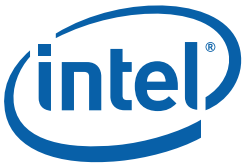


Intel® Trace Analyzer

Reference Guide

Document Number: 318120-004

Revision: 8.0

World Wide Web: http://www.intel.com

Disclaimer and Legal Notices

Revision History

Document Number	Revision Number	Description	Revision Date
318120-001	7.1	Initial draft	/10/16/2007
318120-002	7.2	Sections, "Correctness Checking Reports" and "Intel® Trace Collector Configuration Assistant" are added.	/09/18/2009
318120-003	8.0 Beta	Sections, "Intended Audience", "Conventions and Symbols", "Project Menu", "Trace Idealizer Dialog", "Application Imbalance Diagram Dialog" and "Custom Plug-in Framework" are added. Sections, "Command Line Interface (CLI)" and "View Main Menu" are updated.	/11/03/2009
318120-004	8.0	Sections, "Application Imbalance Diagram Dialog Box" and "Project Menu" are updated.	/02/16/2010

Contents

List of Figures

1 Introduction

Intel® Trace Analyzer is a graphical tool that displays and analyzes event trace data generated by the Intel® Trace Collector. It helps detecting performance problems, programming errors and understanding the behavior of the application. This document describes the feature set of Intel Trace Analyzer.

1.1 Intended Audience

This *Reference Guide* helps you understand the feature of the Intel® Trace Analyzer and how to use this tool to analyze your application performance.

1.2 Conventions and Symbols

The following conventions are used in this document.

Table 1.21-1 Conventions and Symbols used in this Document

This type style	Document or product names
This type style	Hyperlinks
`This type style`	Commands, arguments, options, file names
`THIS_TYPE_STYLE`	Environment variables
`<this type style>`	Placeholders for actual values
`[ items ]`	Optional items
`{ item | item }`	Selectable items separated by vertical bar(s)
(SDK only)	For Software Development Kit (SDK) users only

1.3 Notation and Terms

Menus and menu entries are printed as shown below:

Main Menu > File > Exit

This denotes the **Exit** entry in the **File** section of the **Main Menu**.

Shell commands are printed with a leading `$`. For example,

```
$ ls
```

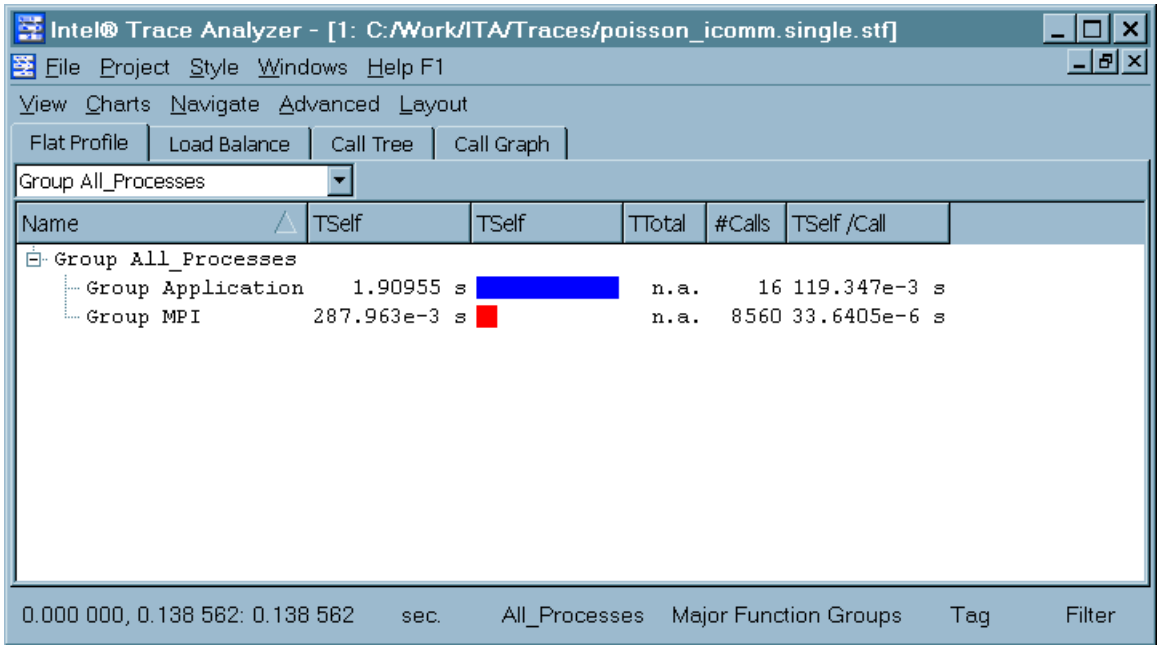
denotes the UNIX command `ls`.

The following line is an example of how source code is presented:

```
CALL MPI_FINALIZE()
```

The term process in this documentation implicitly includes thread. As soon as Intel Trace Analyzer loads a trace file that was generated running a multithreaded application, the GUI uses the term thread only if it is applicable. This is done to avoid confusing MPI application programmers who normally use the term process instead of thread.

Figure 1.1 Intel® Trace Analyzer



1.4 Related Information

Additional information about Intel® Trace Analyzer for Linux* OS and related products are available at: *http://support.intel.com/support/performancetools/cluster/analyzer/lin/index.htm*

Additional information about Intel® Trace Analyzer for Windows* OS and related products are available at: *http://support.intel.com/support/performancetools/cluster/analyzer/win/index.htm*

Intel® Premier support is available at: *https://premier.intel.com/*

1.5 Starting Intel® Trace Analyzer

NOTE: For a proper functioning of Intel® Trace Analyzer you must ensure that files do not get modified while they are opened in Intel Trace Analyzer.

1.5.1 Starting in UNIX* Environment

In a UNIX* environment, invoke Intel® Trace Analyzer through the command line by typing

```
$ traceanalyzer
```

Assuming that the executable is found, this command launches Intel Trace Analyzer. Optionally, one or more trace files are specified as arguments as shown in the following example:

```
$ traceanalyzer poisson_icomm.single.stf &
```

The above example opens the trace file `poisson_icomm.single.stf` in Intel Trace Analyzer (*Figure* 1.1). To open trace files in Intel Trace Analyzer without restarting it, use the File Menu entry **Main Menu > File > Open**. Each time a trace file is opened a new window displaying the trace file's Function Profile Chart is displayed (*Figure* 1.2).

1.5.2 Starting in Windows* Environment

In Windows* environment, double-click on a trace file to invoke Intel® Trace Analyzer or use the menu item **Start > All Programs > Intel Trace Analyzer**.

1.5.3 Trace Cache

For every trace file opened by Intel® Trace Analyzer, a trace cache is created for more efficient access. It is stored in the same directory as the trace file and has the same file name but with suffix `.cache` added.

1.5.4 Command Line Interface (CLI)

Intel® Trace Analyzer provides a command line interface (CLI) in Windows* and UNIX* environments. One application of the CLI is to pre-calculate trace caches for new trace files from batch files without invocation of the graphical user interface. This might be useful for very big trace files. You might also want to automate producing profiling data for several trace files without further interaction. For details, refer to Command Line Interface (CLI).

1.6 Internationalization

Intel® Trace Analyzer target audience is pretty small compared to, say, a word processor and the same installation on a parallel computer will often be used in diverse communities. The GUI is as agnostic as possible when it comes to internationalization (or i18n for short).

There is only English version of the software and the documentation. Also, the number formats in the GUI (refer to Style Menu and Number Formatting Settings) and in exported text files always use a decimal point and when separating three digit groups a space character is used instead of a comma or point.

The problem of non-ASCII characters in paths and file names is a little more complicated. To be able to deal with file and path names that contain characters from most languages of the world, Intel Trace Analyzer uses UTF-8 encoding internally and in its configuration file.

Since Windows* OS uses the same encoding (UTF-16) in file and path names regardless of your language and region settings you do not need to configure anything to make Intel Trace Analyzer work in your environment. As soon as the file names appear correctly in the folder listings they will be usable in Intel Trace Analyzer.

On Linux* systems you will have to set the encoding part in your locale settings correctly to be able to open your trace files. If you use plain ASCII in your path names then you are already done. On recent Linux* systems chances are high that everything is set up correctly out of the box.

Valid locale settings look like `en_US.latin-1` for U.S., `de_DE.UTF-8` for Germany, `fr_CA.iso88591` for French-speaking people in Canada and `ja_JP.ujis` for Japan. The command `locale` will help to list all available locales and encodings on your system. For Intel Trace Analyzer only the encoding part of the locale setting is relevant. All file names and path names given on the command line and read from the file system are expected to be encoded in the encoding given by the locale settings.

CAUTION: Having, for example UTF-8 encoded folder names and storing trace files there with ISO-8859-1 encoded file names can lead to inaccessible files.

Intel Trace Analyzer supports the following encodings (but use the encoding names as given by `locale -m`):

- Latin1
- Big5 — Chinese
- Big5-HKSCS — Chinese
- eucJP — Japanese
- eucKR — Korean
- GB2312 — Chinese
- GBK — Chinese
- GB18030 — Chinese
- JIS7 — Japanese
- Shift-JIS — Japanese
- TSCII — Tamil
- utf8 — Unicode, 8-bit
- utf16 — Unicode
- KOI8-R — Russian
- KOI8-U — Ukrainian
- ISO8859-1 — Western
- ISO8859-2 — Central European
- ISO8859-3 — Central European
- ISO8859-4 — Baltic
- ISO8859-5 — Cyrillic
- ISO8859-6 — Arabic
- ISO8859-7 — Greek
- ISO8859-8 — Hebrew, visually ordered
- ISO8859-8-i — Hebrew, logically ordered
- ISO8859-9 — Turkish
- ISO8859-10
- ISO8859-13
- ISO8859-14
- ISO8859-15 — Western
- IBM 850
- IBM 866
- CP874
- CP1250 — Central European
- CP1251 — Cyrillic
- CP1252 — Western
- CP1253 — Greek

- CP1254 — Turkish
- CP1255 — Hebrew
- CP1256 — Arabic
- CP1257 — Baltic
- CP1258
- TIS-620 — Thai

Summary: to make your life simple when moving trace files between different systems and sharing them among people it is recommended that you use pure ASCII characters in path and file names as much as possible. When using non-ASCII characters in path names seems unavoidable, use UTF-8 encoding on Linux* OS since it is space efficient, can express nearly all languages, is supported on most systems and it is identical to 7bit ASCII encoding for codes 0 to 127.

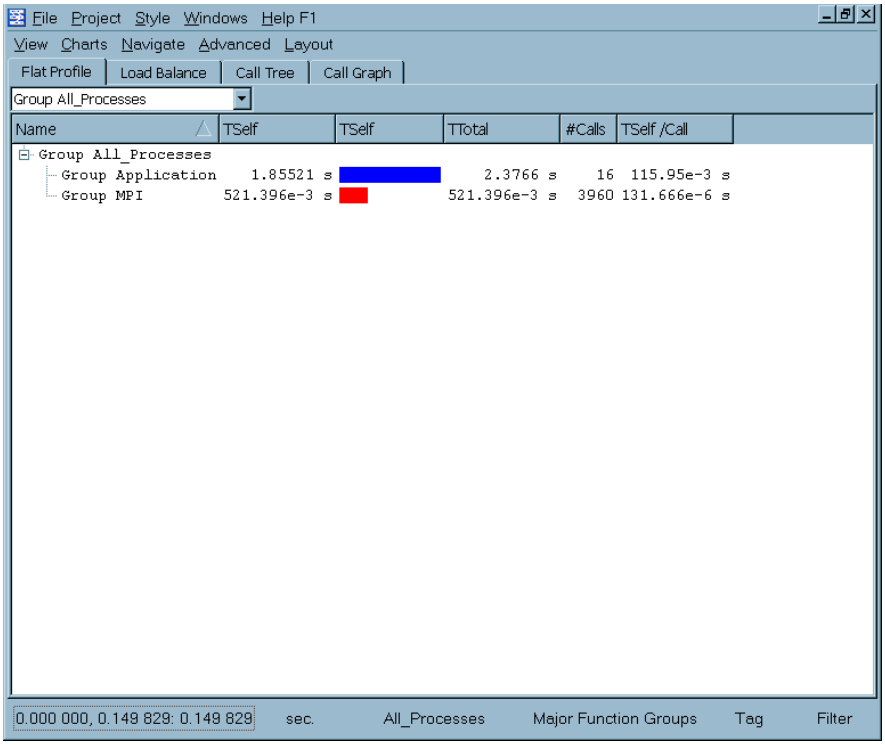
1.7 For the Impatient

This section provides a quick start into Intel® Trace Analyzer. It demonstrates essential features of the program using the example trace files `poisson_sendrecv.single.stf` and `poisson_icomm.single.stf` that are available in the Intel Trace Analyzer examples folder.

The traces were generated with two implementations of the same algorithm computing the same result over the same data set: a poisson solver for a linear equation system. As the names imply, the first version uses `sendrecv` to communicate, while the second version uses non-blocking communication.

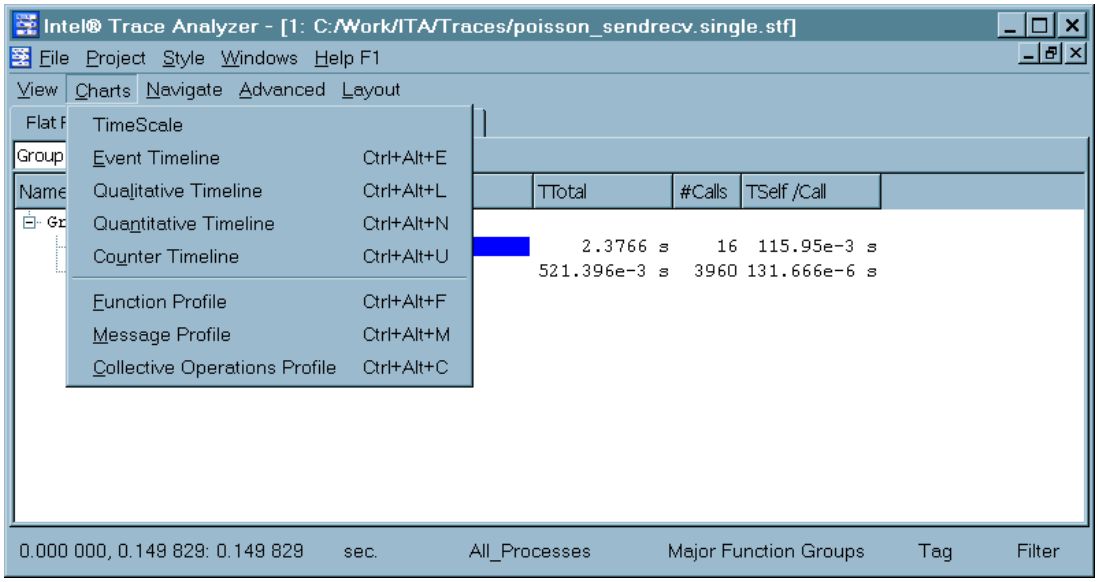
It is illustrated that the first version leads to an overall serialization of the parallel algorithm and how the improved version solves the problem. *Figure* 1.2 shows Intel Trace Analyzer after loading `poisson_sendrecv.single.stf`. The figure shows the main window and a child window, a so called View.

