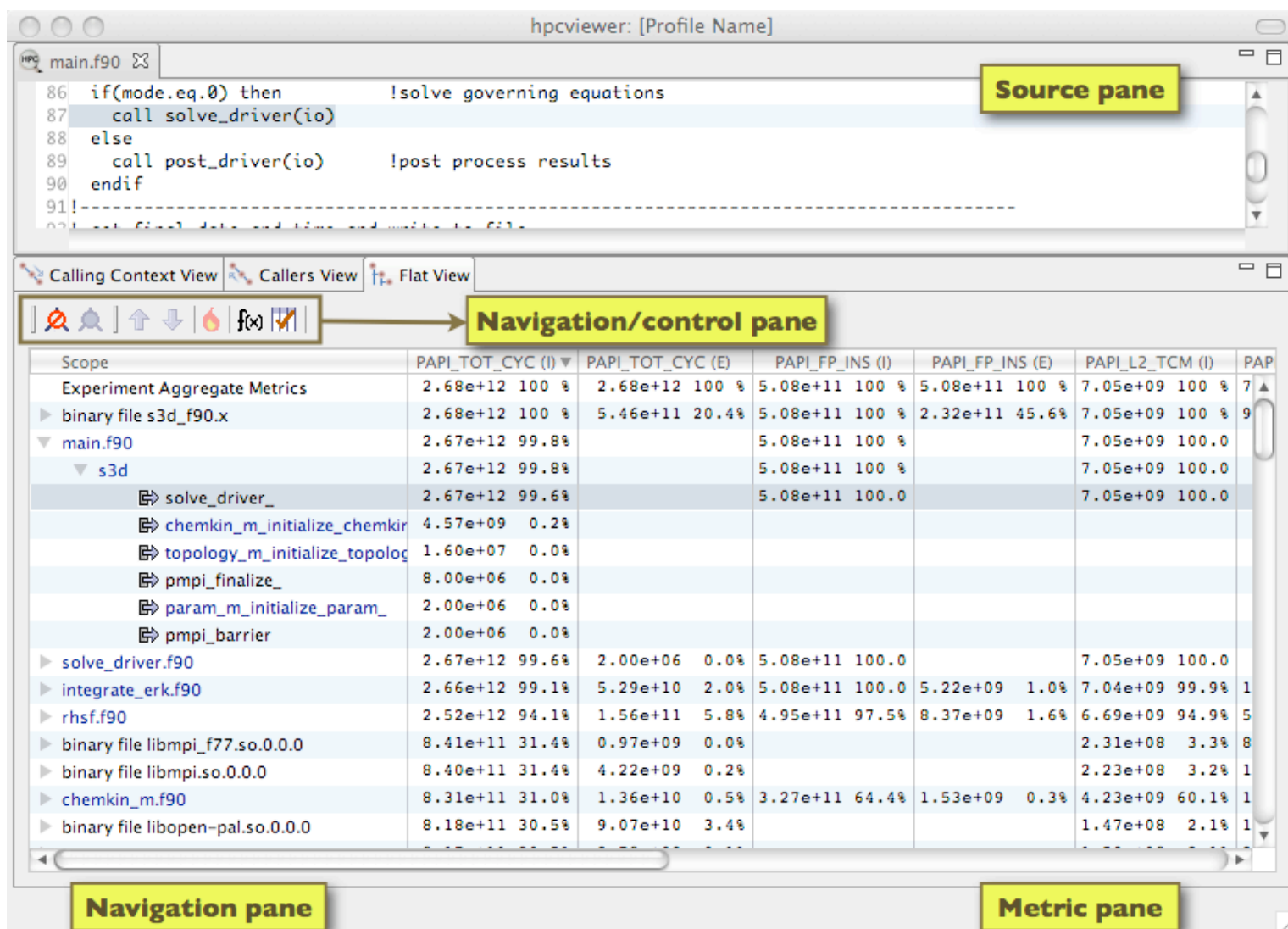


Figure 1: A snapshot of hpcviewer

## Views

hpcviewer supports three principal views of an application's performance data: a top-down calling context view, a bottom-up caller's view, and a flat view. One selects the desired view by clicking on the corresponding view control tab. We briefly describe the three views and their corresponding purposes.