Figure 1.2 poisson_sendrecv.single.stf Loaded



Maximize the View with the respective button in its title bar and open an Event Timeline (**View Menu > Charts > Event Timeline**) as shown in *Figure* 1.3. The result should look like in *Figure* 1.4.

Figure 1.3 Opening an Event Timeline



Together with the Event Timeline (see Event Timeline), a time scale opens above the Timeline in the View. *Figure* 1.4 shows a View containing a time scale, an Event Timeline and a Function Profile. These diagrams are called Charts (see Charts). In the status bar (see Status Bar) found at the bottom of the View the current time interval and some other information are shown.

Zoom into the interesting area on the right edge of the Event Timeline by dragging the mouse over the desired time interval with the left mouse button pressed as shown in *Figure* 1.5. The result should look like in *Figure* 1.6. Note the apparent iterative nature of the application.

Figure 1.4 Event Timeline Opened

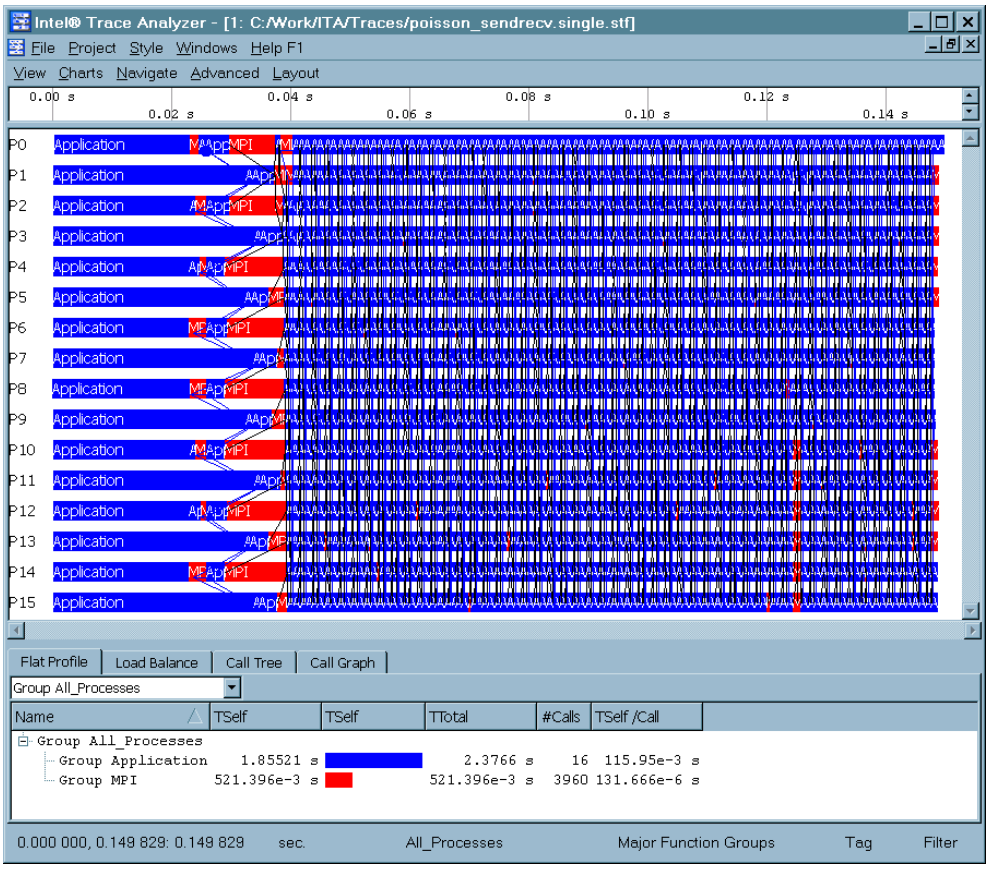


Figure 1.5 Zooming into a Chart

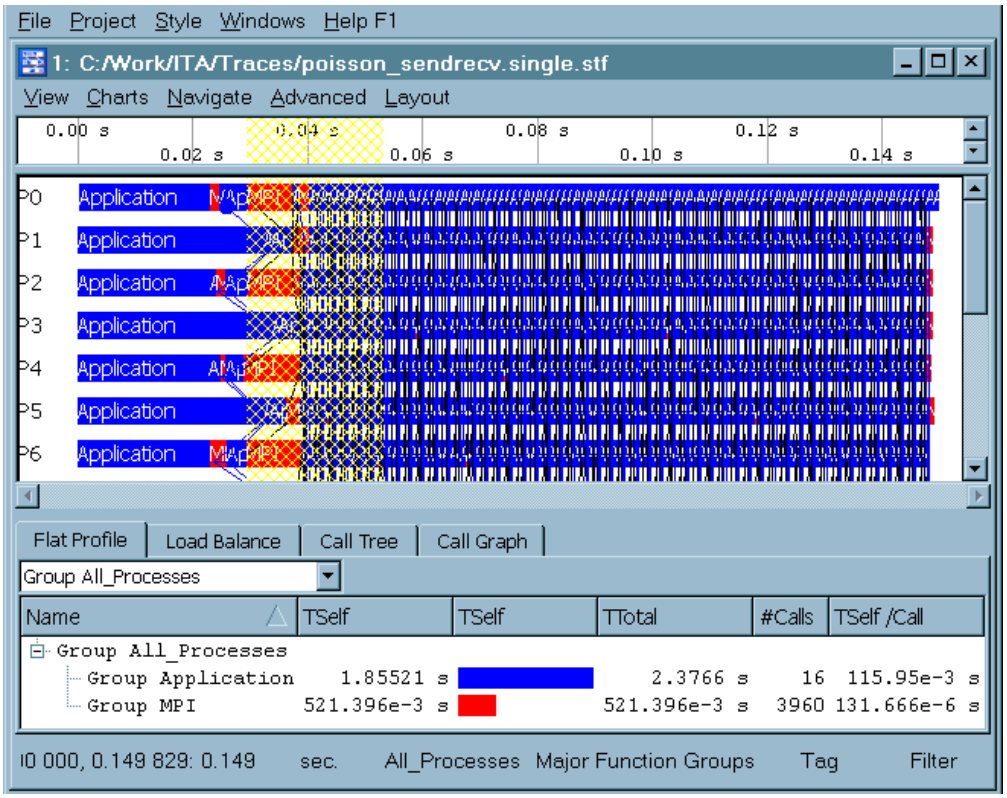
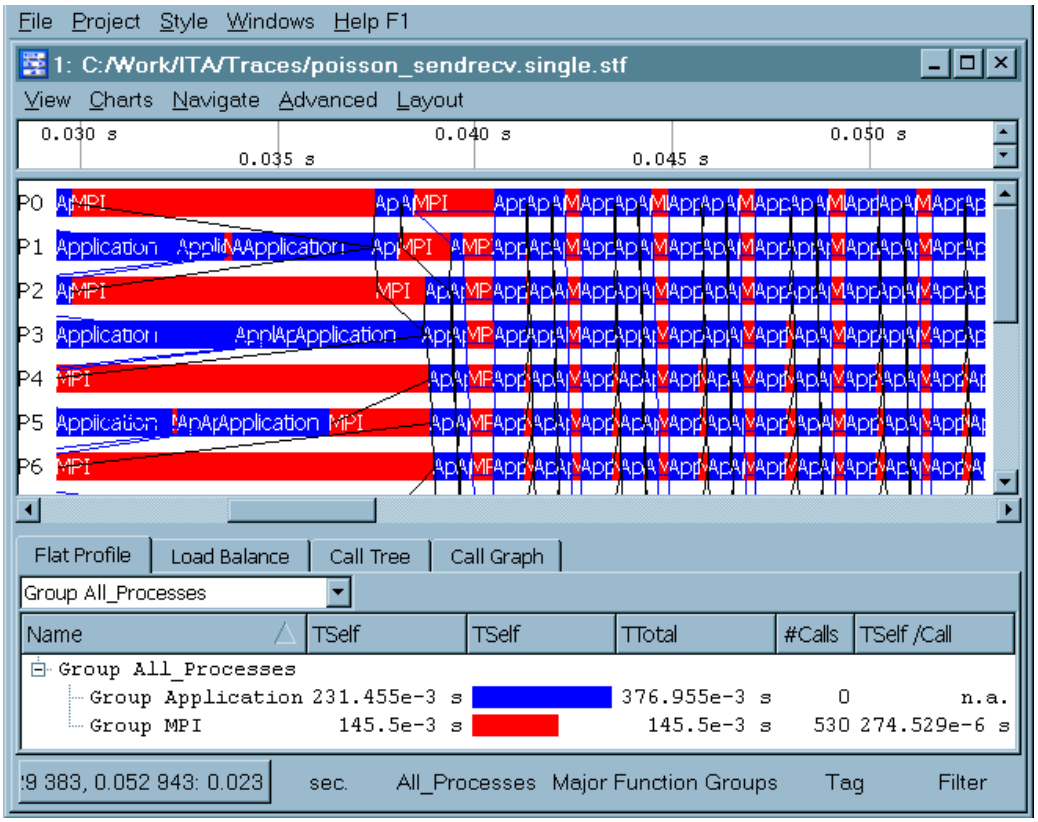


Figure 1.6 Zoomed Result



Now zoom further into the trace to look at a single iteration and close the Function Profile (**Context Menu > Close Chart**). The result should look like in *Figure* 1.7.

Figure 1.7 Zoomed to One Iteration

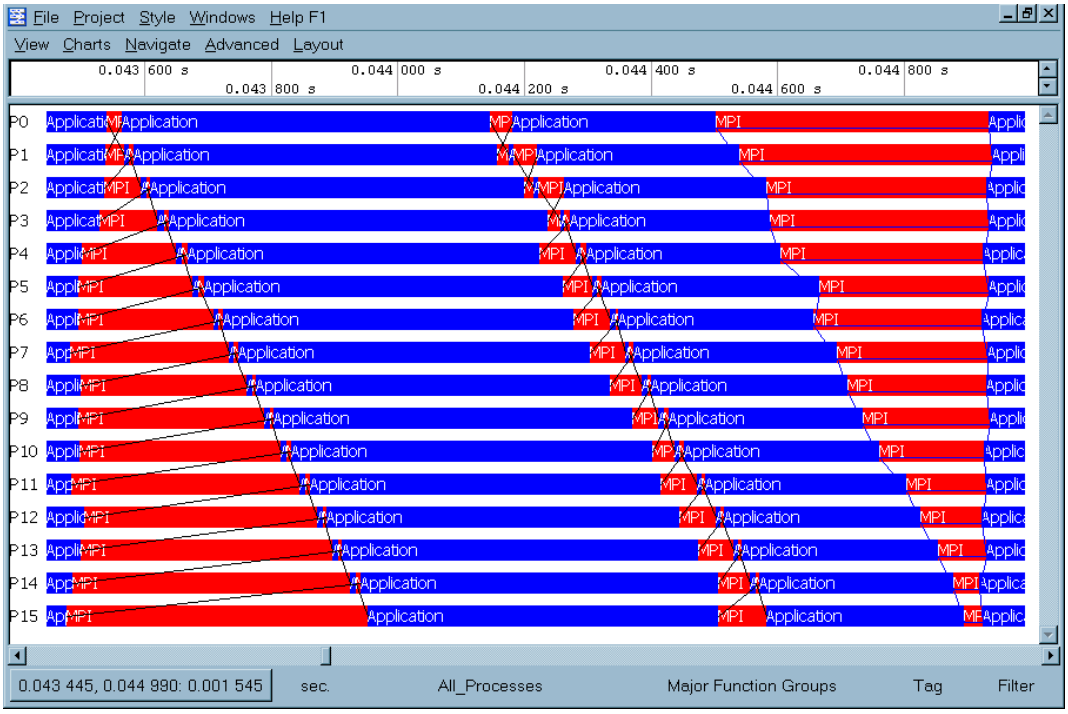
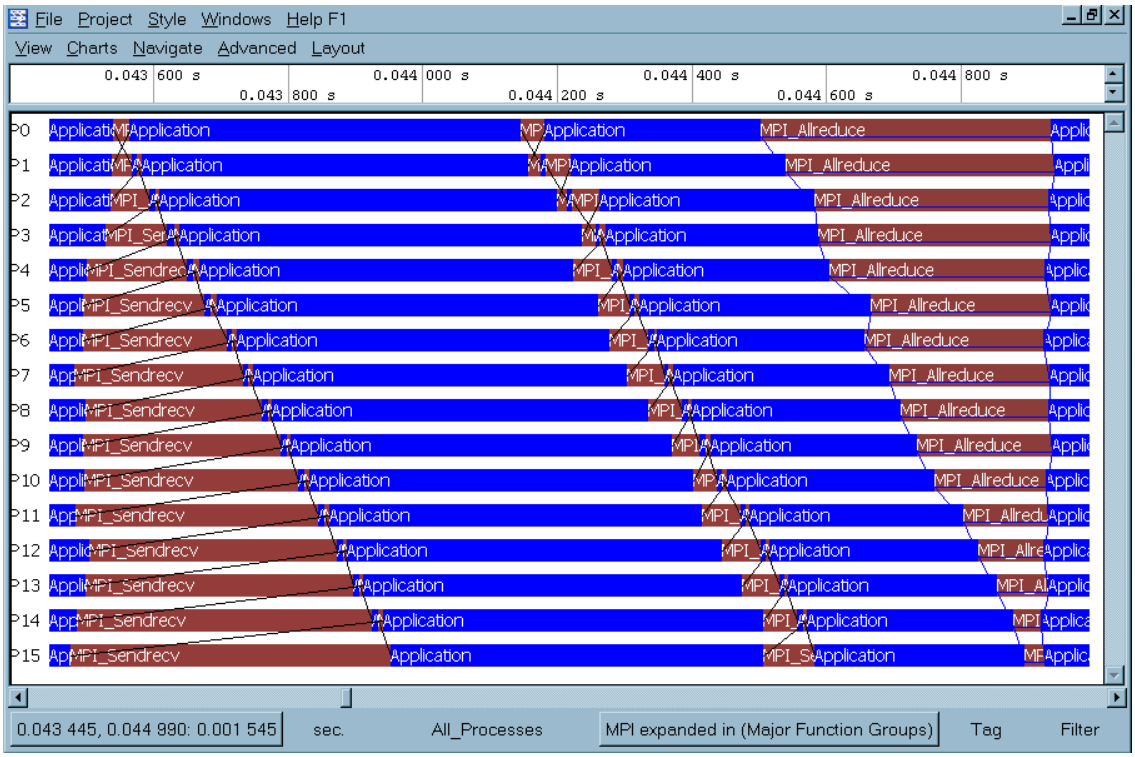


Figure 1.8 MPI Ungrouped



To see which particular MPI functions are used in the program, right-click on MPI in the Event Timeline and choose Ungroup Group MPI. The result should look like *Figure* 1.8. The Function Aggregation of the View changes so that the MPI functions are no longer aggregated (see Aggregation) into the Function Group MPI, but are shown individually. This is shown in the status bar. The button titled **Major Function Groups** changes to **MPI expanded in (Major Function Groups)**. Click this button to open the Function Group Editor (see Function Group Editor), which enables you to create new function groups and to switch between them.