- Calling context view.** This top-down view represents the dynamic calling contexts (call paths) in which costs were incurred. Using this view, one can explore performance measurements of an application in a top-down fashion to understand the costs incurred by calls to a procedure in a particular calling context. We use the term "cost" rather than simply "time" since hpcviewer can present a multiplicity of measured such as cycles, or cache misses) or derived metrics (e.g. cache miss rates or bandwidth consumed) that are other indicators of execution cost.  
 A calling context for a procedure  $f$  consists of the stack of procedure frames active when the call was made to  $f$ . Using this view, one can readily see how much of the application's cost was incurred by  $f$  when called from a particular calling context. If finer detail is of interest, one can explore how the costs incurred by a call to  $f$  in a particular context are divided between  $f$  itself and the procedures it calls. HPCToolkit's call path profiler *hpcprof* and the hpcviewer user interface distinguish calling context precisely by individual call sites; this means that if a procedure  $g$  contains calls to procedure  $f$  in different places, these represent separate calling contexts.
- Callers view.** This bottom up view enables one to look upward along call paths. This view is particularly useful for understanding the performance of software components or procedures that are used in more than one context. For instance, a message-passing program may call `MPI_wait` in many different calling contexts. The cost of any particular call will depend upon the structure of the parallelization in which the call is made. Serialization or load imbalance may cause long waits in some calling contexts while other parts of the program may have short waits because computation is balanced and communication is overlapped with computation.
- Flat view.** This view organizes performance measurement data according to the static structure of an application. All costs incurred in any calling context by a procedure are aggregated together in the flat view. This complements the calling context view, in which the costs incurred by a particular procedure are represented separately for each call to the procedure from a different calling context.

## Panes

The browser window is divided into three panes: the **navigation** pane, the **source** pane, and the **metrics** pane. We briefly describe the role of each pane.

### Source pane

The source pane displays the source code associated with the current entity selected in the navigation pane. When a performance database is first opened with hpcviewer, the source pane is initially blank because no entity has been selected in the navigation pane. Selecting any entity in the navigation pane will cause the source pane to load the corresponding file, scroll to and

highlight the line corresponding to the selection. Switching the source pane to view to a different source file is accomplished by making another selection in the navigation pane.






## Navigation pane

The navigation pane presents a hierarchical tree-based structure that is used to organize the presentation of an applications's performance data. Entities that occur in the navigation pane's tree include load modules, files, procedures, procedure activations, inlined code, loops, and source lines. Selecting any of these entities will cause its corresponding source code (if any) to be displayed in the source pane. One can reveal or conceal children in this hierarchy by "opening" or "closing" any non-leaf (i.e., individual source line) entry in this view.

The nature of the entities in the navigation pane's tree structure depends upon whether one is exploring the calling context view, the callers view, or the flat view of the performance data.

- In the **calling context view**, entities in the navigation tree represent procedure activations, inlined code, loops, and source lines. While most entities link to a single location in source code, procedure activations link to two: the call site from which a procedure was called and the procedure itself.
- In the **callers view**, entities in the navigation tree are procedure activations. Unlike procedure activations in the calling context tree view in which call sites are paired with the called procedure, in the caller's view, call sites are paired with the calling procedure to facilitate attribution of costs for a called procedure to multiple different call sites and callers.
- In the **flat view**, entities in the navigation tree correspond to source files, procedure call sites (which are rendered the same way as procedure activations), loops, and source lines.

The header above the navigation pane contains some controls for the navigation and metric view. In Figure 1, they are labeled as "navigation/metric control".

-  **Flatten** /  **unflatten** (available for the flat view): Enabling to flatten and unflatten the navigation hierarchy. Clicking on the flatten button (the icon that shows a tree node with a slash through it) will replace each top-level scope shown with its children. If a scope has no children (i.e., it is a leaf), the node will remain in the view. This flattening operation is useful for relaxing the strict hierarchical view so that peers at the same level in the tree can be viewed and ranked together. For instance, this can be used to hide procedures in the flat view so that outer loops can be ranked and compared to one another. The inverse of the flatten operation is the unflatten operation, which causes an elided node in the tree to be made visible once again.
-  **Zoom-in** /  **Zoom-out**: Depressing the up arrow button will zoom in to show only information for the selected line and its descendants. One can zoom out (reversing a prior zoom operation) by depressing the down arrow button
-  **Hot call path**: Finding for the cost of performance hot-spots, and is used to show the chain of responsibility for costs. The "hot spot" is computed by comparing parent and child values, and show the chain where the difference is greater than a threshold (by default is 50%). It is also possible to change the threshold value by clicking the menu "File-Preference".

- **f(x) Derived metric:** Creating a new metric based on mathematical formula. See [Derived metrics](#) section for more details.
- **Hide/show metrics:** Showing and hiding metric columns. A dialog box will appear, and user can select which columns to show or hide. See [hide/show column](#) section for more details

## Metric pane

The metric pane displays one or more performance metrics associated with entities to the left in the navigation pane. Entities in the tree view of the navigation pane are sorted at each level of the hierarchy by the metric in the selected column. When *hpcviewer* is launched, the leftmost metric column is the default selection and the navigation pane is sorted according to the values of that metric in descending order. One can change the selected metric by clicking on a column header. Clicking on the header of the selected column toggles the sort order between descending and ascending.

During analysis, one often wants to consider the relationship between two metrics. This is easier when the metrics of interest are in adjacent columns of the metric pane. One can change the order of columns in the metric pane by selecting the column header for a metric and then dragging it left or right to its desired position. The metric pane also includes scroll bars for horizontal scrolling (to reveal other metrics) and vertical scrolling (to reveal other scopes). Vertical scrolling of the metric and navigation panes is synchronized.

## Understanding Metrics

The data gathered by the profiler attributes the cost for each scope (a file, procedure, loop, or inlined function) exclusively. *hpcviewer* presents inclusive values for each cost metric associated with a program scope as well. Exclusive costs are those incurred by scope itself; whereas inclusive costs include costs incurred by any calls it makes.

### How metrics are computed?

Call path profile measurements collected by *hpcrun* correspond directly to the calling context view. *hpcviewer* derives all other views from exclusive metric costs in the calling context view. For the caller view, *hpcviewer* collects the cost of all samples in each function and attribute that to a top-level entry in the caller view. Under each top-level function, *hpcviewer* can look up the call chain at all of the context in which the function is called. For each function, *hpcviewer* apportions its costs among each of the calling contexts in which they were incurred. *hpcviewer* computes the flat view by traversing the calling context tree and attributing all costs for a scope to the scope within its static source code structure. The flat view presents a hierarchy of nested scopes from the load module, file, routine, loops, inlined code and statements.

### Example

**Figure 2a: file1.c**

Figure 2a: file1.c

**Figure 2b: file2.c**

```

r () {
    g ();
}
// m is the main routine
m () {
    f ();
    g ();
}

// g can be a recursive function
g () {
    if ( . . . ) g ();
    if ( . . . ) h ();
}
h () {
}