It is apparent that at the start of the iteration the processes communicate with their direct neighbors using `MPI_Sendrecv`. The way this data exchange is implemented shows a clear disadvantage: process i does not exchange data with its neighbor i+1 until the exchange between i-1 and i is complete. This makes the first `MPI_Sendrecv` block look like a staircase. The second block is already deferred and hence does not show the same effect. The `MPI_Allreduce` at the end of the iteration nearly resynchronizes all processes. The net effect is that the processes spend most of their time waiting for each other.

Looking at the status bar (see *Figure* 1.8) shows that one iteration is roughly 1.5 milliseconds long.

Figure 1.9 shows a View with an Event Timeline (**View Menu > Charts > Event Timeline**), two Function Profiles (**View Menu > Charts > Function Profile**) with their Load Balance tab and a Message profile (**View Menu > Charts > Message Profile**) that reveals the asymmetric pattern in the point to point messages.

The time spent in `MPI_Sendrecv` grows with the process number while the time for `MPI_Allreduce` decreases. The Message Profile (section Message Profile) in the bottom right corner of *Figure* 1.9 shows that messages traveling from a higher rank to a lower rank need more and more time with increasing rank while the messages traveling from lower rank to higher rank do reveal a weak even-odd kind of pattern.

Figure 1.9 Many Charts

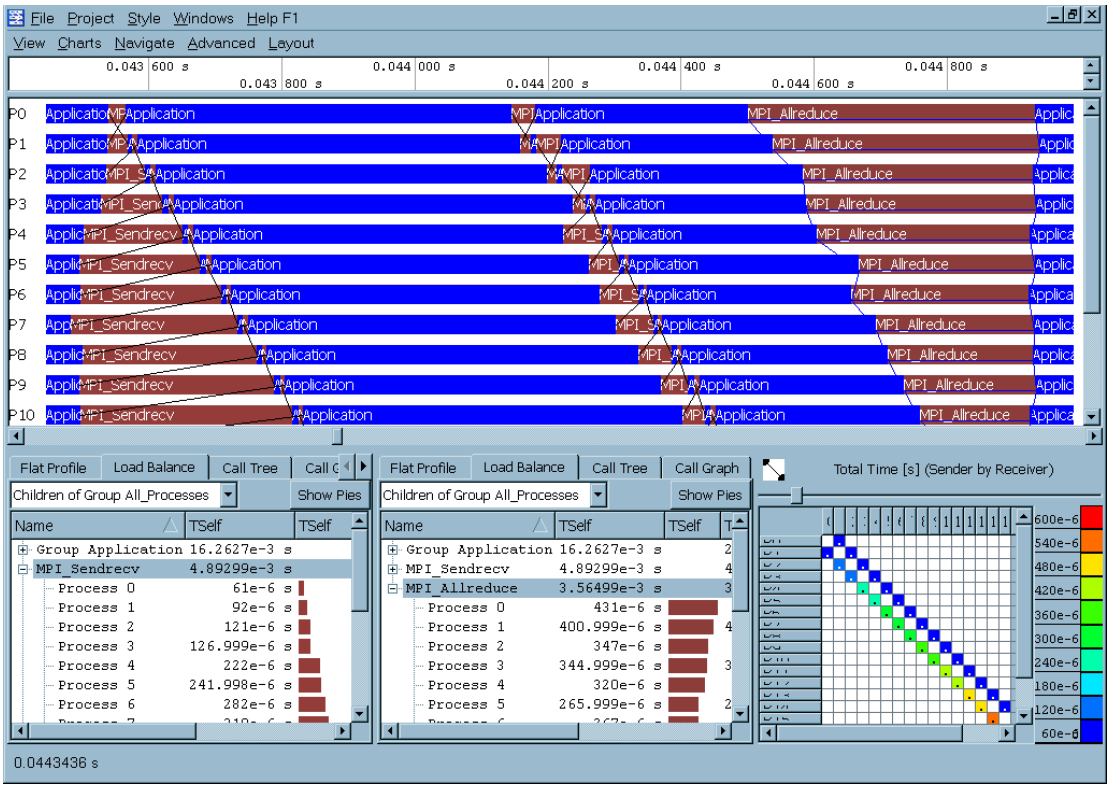
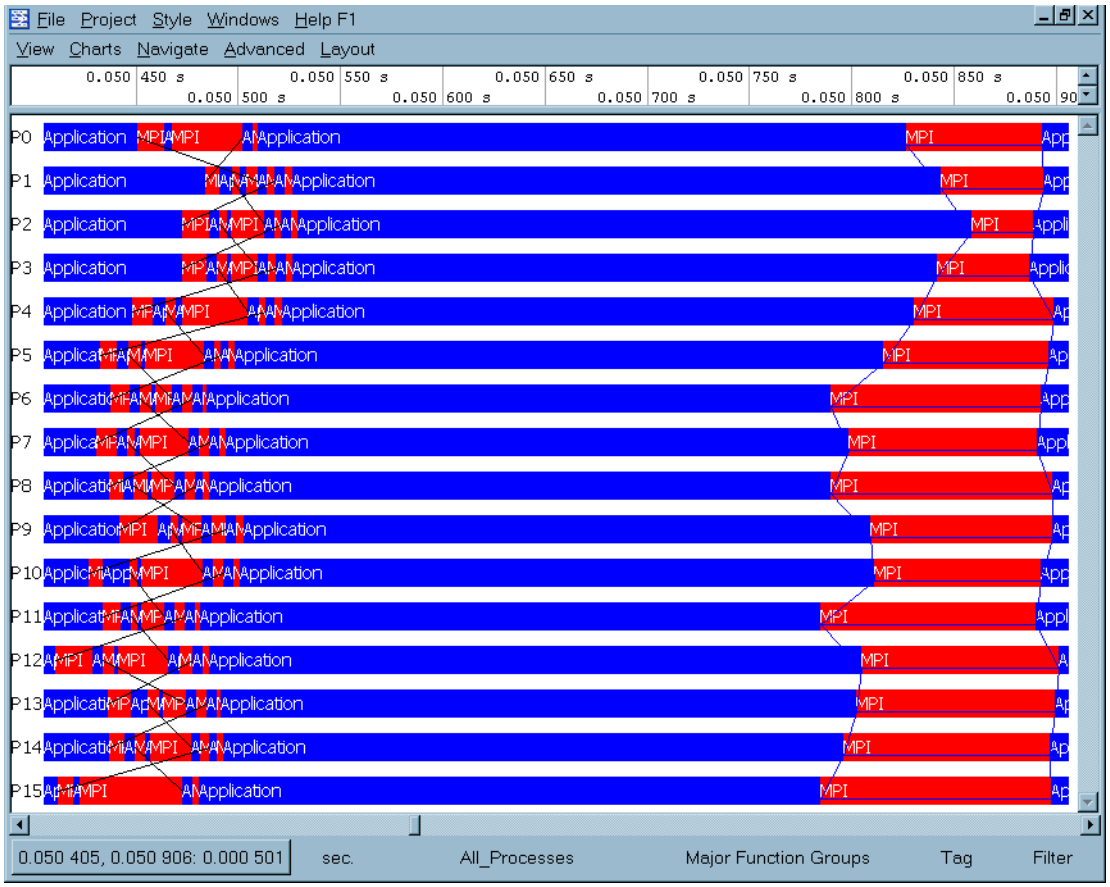


Figure 1.10 One Iteration of the Improved Version



As `poisson_sendrecv.single.stf` is such a striking example of serialization, almost all Charts provided by Intel® Trace Analyzer reveal this interesting pattern. But in real-world cases it might be necessary to formulate a hypothesis regarding how the program should behave and to check this hypothesis using the most adequate Chart.

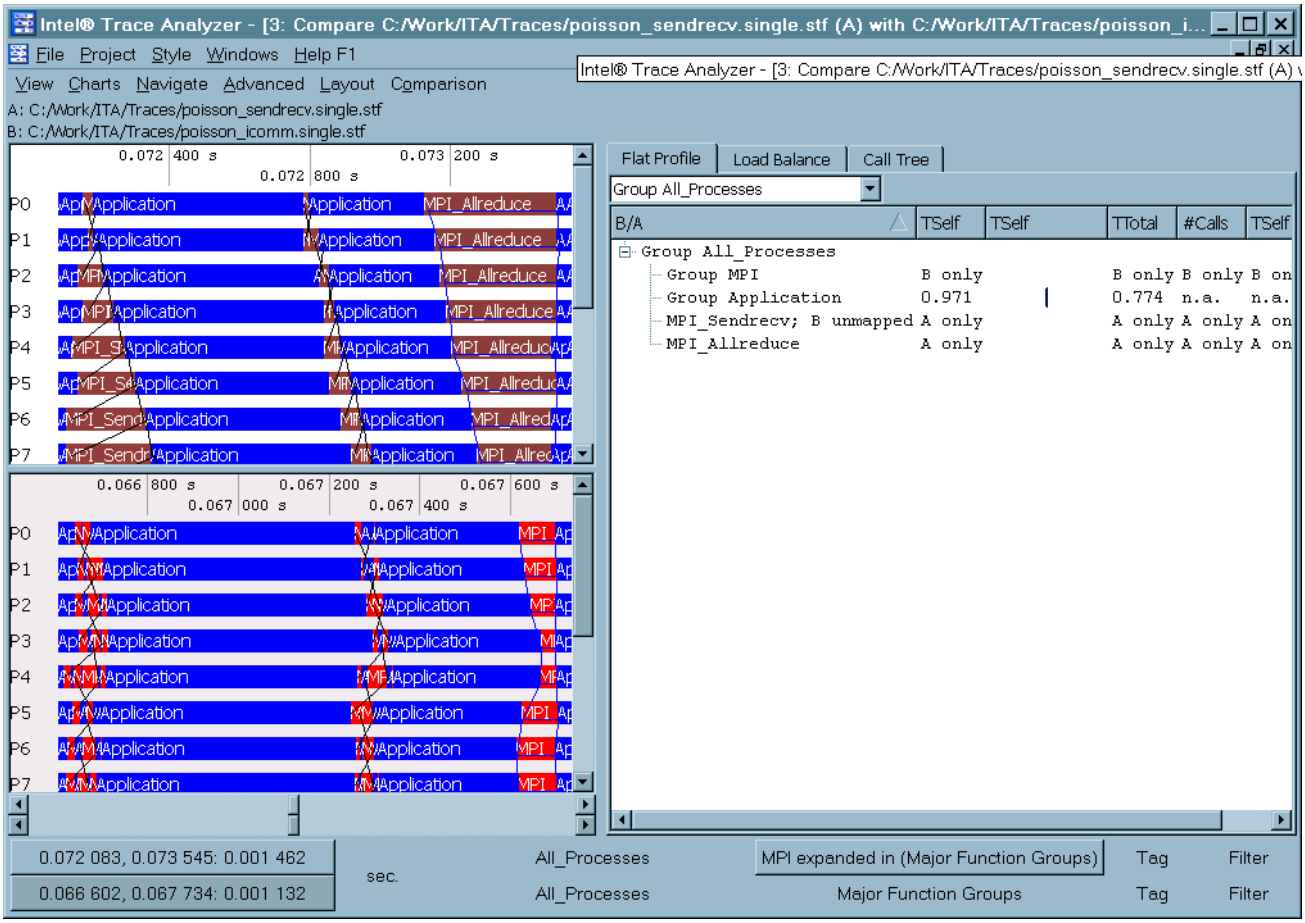
A possible way to improve the performance of the program is to use non-blocking communication to replace the usage of `MPI_Sendrecv` and to avoid the serialization in this way. One iteration of the resulting program looks like the one shown in *Figure* 1.10.

NOTE: A single iteration now takes about 0.9 milliseconds, while it took about 1.4 milliseconds before the change.

To compare two trace files, Intel Trace Analyzer offers the so-called Comparison View (see Comparison of two Trace Files). Just use the menu entry **View Menu > View > Compare** in the View showing `poisson_sendrecv.single.stf`. In the now appearing dialog, choose another View that shows `poisson_icomm.single.stf`. A new Comparison View is opened that shows an Event Timeline for each file and a Comparison Function Profile Chart (see Comparison Function Profile) that shows a profile computed from both trace files. The time intervals, aggregation settings and filters are taken from the original Views.

After adjusting some configurable behavior (see Comparison of two Trace Files) and zooming to the first iteration in each trace file, a Comparison View for these two files can look like in *Figure* 1.11. It is immediately obvious that one iteration in the improved program needs considerable less time than in the original program.

Figure 1.11 Comparing the First Iterations Taken from Two Program Runs



You can create a Comparison View that compares one time interval and one process group against another time interval and another process group of the same trace file. Other useful applications of the Comparison View include scalability analysis where you compare two runs of the same unmodified program with different processor counts and try to find out which functions scale well and which suffer from Amdahl's Law. Other scenarios could be the comparison of two different MPI libraries interconnects or machines.

Of course this introduction only scratches the surface. If there is not enough time to browse through the whole documentation, it is recommended to at least read through Concepts to learn about features like filtering, tagging, process aggregation and function aggregation. These features have the potential to make analyzing parallel applications more efficient.

2 Main Menu

The main menu of Intel® Trace Analyzer contains options using which the general settings of Intel Trace Analyzer can be changed. Use these entries to edit the configuration settings and the display style.

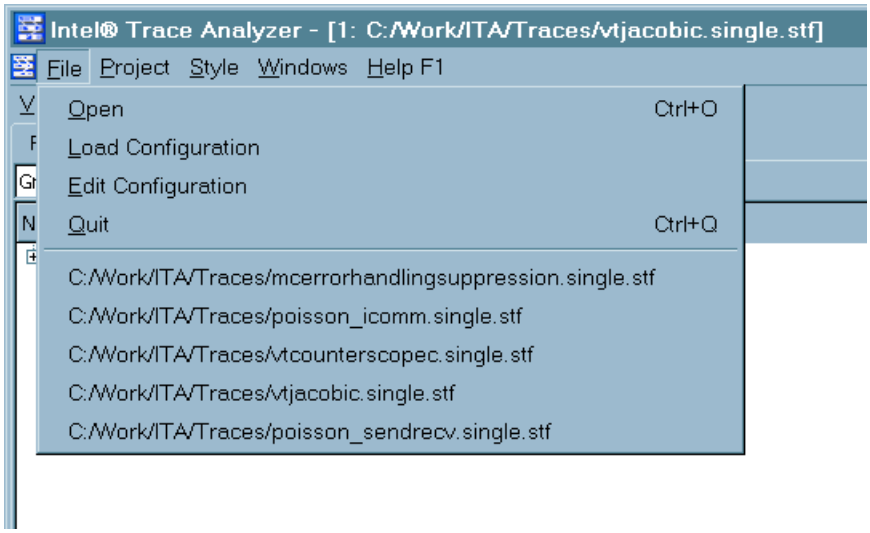
There are five sub-menus in the main menu. These are:

1. File Menu
2. Project Menu
3. Style Menu
4. Windows Menu
5. Help Menu

2.1 File Menu

The **File** menu has four entries. The **Open** menu entry opens a dialog. Specify one or several trace files to open in this dialog.