```

Figure 3: Calling context view

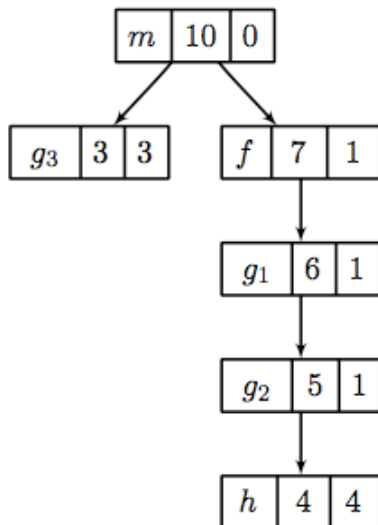


Figure 4: Caller view

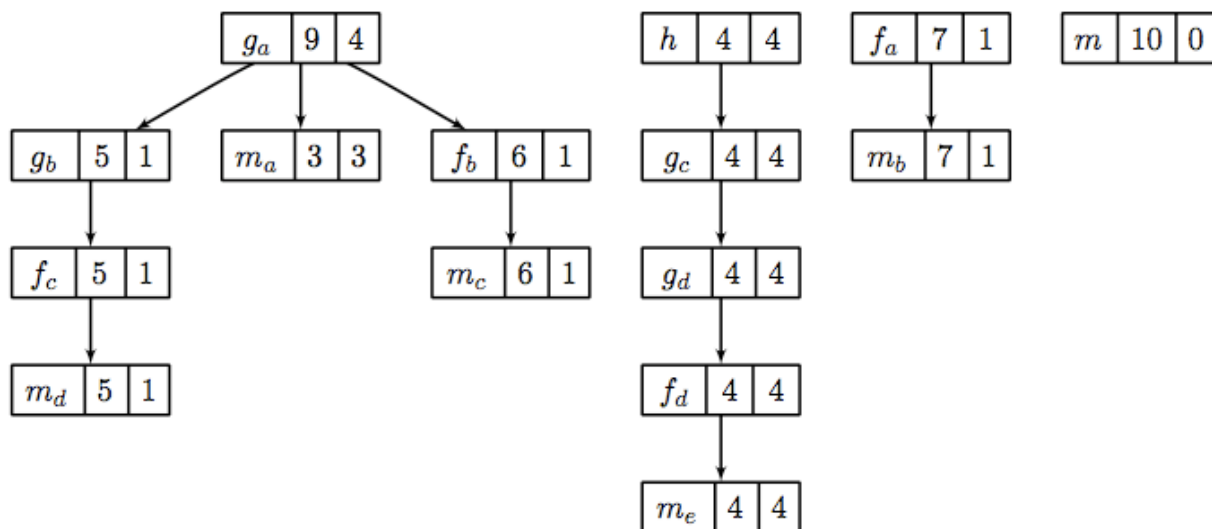


Figure 5: Flat view

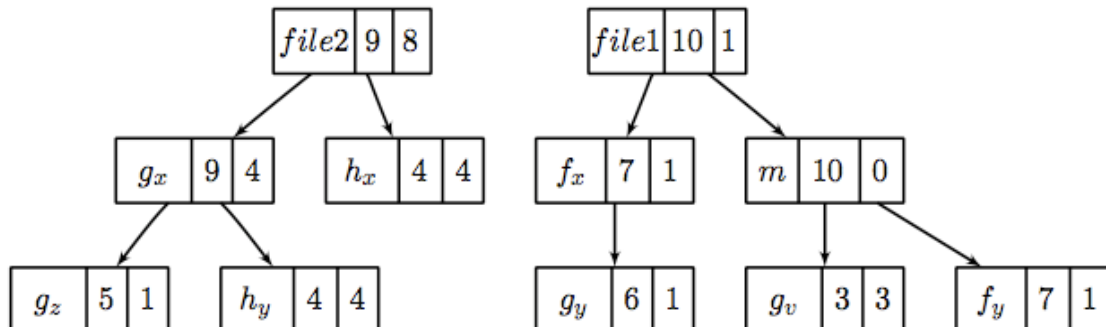


Figure 2 (a and b) shows an example of a recursive program separated into two files: `file1.c` and `file2.c`. In this figure, we use numerical subscripts to distinguish between different instances of the same procedure. In the other parts of this figure, we use alphabetic subscripts. We use different labels because there is no natural one-to-one correspondence between the instances in the different views.

Routine  $g$  (Figure 2b) can behave as a recursive function depending on the value of the condition branch (Line 3-4). Figure 3 shows an example of the call chain execution of the program annotated with both inclusive and exclusive costs. Computation of inclusive costs from exclusive costs in the calling context view involves simply summing up all of the costs in the subtree below.

In this figure, we can see that on the right path of the routine  $m$ , routine  $g$  (instantiated in the diagram as  $g_1$ ) performed a recursive call ( $g_2$ ) before calling routine  $h$ . Although  $g_1$ ,  $g_2$  and  $g_3$  are all instances from the same routine (i.e.,  $g$ ), we attribute a different cost for each instance. This separation of cost can be critical to identify which instance has a performance problem.

Figure 4 shows the corresponding scope structure for the caller view and the costs we compute for this recursive program. The procedure  $g$  noted as  $g_a$  (which is a root node in the diagram), has different cost to  $g$  as a callsite as noted as  $g_b$ ,  $g_c$  and  $g_d$ . For instance, on the first tree of this figure, the inclusive cost of  $g_a$  is 9, which is the sum of the highest cost for each branch in calling context tree (Figure 3): the inclusive cost of  $g_3$  (which is 3) and  $g_1$  (which is 6). We do not attribute the cost of  $g_2$  here since it is a descendant of  $g_1$  (in other term, the cost of  $g_2$  is included in  $g_1$ ).

Inclusive costs need to be computed similarly in the flat view. The inclusive cost of a recursive routine is the sum of the highest cost for each branch in calling context tree. For instance, in Figure 5, The inclusive cost of  $g_x$ , defined as the total cost of all instances of  $g$ , is 9, and this is consistently the same as the cost in caller tree. The advantage of attributing different costs for each instance of  $g$  is that it enables a user to identify which instance of the call to  $g$  is responsible for performance losses.