Figure 2.1 File Menu



To select a new configuration, use the **Load Configuration** entry. This entry opens a dialog where the required configuration is selected.

The option **Edit Configuration** opens the **Configuration** dialog that is explained in Edit Configuration Dialog.

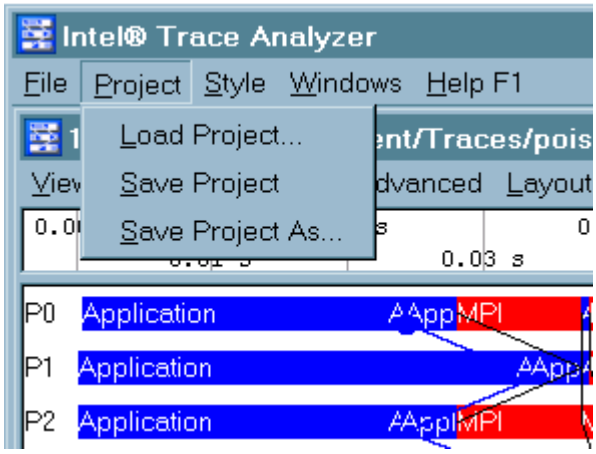
The **Quit** entry exits Intel® Trace Analyzer. The list below the **Quit** entry displays the 10 trace files opened most recently.

2.2 Project Menu

The **Project** menu enables you to save your working environment. Intel® Trace Analyzer and Collector does not save your environment automatically. You need to save the environment manually.

For the current project files, you can only store views of opened traces and corresponding charts.

Figure 2.2 Project Menu



You can perform the following actions with the Project menu:

- To open a previously saved project file, select **Load Project…** and browse your previously saved file.
- To save a project using the default naming, select **Save Project**.
- To save a project to a specific folder or project name, select **Save Project As…**.

2.3 Style Menu

The **Style** menu provides the option of choosing the design in which the view is displayed. The options in the **Style** menu depend on the environment. The first entry always shows the default style for the respective platform (or for the respective window manager when using X Windows). For example, in a Windows* environment, you have the option of choosing between the **Windows** option and the **WindowsXP** option. In a KDE* environment, you have as options **CDE**, **Cleanworks**, **Motif**, **Plastique** and **Windows** styles.

Additional styles may be available when provided by a desktop environment like KDE. If you experience problems with those styles, switch back to the default style. There is no support in case of problems that cannot be reproduced using the default style.

Figure 2.3 Style Menu on Linux* OS

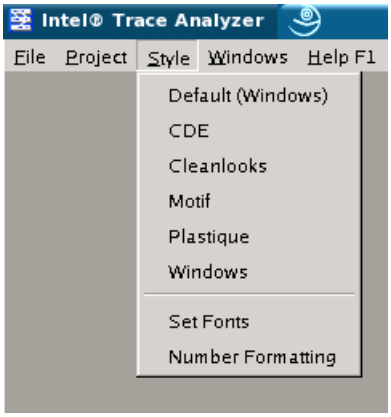
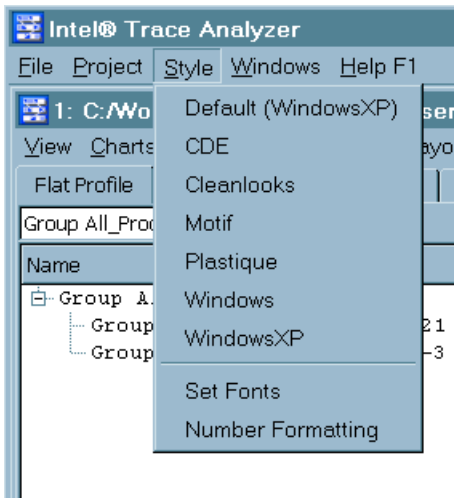


Figure 2.4 Style Menu on Windows* OS



2.3.1 Set Fonts

Use the **Set Fonts** option to change the fonts in Intel® Trace Analyzer. A dialog opens where the fonts can be changed for the individual Charts. For more information on the **Font** dialog box refer to Font Settings.

2.3.2 Number Formatting

This option lets you specify the format of the various numerical values displayed in Intel® Trace Analyzer. For more information on Number Formatting, refer to Number Formatting Settings.

2.4 Windows Menu

Use the menu options in the **Windows** menu to arrange the open sub-windows as required. There are three possibilities which are explained below. The **Windows** sub-menu also shows the name and path of the trace file that is presently opened.

- **Cascade**

Select the **Cascade** option to arrange the open sub-windows one behind the other.

- **Tile**

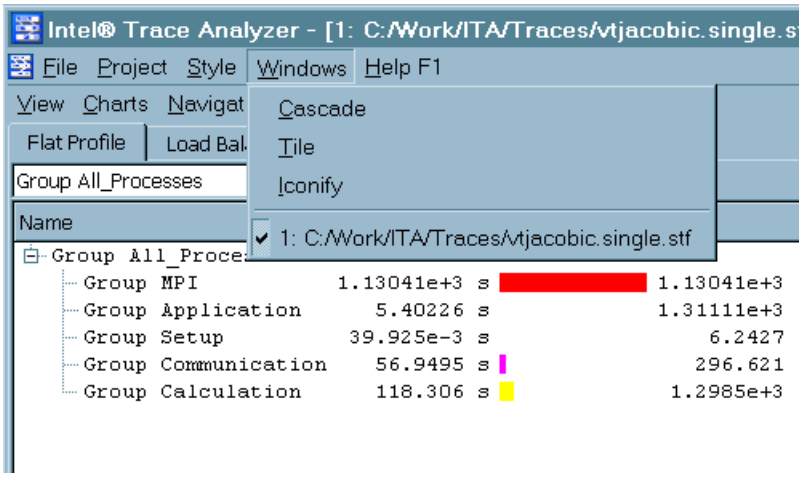
The **Tile** option arranges the sub-windows next to each other.

NOTE: This option is useful only when charts are opened in two or more sub-windows.

- **Iconify**

The **Iconify** option minimizes the open sub-windows and shows icons of them in the bottom of the View.

Figure 2.5 Windows Menu



2.5 Help Menu

The **Help** option allows you to access the HTML formatted online help through an integrated HTML browser. The first entry points to the table of contents while the next two point to a section about online resources.

The help is also installed in PDF format to ease printing.

The last entry gives you details regarding Intel® Trace Analyzer in use.

Throughout the application context sensitive help for the current Chart, dialog box or context menu is available by pressing F1.

3 Views

For a flexible analysis of a trace file it is helpful to look at multiple partitions of the data from various perspectives. The concept of Views is introduced here. A View holds a collection of Charts in a single window. These Charts, inherent in the same View, use the same perspective on the data. This perspective is made up of the following attributes: the time interval, process aggregation, function aggregation and filters. For an in-depth explanation of aggregation and filters, refer to Concepts.

Whenever an attribute in the current perspective is changed for one of the Charts, all other Charts follow. Opening several Views offers a very flexible and variable mechanism for exploring, analyzing and comparing trace data. The View status bar shows the current perspective on the data and allows to quickly change the perspective. More about the status bar is described in Status Bar.

3.1 View Main Menu

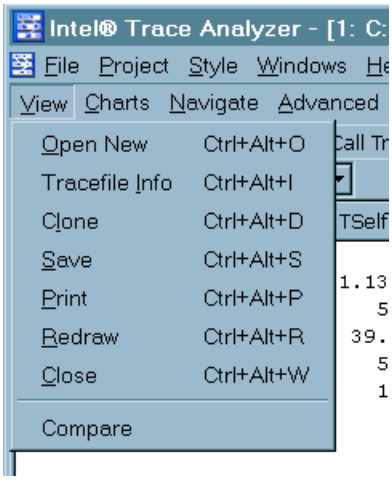
All operations that can be performed on a View are found on the View menu bar. This menu bar consists of six menu options:

- View Menu
- Charts Menu
- Navigate Menu
- Advanced Menu
- Layout Menu
- Comparison Menu

3.1.1 View Menu

The **View** menu consists of options that are carried out on the entire View. These options are described below:

Figure 3.1 View Menu



- **Open New** (Ctrl+Alt+O)

This opens the New View dialog box, which opens a new View with the selected Charts. The **New View** dialog box is explained in New View Dialog.

- **Tracefile Info** (Ctrl+Alt+I)

This opens the **Tracefile Info** dialog box, which contains the full information about the current trace.

- **Clone** (Ctrl+Alt+D)

This command replicates the existing View. It creates a one-to-one clone of the current View.

- **Save** (Ctrl+Alt+S)

This command saves the entire View as a picture.

- **Print** (Ctrl+Alt+P)

This entry prints a copy of the View.

- **Redraw** (Ctrl+Alt+R)

This command repaints the entire View.

- **Close** (Ctrl+Alt+W)

This command closes the current View. If the last View for a trace file is closed, the trace file is closed as well.

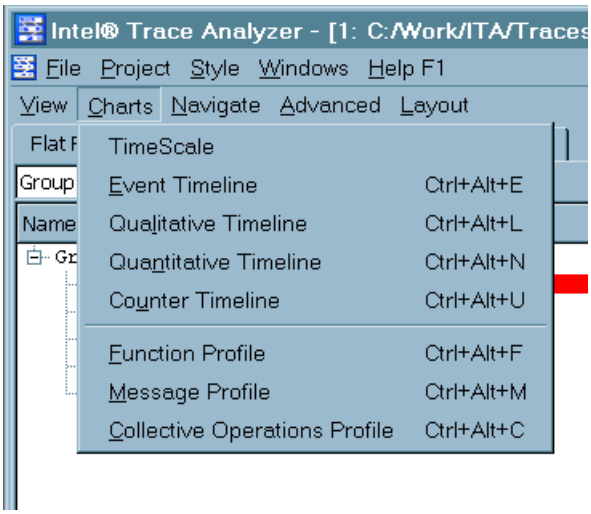
- **Compare**

This opens a dialog to select a file for comparison with the present one. This can be the current file of another trace file that is already open. After choosing the file, a new Comparison View will be opened that displays data from the two files. See Comparison of two Trace Files.

3.1.2 Charts Menu

The Charts menu enables you to open the various Charts of the Intel® Trace Analyzer. Click on a chart name commend to open its related view. The following are the available charts:

Figure 3.2 Charts Menu



TimeScale

This chart displays the Timescale of the project?. The Timescale chart is the default chart shown when you open any Timeline Chart.

Event Timeline (Ctrl+Alt+E)

This Chart that visualizes what the individual processes in the project are doing. For more information, see Event Timeline.

Qualitative Timeline (Ctrl+Alt+L)

This chart shows event attributes as they occur over time. For more information, see Qualitative Timeline.

Quantitative Timeline (Ctrl+Alt+N)

This chart gives an overview of the parallel behavior of the application. For more information, see Quantitative Timeline.

Counter Timeline (Ctrl+Alt+U)

The Counter Timeline chart shows the values of all counters that are present in the trace file. For more information, see Counter Timeline.

Function Profile (Ctrl+Alt+F)

This entry opens the Function Profile. On opening a trace file in Intel® Trace Analyzer, the Function Profile is shown by default. For more information on the Function Profile, see Function Profile Chart.

Message Profile (Ctrl+Alt+M)

The Message Profile provides statistical information on MPI point to point messages. Messages are categorized by grouping them in two dimensions. A detailed description of the Message Profile is available in Message Profile.

Collective Operations Profile (Ctrl+Alt+C)

The Collective Operations Profile provides statistical information on MPI's collective operations. Operations are categorized by grouping them in two dimensions. For more information on the Collective Operations Profile, refer to Collective Operations Profile.

3.1.3 Navigate Menu

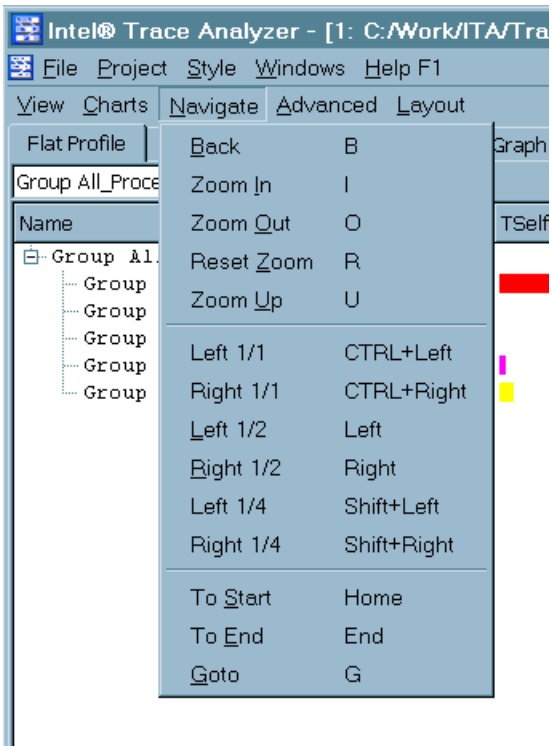
All functions provided by the Navigation Menu are also available through keyboard shortcuts. These keyboard shortcuts work only when a timeline has the keyboard focus. To give keyboard, focus to a timeline just left-click into it. The keyboard focus is indicated by a black rectangle around the timeline.

The entries in the Navigate menu are:

Back (B):

Pops the current interval from the zoom stack (see Zoom Stack).

Figure 3.3 Navigate Menu



Zoom Out (O):

Scales by a factor of 0.5 around the current center and pushes this new interval onto the zoom stack (see Zoom Stack).

Reset Zoom (R):

Clears the zoom stack and then pushes a time interval onto the zoom stack (see Zoom Stack) that corresponds to the complete trace file.

Zoom Up (U):

Maintains the current center and goes back to the previous zoom step.

Left 1 / 1 (Ctrl+Left):

Moves one whole screen to the left and pushes the new interval onto the zoom stack (see Zoom Stack).

Right 1 / 1 (Ctrl+Right):

Moves one whole screen to the right and pushes the new interval onto the zoom stack (see Zoom Stack).

Left 1 / 2 (Left):

Moves half a screen to the left and pushes the new interval onto the zoom stack (see Zoom Stack).

Right 1 / 2 (Right):

Moves half a screen to the right and pushes the new interval onto the zoom stack (see Zoom Stack).

Left 1 / 4 (Shift+Left):

Moves a quarter of a screen to the left and pushes the new interval onto the zoom stack (see Zoom Stack).

Right 1/4 (Shift+Right):

Moves a quarter of a screen to the right and pushes the new interval onto the zoom stack (see Zoom Stack).

To Start (Home):

Moves to the start of the trace file and pushes the new interval onto the zoom stack (see Zoom Stack).