## Derived Metrics

Frequently, the data become useful only when combined with other information such as the number of instructions executed or the total number of cache accesses. While users don't mind a bit of mental arithmetic and frequently compare values in different columns to see how they relate for a scope, doing this for many scopes is exhausting. To address this problem, *hpcviewer* provides a mechanism for defining metrics. A user-defined metric is called a "derived metric". A derived metric is defined by specifying a spreadsheet-like mathematical formula that refers to data in other columns in the metric table by using  $s_n$  to refer to the value in the  $n^{\text{th}}$  column.

## Formula

The formula supported by *hpcviewer* is mainly inspired by spreadsheet-like mathematical

formula, and can be in any form of combined complex expression. The syntax is very simple:

```
<expression> ::= <binary_op> | <function>
<binary_op> ::= <expression> <binary_operand> <expression>
<binary_operand> ::= + | - | * | /
```

## Intrinsic Functions

The list of intrinsic functions supported can be found in [here](#). Creating a new intrinsic function requires adjusting the source code. If you want to do it yourself, source code for the viewer is available at <https://outreach.scidac.gov>. Otherwise, you can contact the HPCToolkit team at [hpc@rice.edu](mailto:hpc@rice.edu).

## Examples

Suppose the database contains information about 5 processes, each with two metrics:

1. Metric 0, 2, 4, 6 and 8: total number of cycles
2. Metric 1, 3, 5, 7 and 9: total number of floating point operations

To compute the average number of cycles per floating point operation across all of the processes, we can define a formula as follows:

```
avg($0, $2, $4, $6, $8) / avg($1, $3, $5, $7, $9)
```

## Derived metric dialog box

A derived metric can be created by clicking the **Derived metric** tool item in the navigation/control pane. A derived metric window will then appear as shown in Figure 6 below.



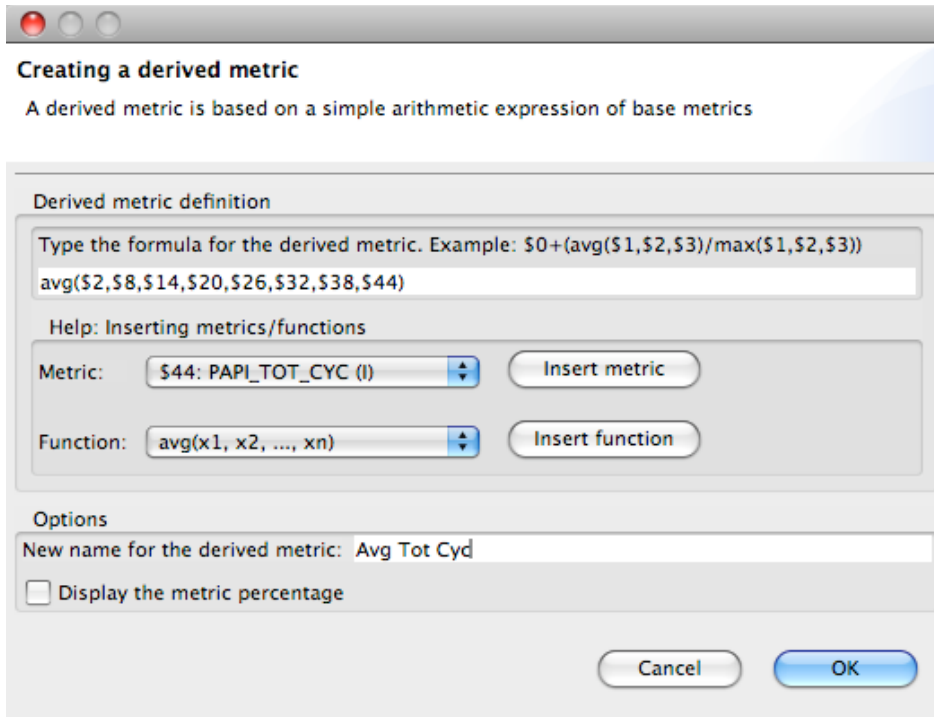


Figure 6: Derived metric dialog box

The window has two main parts:

**Derived metric definition**, which consists of:

- *Formula definition field.* In this field the user can define a formula with spreadsheet-like mathematical formula. This field is required to be filled.
- *Metric help.* This is used to help the user to find the *ID* of a metric. For instance, in this snapshot, the metric `PAPI_TOT_CYC` has the ID 44. By clicking the button **Insert metric**, the metric ID will be inserted in formula definition field.
- *Function help.* This help is to guide the user to insert functions in the formula definition field. Some functions require only one metric as the argument, but some can have two or more arguments. For instance, the function `avg()` which computes the average of some metrics, need to have two arguments.