To End (End):

Moves to the end of the trace file and pushes the new interval onto the zoom stack (see Zoom Stack).

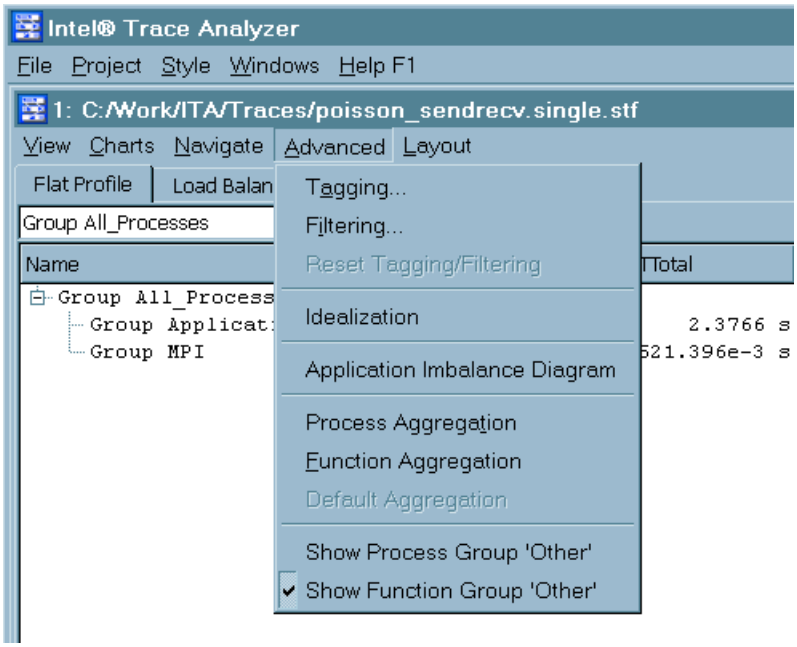
Goto (G):

Opens the **Time Interval Selection** dialog box (see Time Interval Selection).

3.1.4 Advanced Menu

The Advanced menu enables you to do the following advanced actions on your traces

Figure 3.4 Advanced Menu



Tagging Specific Events

You can tag, or highlight events that satisfy specific conditions. To set the conditions that are highlighted select **Advanced** > **Tagging** to open the **Tagging** dialog box.

To remove a previously set tag, select **Advanced** > **Reset Tagging/Filtering**. This command removes all tags and filters.

See Also

Tagging Dialog Box.

Filtering Visible Events

You can filter the events that are visible in the views. Filtering enables showing only those events that satisfy specific condition(s). All other events are suppressed as if they were not written to the trace file. To set the events that are filtered select **Advanced** > **Filtering**

To remove a previously set filter, select **Advanced** > **Reset Tagging/Filtering**. This command removes all tags and filters.

See Also

Filtering Dialog Box.

Simulating Ideal Communication

To view the results of a simulation of application behavior in the ideal communication environment based on the real trace generated by Intel® Trace Collector, select **Advanced** > **Idealization**. Use this feature to understand application imbalance and estimate a potential application speedup from tuning MPI implementation and/or network upgrades.

Idealization assumes the following conditions at trace processing time:

- zero network latency
- infinite network bandwidth
- zero buffer copy time
- infinite buffer size

Idealization assumes concurrency as limitation factors. For example,

- a message cannot be received before it is sent
- an All-To-All collective operation ends when the last thread starts

See Also

Trace Idealizer Dialog Box.

Checking Application Imbalance

Use the **Advnaced > Application Imbalance Diagram** option to compare the performance between a real trace and an ideal trace. This command enables you to choose the traces for Application Imbalance Diagram in the **Choose Idealized Trace** dialog box. Then specify the view in the **Application Imbalance Diagram** dialog box.

See Also

Application Imbalance Diagram Dialog.

Aggregating Results

Use the **Advanced** > **Process Aggregation** command to set the process groups to focus on and aggregates the results. Processes that are present in the trace file, but not in the process group are represented in a group Other. By default, the data for the Other group is not visible. To view data for the group Other, select **Advanced** > **Show Process Group 'Other'**.

To revert back to the default aggregation of All processes, select Advanced > Default Aggregation

See Also

Process Group Editor.

Aggregating Functions

Use the **Advanced** > **Function Aggregation** command to set the subset of functions focuses on and aggregate these into Function Groups. This command helps you to avoid being distracted by other functions that are currently not in focus. Functions that are present in the trace file, but not in the function group are represented in a group Other. By default, the data for the Other group is visible. To hide data for the group Other on the Quatative chart, select **Advanced** > **Show Function Group 'Other'**.

To revert back to the default aggregation of Major Function groups, select **Advanced** > **Default Aggregation**.

See Also

Function Group Editor

3.1.5 Layout Menu

Generally, every View is split into two sections: one for timelines and the other for Profiles. The Layout menu adjusts the position of the Charts or the respective sections on screen. For example, it provides the option of viewing the timelines and profiles next to each other or like the default, one below the other. The different entries of the layout menu, which are enabled when two or more charts are open, are explained below.

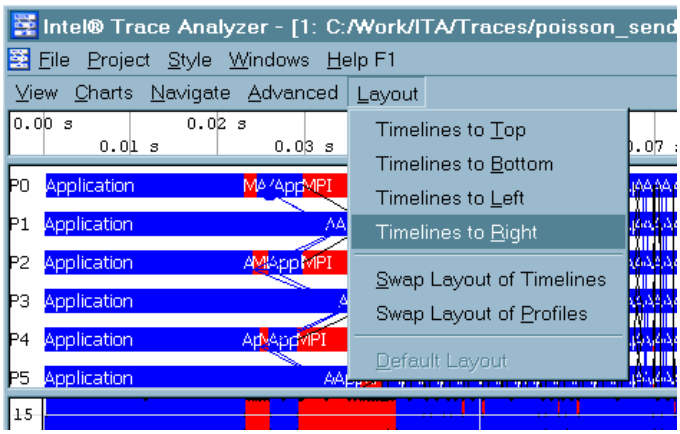
Timelines to Top

Use this entry to move the timelines to the upper portion of the View and the profiles to the bottom.

Timelines to Bottom

Use this entry to move the timelines to the lower portion of the View and the profiles to the top.

Figure 3.5 Layout Menu



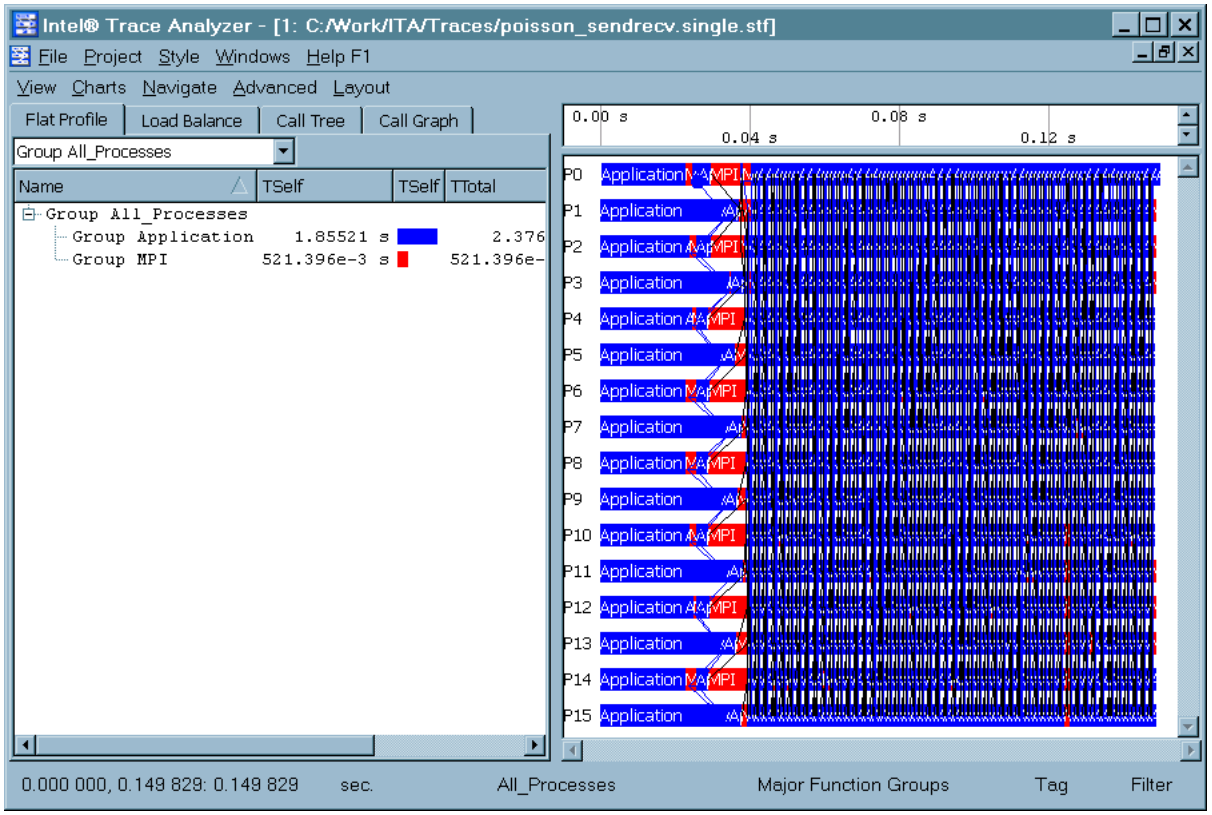
Timelines to Left

Use this entry to place the timelines to the left of the View. For example, to display the Charts next to each other, select the option **View Menu > Layout > Timelines to Left**. This moves the timeline(s) being shown in the View to the left of the screen, and thereby also shifts the profiles being shown to the other side. In case multiple timelines are open, these are collectively shown one below the other on the chosen side.

Timelines to Right

Use this entry to place all the timelines to the right of the View. It works the same way as Timelines to Left.

Figure 3.6 Placing Timelines to the Right



Swap Layout of Timelines

Swap the timelines. It is useful only when two or more timelines are open at the same time. If the timelines are stacked and aligned on top of each other, use this option to present them next to each other (or vice versa). The menu entry switches between the two choices.

Swap Layout of Profiles

Swaps the position of the profiles relative to each other, similar to that of the timelines as explained above.

Default Layout

This option brings the layout of the View back to the default settings. By default, timelines are stacked in the upper portion of the View and Profiles in the lower portion, side by side.

3.1.6 Comparison Menu

The Comparison menu is displayed in the Views menu only when the Comparison mode is activated (**View Menu > Compare**). This menu has four entries which are explained in Comparison of two Trace Files.

3.2 Status Bar

When moving the mouse cursor over an event in a timeline or an entry in a profile, expanded information on the respective entry is seen in the status bar at the bottom of the View for a few seconds. After this, or when the mouse moves into the status bar, it provides buttons that show the current settings and that enable changing the current settings. These shortcut buttons are described below:

Time Interval

The leftmost button in the status bar opens a dialog box that sets the time interval. More about the **Time Interval** dialog box is available in the section Time Interval Selection. The title on this button has the form *<start>, <end>: <duration>*. The status bar in *Figure* 3.6 shows that the start position of the time interval is 0.0 seconds, the end position is 0.149829 seconds and that the duration is 0.149829 seconds. The button appears flat when the full time range of the file is shown.

Sec. / Tick

This button shows the current unit of time and switches the unit from seconds to ticks and vice versa. The button appears flat if the unit in use is seconds and appears raised when using ticks.

Process Aggregation

This button shows the current process aggregation. By default, it is All_Processes. Click this button to open the **Process Group Editor** dialog box, which is explained in Process Group Editor. The button appears flat for the default process aggregation.

Function Aggregation

This button shows the current function aggregation. By default, it is Major Function Groups. Clicking this button opens the **Function Group Editor** dialog box, which is explained in the section Function Group Editor. The button appears flat for the default function aggregation.

Tagging and Filtering

The buttons labeled **Tag** and **Filter** open the **Tagging** dialog box (see Tagging Dialog) and the **Filtering** dialog box (see Filtering Dialog) respectively. These buttons appear flat if no tagging/filtering is selected.

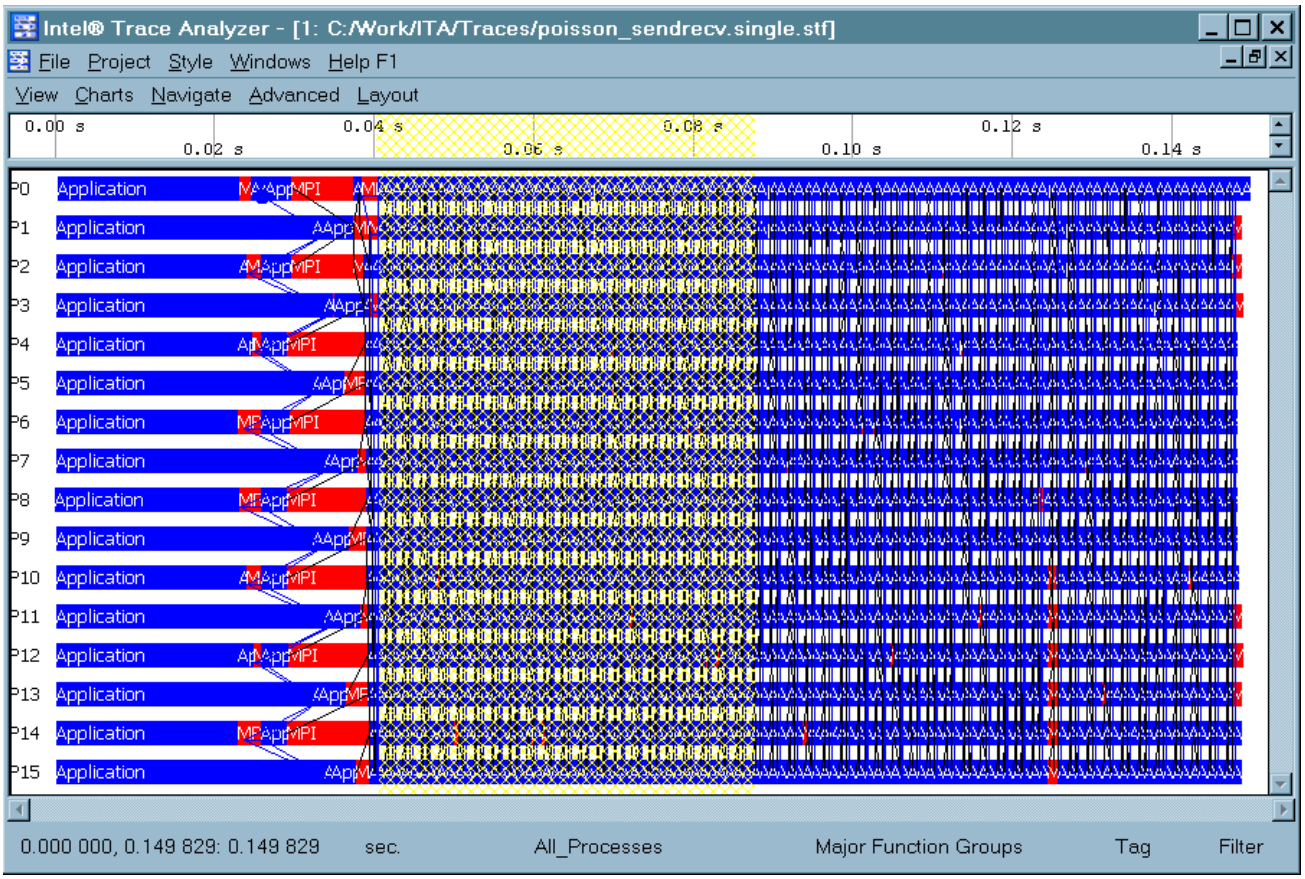
3.3 Navigation In Time

In addition to the menu entries and keystrokes mentioned in the section Navigate Menu, there are two more ways to move in time in Intel® Trace Analyzer. One is by using the scroll bar found below the timelines and another is by using the mouse to zoom into a time interval. Similar to the keystrokes, these operations manipulate the zoom stack. Refer to Zoom Stack for a detailed explanation.

To move the View by a quarter of a screen to a selected direction, click on the arrow buttons of the scroll bar. To move the View by one entire screen, click into the bar. Both actions push the new interval onto the zoom stack.

To use the mouse for zooming, left-click into the timeline (even clicking into the Timescale works) and select the region of the new time interval, keeping the left mouse button pressed. On releasing the left mouse button this new interval is pushed onto the zoom stack.

Figure 3.7 Zooming with Mouse



3.3.1 Zoom Stack

The zoom stack supports navigation in time by storing previously displayed time intervals. These time intervals are restored when required. Navigation using the keyboard or mouse is quite intuitive without detailed knowledge of the zoom stack. Nevertheless, an explanation is given below for the sake of complete reference.

Consider a trace file spanning the time from 0 to 100 seconds. When the first View is opened the zoom stack looks like this:

Figure 3.8 Zoom Stack

```
0:(0;100)(50;100)
```

Each stack entry has the syntax `(<start>;<stop>)(<center>;<width>)`. Zooming in using **Zoom In** (`I`), magnifies the Chart by a factor of 2 around the current center. In the above example, the center is at 50 seconds. Therefore, zooming in twice using I will result in a stack shown in *Figure* 3.9.

Figure 3.9 State of the Zoom Stack — Zoomed in Twice

```
3:(55;65)(60;10)
2:(50;70)(60;20)
1:(40;80)(60;40)
0:(0;100)(50;100)
```

Figure 3.9 shows that a stack level is created for each action performed. On pressing `Ctrl+Right` twice, the time frame of the Chart is moved to the right by two window sizes. Using the state shown in *Figure* 3.10, the differences between **Back** (`B`), **Zoom Out** (`O`) and **Zoom Up** (`U`) are explained below.

Figure 3.10 State of the Zoom Stack — Moved Two Window Sizes to the Right

```
5:(75;85)(80;10)
4:(65;75)(70;10)
3:(55;65)(60;10)
2:(50;70)(60;20)
1:(40;80)(60;40)
0:(0;100)(50;100)
```

Back (`B`): Pressing `B` in the state shown in *Figure* 3.10 pops out one level of the stack and the previous time frame is displayed as shown in *Figure* 3.11.

Figure 3.11 State of the Zoom Stack - after *Back (B)*

```
4:(65;75)(70;10)
3:(55;65)(60;10)
2:(50;70)(60;20)
1:(40;80)(60;40)
0:(0;100)(50;100)
```

Zoom Out (`O`): Using `O` in *Figure* 3.10 lowers the magnification (doubles the width) and pushes a new item on the stack. Zooming out using `O` in the state shown in *Figure* 3.10, results in the stack as shown below.

Figure 3.12 State of the Zoom Stack - after *Zoom Out (O)*

```
6:(70;90)(80;20)
5:(75;85)(80;10)
4:(65;75)(70;10)
3:(55;65)(60;10)
2:(50;70)(60;20)
1:(40;80)(60;40)
0:(0;100)(50;100)
```

Zoom Up (`U`): The resultant zoom after using U is a little more complicated. Pressing `U` keeps the current center and uses the magnification from before the latest zoom in (which is the change from level 2 to 3 in *Figure* 3.10) and to clear the stack up to and including the last zoom in found. In our example the current center is 80 and the width from stack level 2, which is 20, is used.

Figure 3.13 State of the Zoom Stack - after *Zoom Up (U)*

```
2:(70;90)(80;20)
1:(40;80)(60;40)
0:(0;100)(50;100)
```

Reset (R): Reset clears the stack and pushes one entry covering the full time range of the trace file. Therefore, on pressing R, the stack shown in *Figure* 3.8 is reestablished.

4 Charts

Charts in Intel® Trace Analyzer are graphical or alphanumerical diagrams that are parameterized with a time interval, a process grouping and a function grouping (see Aggregation) and an optional filter. Together they define the structure in which data is presented and the amount of data to be displayed.

The Charts supported by Intel Trace Analyzer are divided into:

1. Timelines: Event Timeline, Qualitative Timeline, Quantitative Timeline and Counter Timeline.
2. Profiles: Function Profile, Message Profile and Collective Operations Profile.

Charts are grouped into Views. These Views provide ways to choose the time interval, the process grouping and optional filters that all Charts in the View use. For more details on Views, see Views.

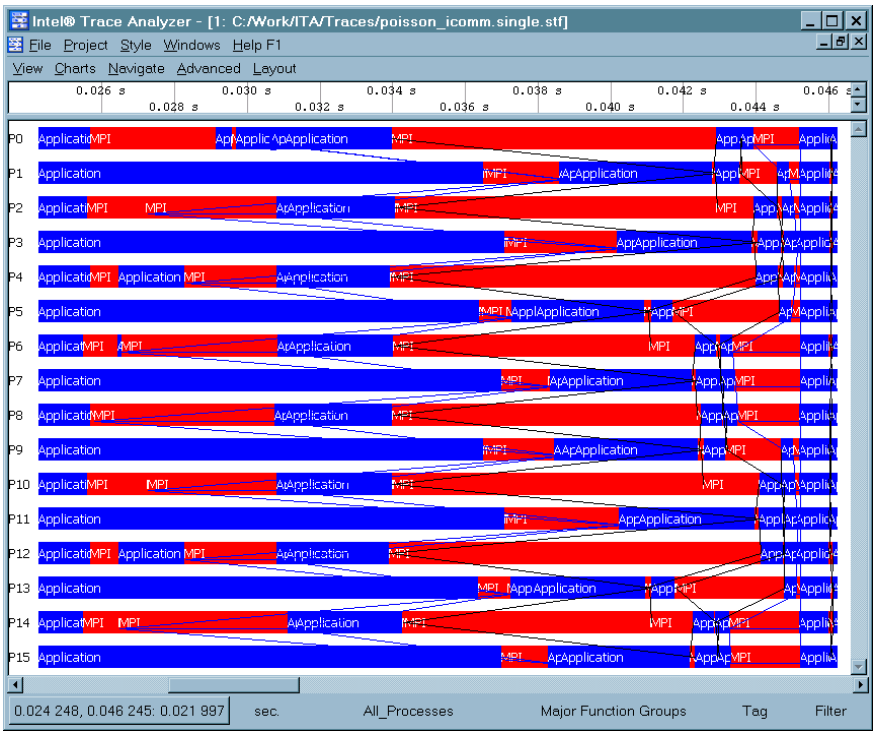
While the former show trace data in graphical form over a horizontal axis representing runtime, the latter show statistical data. All these Charts are found under **Views Menu > Charts**. Opening a file in Intel Trace Analyzer displays a View containing the Function Profile Chart (showing the **Flat Profile** tab) for the opened file.

The following sections describe each Chart in detail. For each Chart there is a subsection about Mouse Hovering, the context menu, the settings dialog box, the effects of filtering and tagging and the effects of aggregation, if it is applicable.

4.1 Event Timeline

The Event Timeline provides a graphical display of the individual process activities.

Figure 4.1 Event Timeline



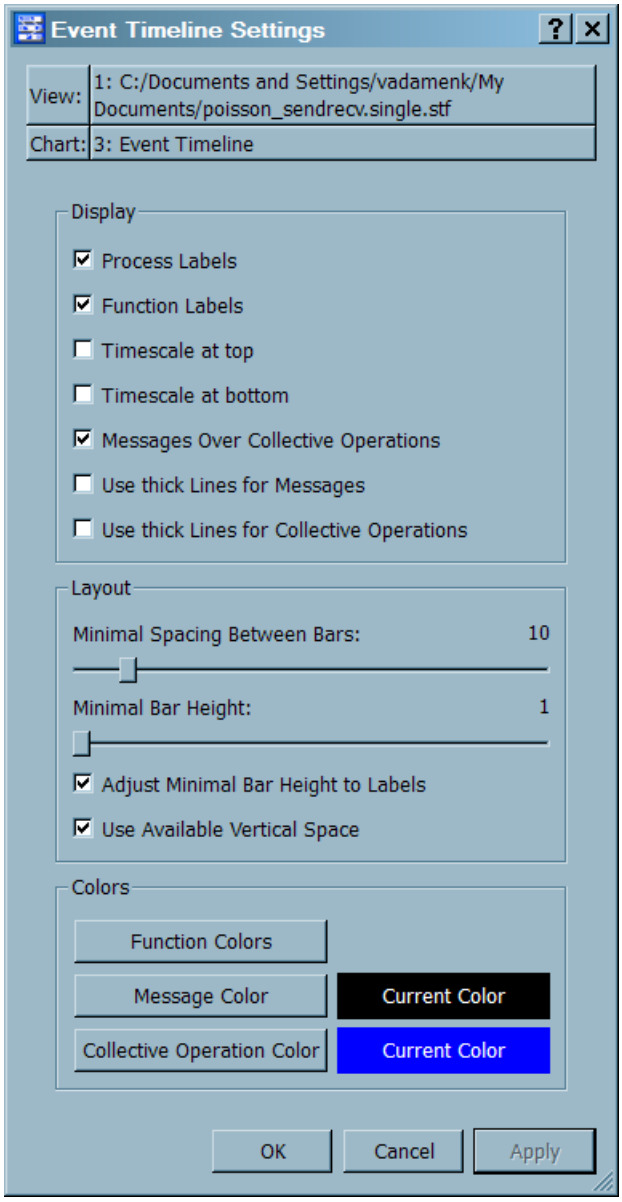
The horizontal axis represents an interval of the runtime of the inspected program. Vertical bars show the function in which the process is. The bars consist of rectangles labeled with the current function name and with color. Black lines indicate messages sent between processes; they connect sending and receiving processes. Blue lines, forming a grid, represent collective operations, such as broadcast or reduce operations. The status bar at the bottom of this panel shows information on events under the mouse cursor. By default, the entire runtime of the trace is visible. To get a better impression on the visualization primitives, use zooming. Zooming and navigation are explained in detail in the section Navigate Menu.

4.1.1 Mouse Hover

When the mouse pointer hovers over the Event Timeline, its exact position in terms of time is shown in the status bar. The status bar also shows which function, collective operations or messages are under the mouse cursor at the moment.

4.1.2 Event Timeline Settings

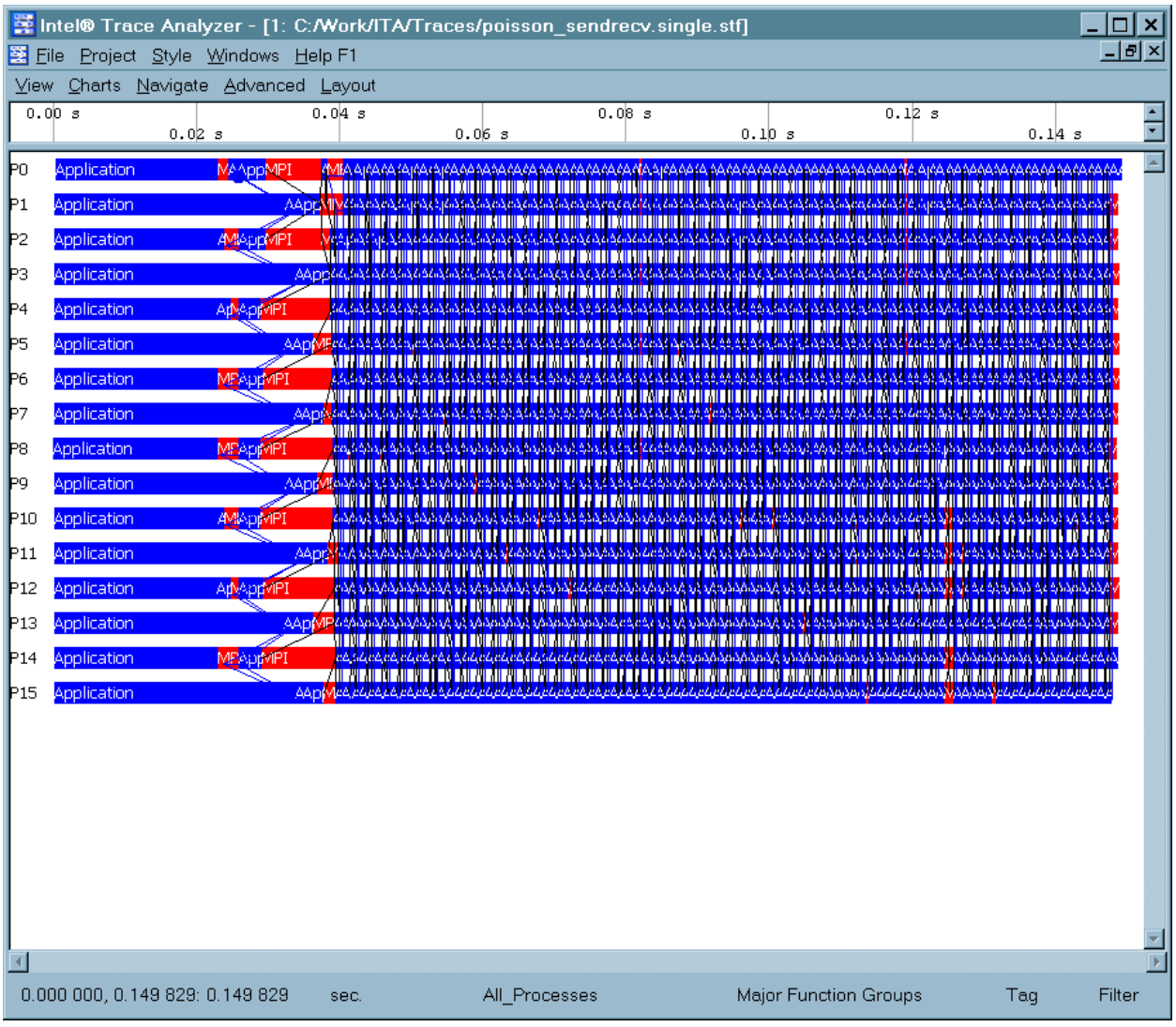
Figure 4.2 Settings Dialog Box for the Event Timeline