**Options**, offers two customizations:



- *New name for the derived metric.* Supply a string that will be used as the column header for the derived metric. If you don't supply one, the metric will have no name.
- *Display the metric percentage.* When this box is checked, each scope's derived metric value will be augmented with a percentage value, which for scope *s* is computed as the  $100 * (s\text{'s derived metric value}) / (\text{the derived metric value computed by applying the metric formula to the aggregate values of the input metrics})$  the entire execution). Such a computation can lead to nonsensical results for some derived metric formulae. For instance, if the derived metric is computed as a ratio of two other metrics, the aforementioned computation that compares the scope's ratio with the ratio for the entire program won't yield a meaningful result. To avoid a confusing metric display, think before you use this button to annotate a metric with its percent of total.

## For Convenience Sake

In this section we describe some features of hpcviewer that help improve productivity.

### Editor pane


The editor pane is used to display *a copy* of your program's source code or HPCToolkit's performance data in XML format; for this reason, it does not support editing of the pane's contents. To edit your program, you should use your favorite editor to edit *your* original copy of the source, not the one stored in HPCToolkit's performance database. Thanks to built-in capabilities in Eclipse, hpcviewer supports some useful shortcuts and customization:

- **Go to line.** To scroll the current source pane to a specific line number, `<ctrl>-l` (on Linux and Windows) or `<command>-l` (Mac) will bring up a dialog that enables you to enter the target line number.
- **Find.** To search for a string in the current source pane, `<ctrl>-f` (Linux and Windows) or `<command>-f` (Mac) will bring up a find dialog that enables you to enter the target string.
- **Font.** You can change the font used by hpcviewer for the metric table using the Preferences dialog from the File menu. Once you've opened the Preferences dialog, select *hpcviewer preferences* (the item at the bottom of the list in the column on the left side of the pane). The new font will take effect when you next launch hpcviewer.
- **Minimize/Maximize window.** Icons in the upper right corner of the window enable you to minimize () or maximize () the hpcviewer window.

### Metric pane

For the metric pane, hpcviewer has some convenient features:

- **Maximizing a view.** To expand the source or metric pane to fill the window, one can double click on the tab with the view name. Double clicking again on the view name will restore the view back to its original size.
- **Sorting the metric pane contents by a column's values.** First, select the column on which you wish to sort. If no triangle appears next to the metric, click again. A downward pointing triangle means that the rows in the metric pane are sorted in descending order according to the column's value. Additional clicks on the header of the selected column will toggle back and forth between ascending and descending.
- **Changing column width.** To increase or decrease the width of a column, first put the cursor over the right or left border of the column's header field. The cursor will change into a vertical bar between a left and right arrow. Depress the mouse and drag the column border to the desired position.
- **Changing column order.** If it would be more convenient to have columns displayed in a different order, they can be permuted as you wish. Depress and hold the mouse button over the header of column that you wish to move and drag the column right or left to its new position.
- **Hiding or showing metric columns.** Sometimes, it may be more convenient to suppress the display of metrics that are not of current interest. When there are too many metrics to

fit on the screen at once, it is often useful to suppress the display of some. The icon  above the metric pane will bring up the column selection dialog shown below.