Use the **Event Timeline Settings** to change the display settings, the layout settings and the colors of the Event Timeline Chart. The display options, which are enabled using check boxes, include:

- **Process Labels**: Displays the name of the process or process group on the left.
- **Function Labels**: Displays the name of the function or function group inside the colored bars
- **Timescale at top**: Displays a timescale above the Chart
- **Timescale at bottom**: Displays a timescale below the Chart
- **Messages over Collective Operations**: This specifies what is drawn first. If enabled, messages are drawn over the collective operations so that the messages are seen and the collective operations are concealed. If disabled, the collective operations are drawn over the messages.
- **Use thick Lines for Messages**: Displays messages using thick lines
- **Use thick Lines for Collective Operations**: Displays collective operations using thick lines

Figure 4.3 Event Timeline: Use Available Vertical Space Unchecked

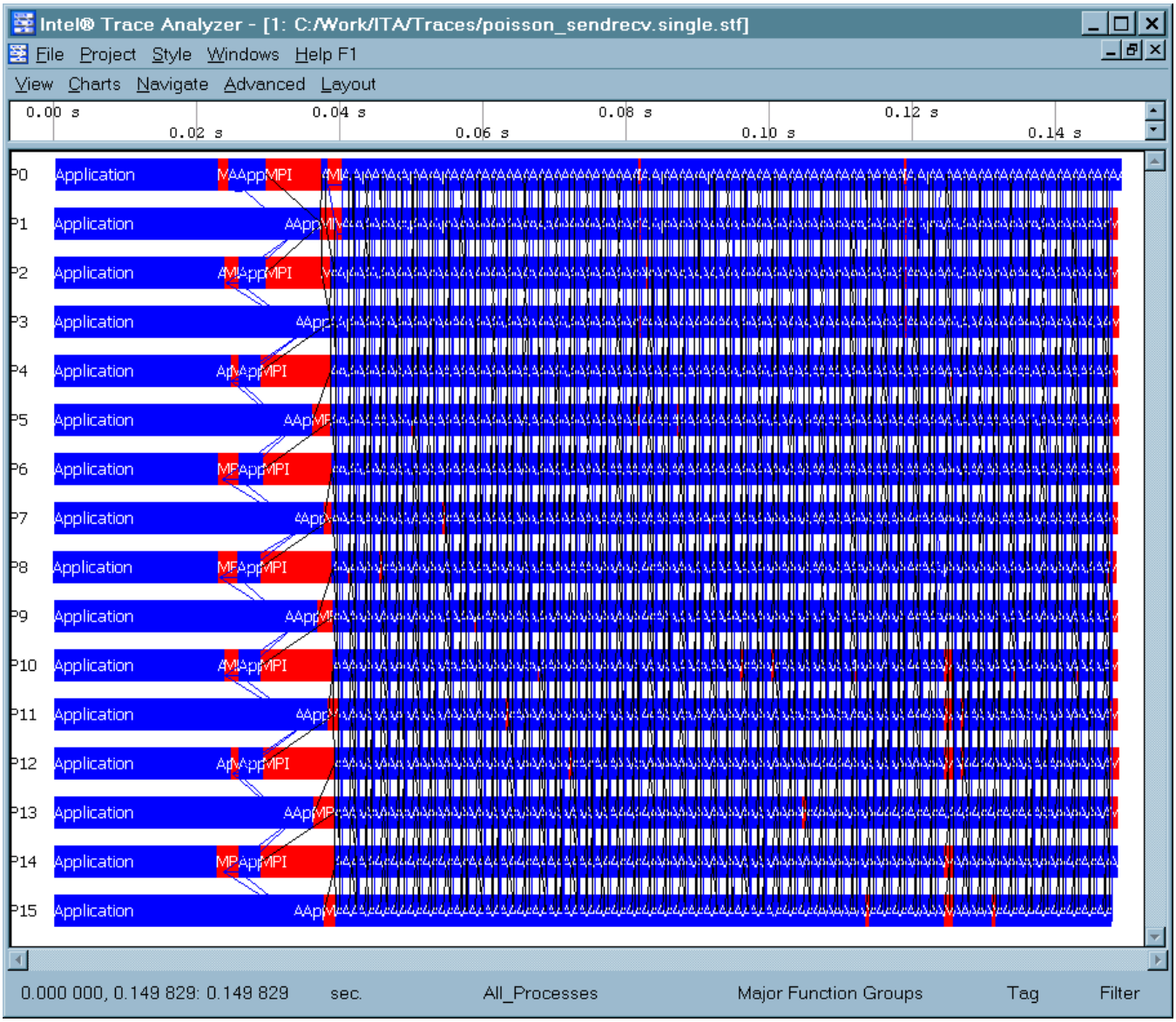


The **Layout** option consists of two sliders and two check boxes. The slider **Minimal Spacing Between Bars** adjusts the space between the function bars. The slider **Minimal Bar Height** adjusts the height of the function bar itself. For both, the unit is pixel.

The **Adjust Minimal Bar Height to Labels** check box alters the size of the bars. Check this box to make the bars tall enough to display a function label. This option is checked by default.

The check box **Use Available Vertical Space** influences the overall vertical layout of the bars and is also checked by default. For a better understanding, see *Figure* 4.3 where the Event Timeline is shown with this checkbox unchecked and *Figure* 4.4 where the **Use Available Vertical Space** is checked.

Figure 4.4 Event Timeline: Use Available Vertical Space Checked



To change the colors of the functions, messages or collective operations, use three **Colors** buttons at the bottom of the **Settings** dialog box. **Function Colors** call up the Function Group Color Editor (section Function Group Color Editor). To change the color in which the messages are displayed, click on the **Message Color** button. This opens a dialog box where the required color is chosen. Change the color of the Collective Operations in the same way using **Collective Operation Color** button.

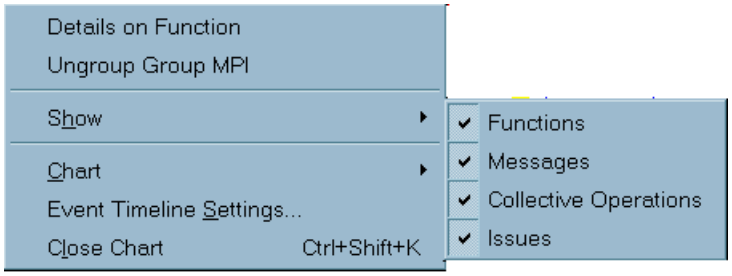
Colors chosen for messages and collectives are local to the Event Timeline Chart. Colors chosen for functions or function groups are shared by all Charts and Views belonging to the same trace file.

4.1.3 Context Menu

Apart from the general options common for all Charts (see Common Chart Features), the context menu of the Event Timeline provides details pertaining to the function, the message or the Collective Operation. For example, to access details about a particular message in the Event Timeline, right-click on the message and select the option Details on Message. This opens a new dialog box displaying information on sender, receiver and other message attributes. This is further explained in Details Dialog.

When invoked for a collective operation the context menu will offer the entry **Zoom to Collective** that will set the View time interval to the time covered by the respective collective operation.

Figure 4.5 Event Timeline: context menu example



Furthermore, the context menu has an entry to ungroup/regroup the function group, which works the same way in every Chart. To display functions, messages and/or collective operations, select **Show** from the context menu. Select/deselect one or more of the above from the sub-menu that opens on clicking **Show**.

The context menu entries that are common to all Charts are explained in the section Common Chart Features.

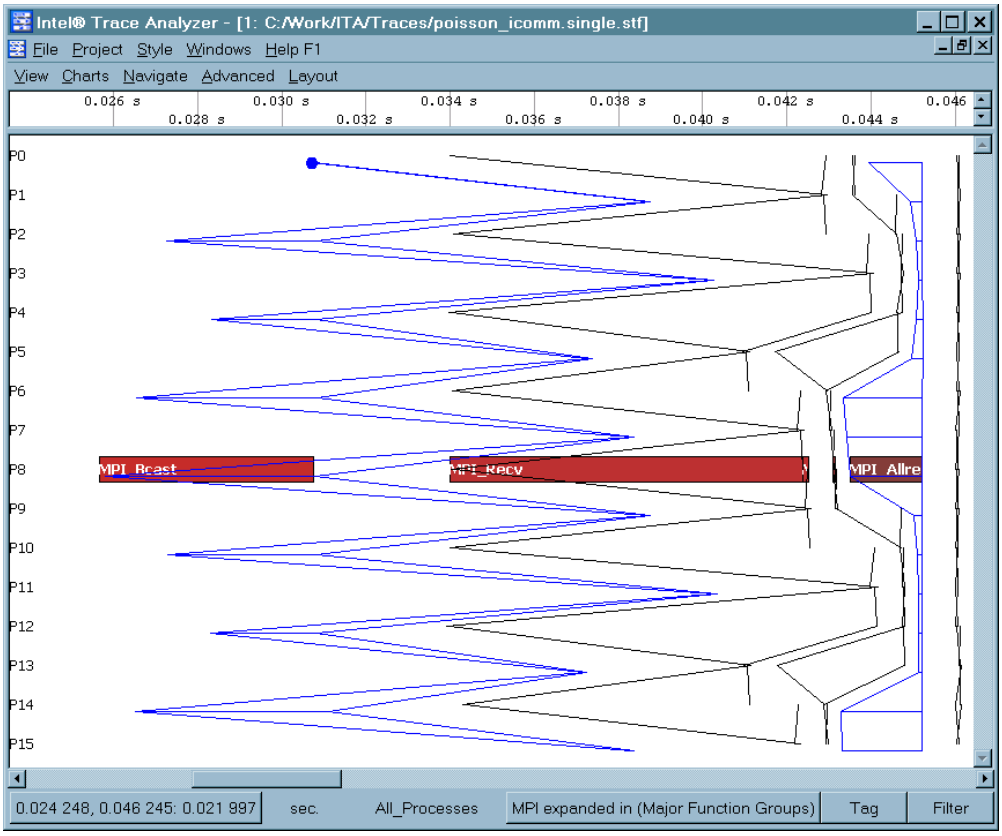
4.1.4 Filtering and Tagging

Tagged functions in the Event Timeline are shown with a frame around them and with the function label rendered in a bold font. Tagged messages and collectives are shown with thicker lines.

Figure 4.6 shows an Event Timeline where the MPI function in process number 8 (P8) is tagged. On filtering the Event Timeline with the same clause (MPI in P8), only the MPI functions of P8 pass the filter and are displayed, while all other functions and processes are filtered out as shown in *Figure* 4.7. More on tagging and filtering is available in the section Tagging and Filtering.

Figure 4.6 Tagging Functions in a Process in the Event Timeline

Figure 4.7 Filtering Functions in a Process in the Event Timeline



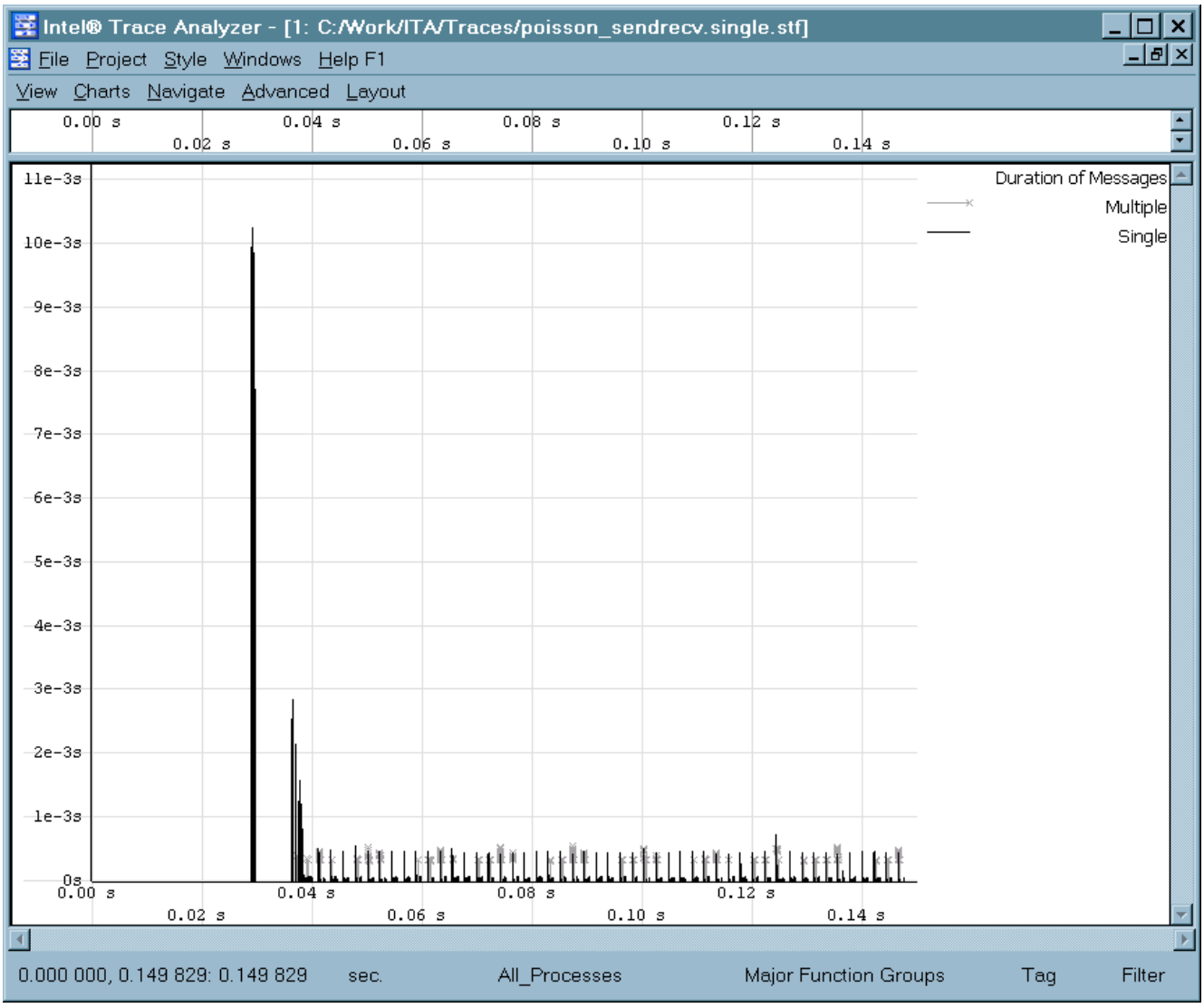
4.2 Qualitative Timeline

The Qualitative Timeline (**Views Menu > Charts > Qualitative Timeline**) shows event attributes like the data volume of messages as they occur over time. The value of this attribute is plotted along the y-axis while time is plotted along the x-axis. Select the required event type and attributes from the context menu. It helps detect patterns and irregular behavior such as extreme deviations or long-term changes in attribute values.

For the Poisson example, *Figure* 4.8 shows that the Qualitative Timeline gives a good impression of the pattern of function events (**Context Menu > Events to show**). Zooming to a bunch of iterations again shows the staircase pattern observed above (see For the Impatient). Showing the transfer rates for messages results in a very instructive pattern for the given trace file.