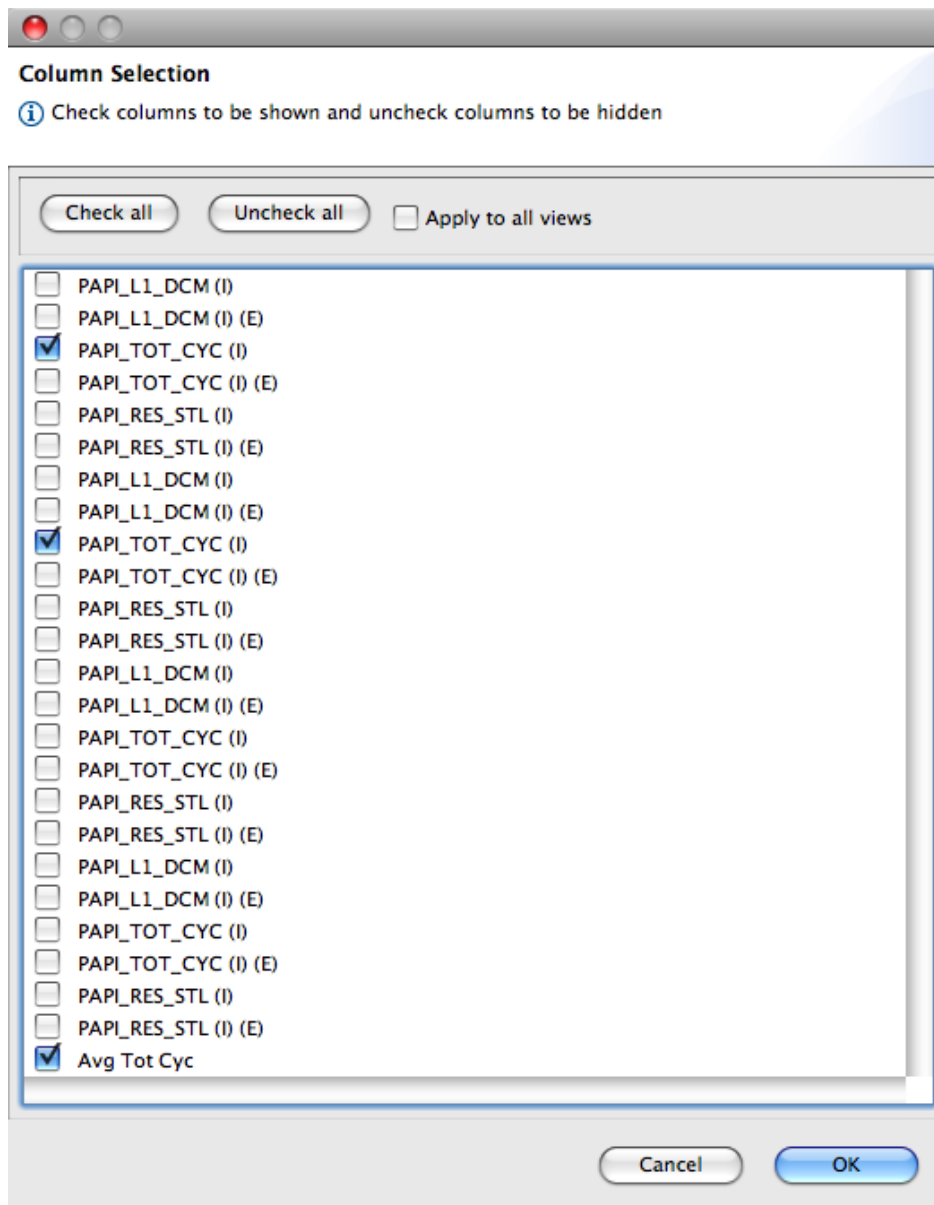


Figure 7: Hide/Show columns dialog box

The dialog box contains a list of metric columns sorted according to their order in HPCToolkit's performance database for the application. Each metric column is prefixed by a check box to indicate if the metric should be *displayed* (if checked) or *hidden* (unchecked). To display all metric columns, one can click the **Check all** button. A click to **Uncheck all** will hide all the metric columns.

Finally, an option **Apply to all views** will set the configuration into all views when checked. Otherwise, the configuration will be applied only on the current view.

## Limitations

As of this writing, hpcviewer has some limitations:

- **Limited number of metrics.** Although most hpctoolkit components such as *hpcrun* and *hpcstruct* support large number of metrics, it is not recommended to try to work with more than 100 metrics (including exclusive and inclusive variants) using *hpcviewer*. Having a huge number of metrics will not only require a large amount of memory, it will also make the viewer interface sluggish. If a profiled application has more than 50 processes and each process has its own metrics, we recommend analyzing only a representative few processes. To pinpoint scalability bottlenecks, we recommend an approach based on differential analysis of a representative process from two executions at different scales as described in reference [2] below.
- **Copying or printing metrics.** hpcviewer does not currently support copying or printing metric values.

## References

1. <http://www.hpctoolkit.org>
2. Coarfa, C., Mellor-Crummey, J., Froyd, N., and Dotsenko, Y. 2007. [Scalability analysis of SPMD codes using expectations](#). In Proceedings of the 21st Annual International Conference on Supercomputing (Seattle, Washington, June 17 - 21, 2007). ICS '07. ACM, New York, NY, 13-22.