A vertical line in the Qualitative Timeline either represents a single event (denoted as **Single** in the legend) or several events grouped together (denoted as **Multiple** in the legend). Refer to Level of Detail to find details on the merging of events.

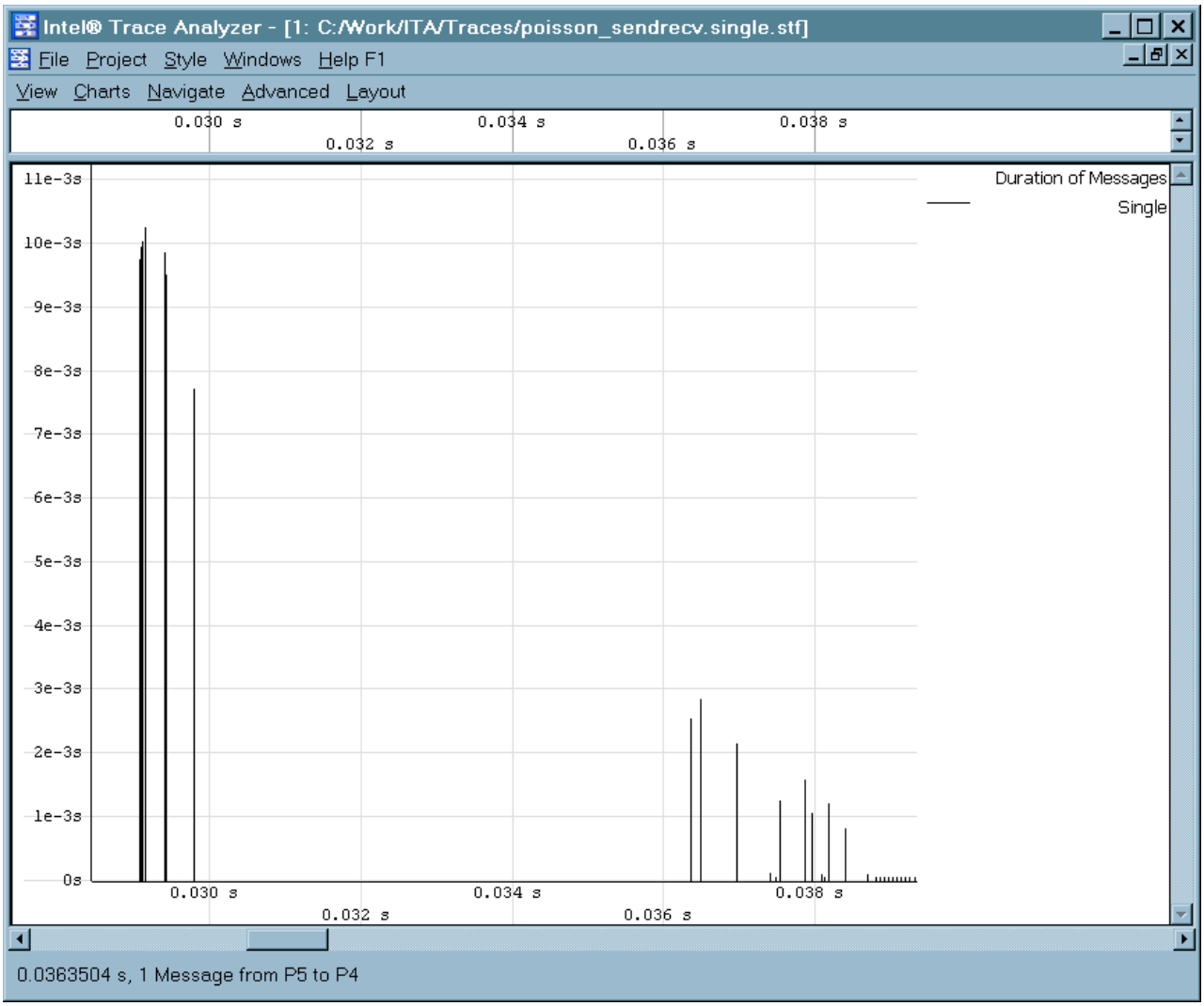
Figure 4.8 Qualitative Timeline



4.2.1 Mouse Hover

When the mouse hovers over the Qualitative Timeline Chart, the value of the x-axis (representing time) for the given position is shown in the status bar. The status bar also shows details regarding the given position of the mouse. For example, when the Qualitative Timeline displays the message attribute, the status bar shows how many messages are represented at that point. In case there is only one message, the sender and the receiver of the message are displayed as well. In the case where function events are displayed, mouse hovering indicates which Function/Group is present at the given position.

Figure 4.9 Status Bar When the Mouse Hovers over the Qualitative Timeline

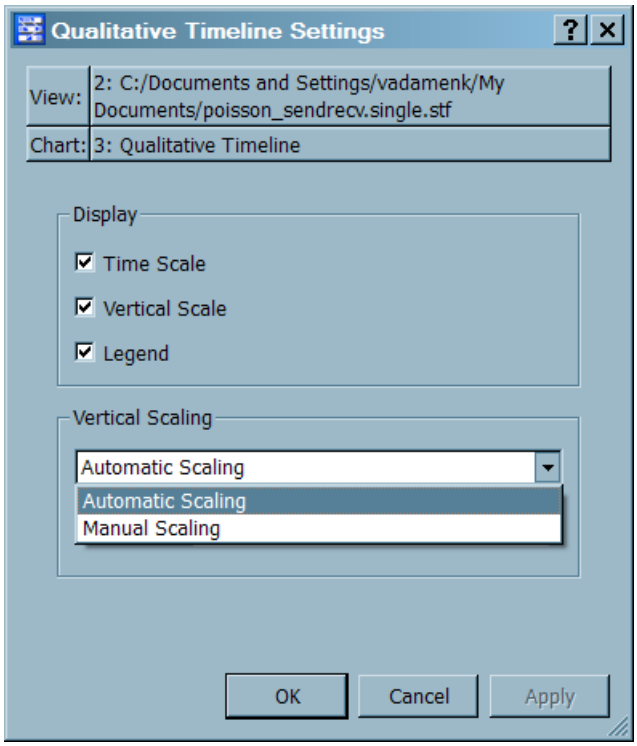


4.2.2 Qualitative Timeline Settings

The settings dialog box of the Qualitative Timeline consists of a Display group with check boxes and a Vertical scaling combo box. Use the combo box to adjust the vertical scaling of the timeline.

The **Display** Group specifies which scales to show and whether an own legend needs to be displayed for the timeline. By default, the vertical scale and the legend are shown.

Figure 4.10 Qualitative Timeline Settings Dialog Box



Use the **Vertical Scaling** group to switch between the default **Automatic Scaling** and the **Manual Scaling** of the y-axis. To explicitly specify the maximum scale value, use **Manual scaling**. To visually compare two or more Charts in the same or distinct Views, specify the same maximum value for the charts.

4.2.3 Context Menu

The entry **Context Menu > Events to show** allows to choose the event type from **Function Events**, **Messages** and **Collective Ops** through a sub-menu.

The entry **Context Menu > Attribute to show > xxx** allows to choose the particular attribute value of the event from **Duration**, **Transfer Rate** or **Data Volume** through a sub-menu.

NOTE: Not all attributes are available for all event types.

4.2.4 Filtering and Tagging

Tagged items in the Qualitative Timeline are highlighted with red. For example, to tag all messages sent by P9, open the **Tagging** dialog box (section Tagging Dialog) and go to the **Messages** tab.

In the group **Messages to be Tagged,** select the radio button **Custom**. Specify whatever needs to be tagged in the given field. *Figure 4.11* shows that P3 and P13 are tagged, resulting in the timeline shown in *Figure 4.12*.

Figure 4.11 Tagging in the Qualitative Timeline

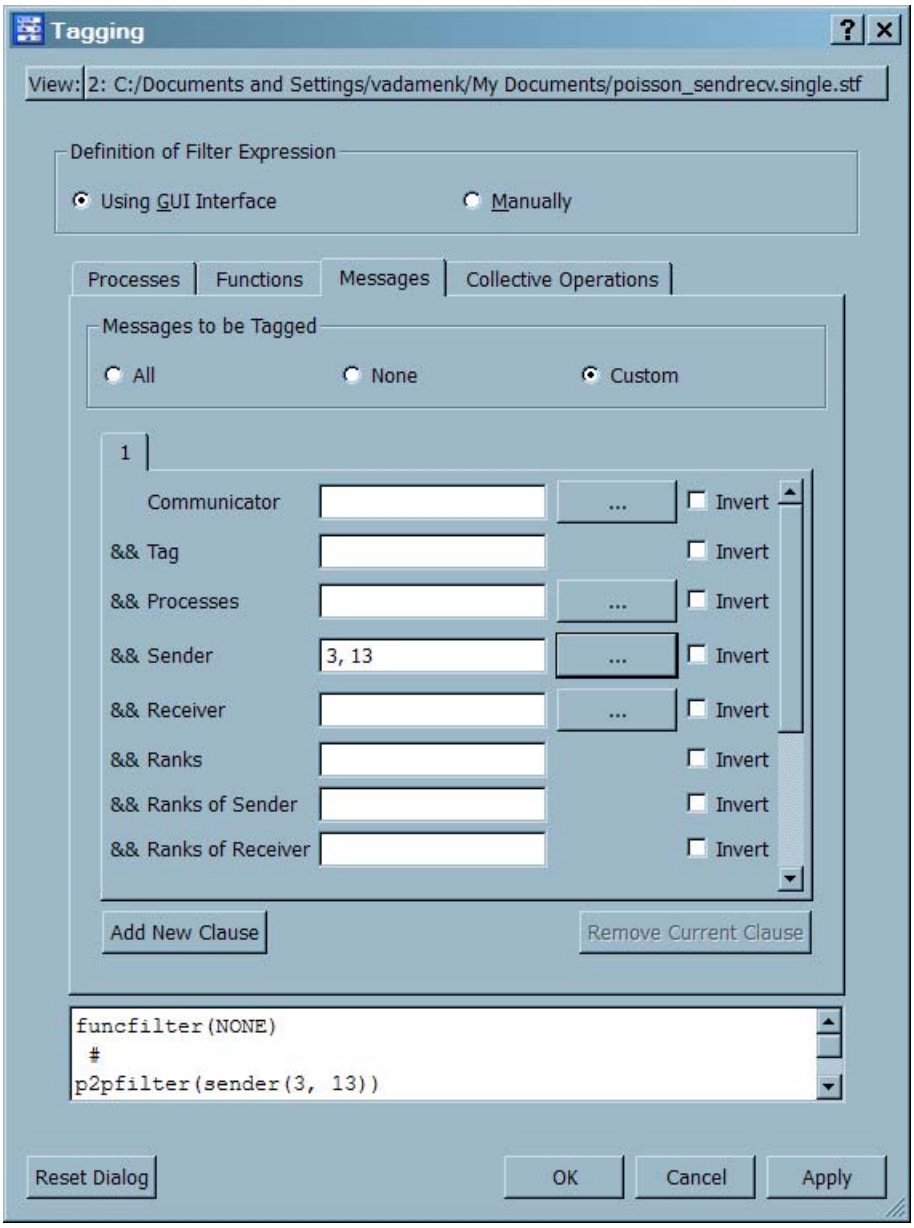
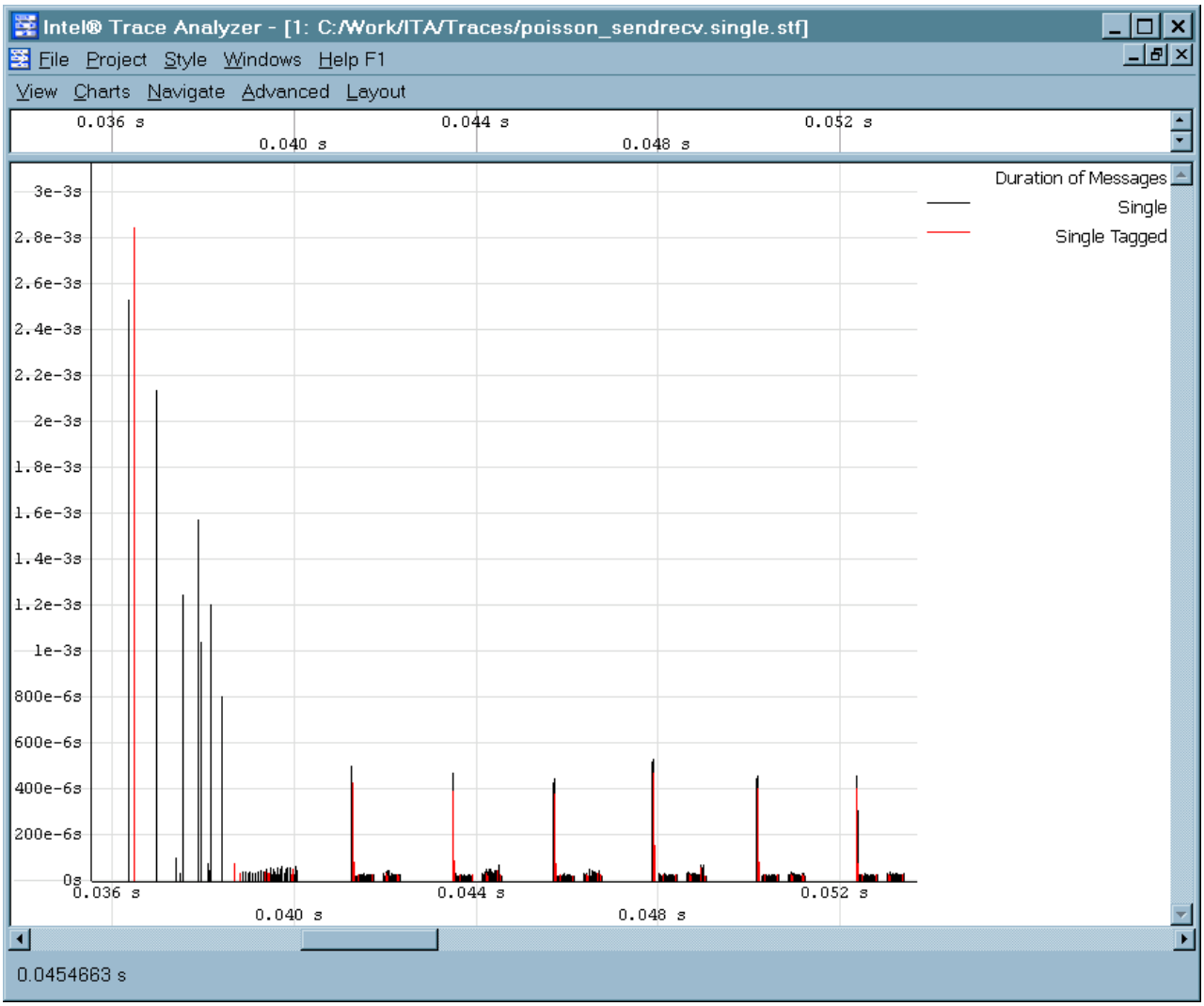


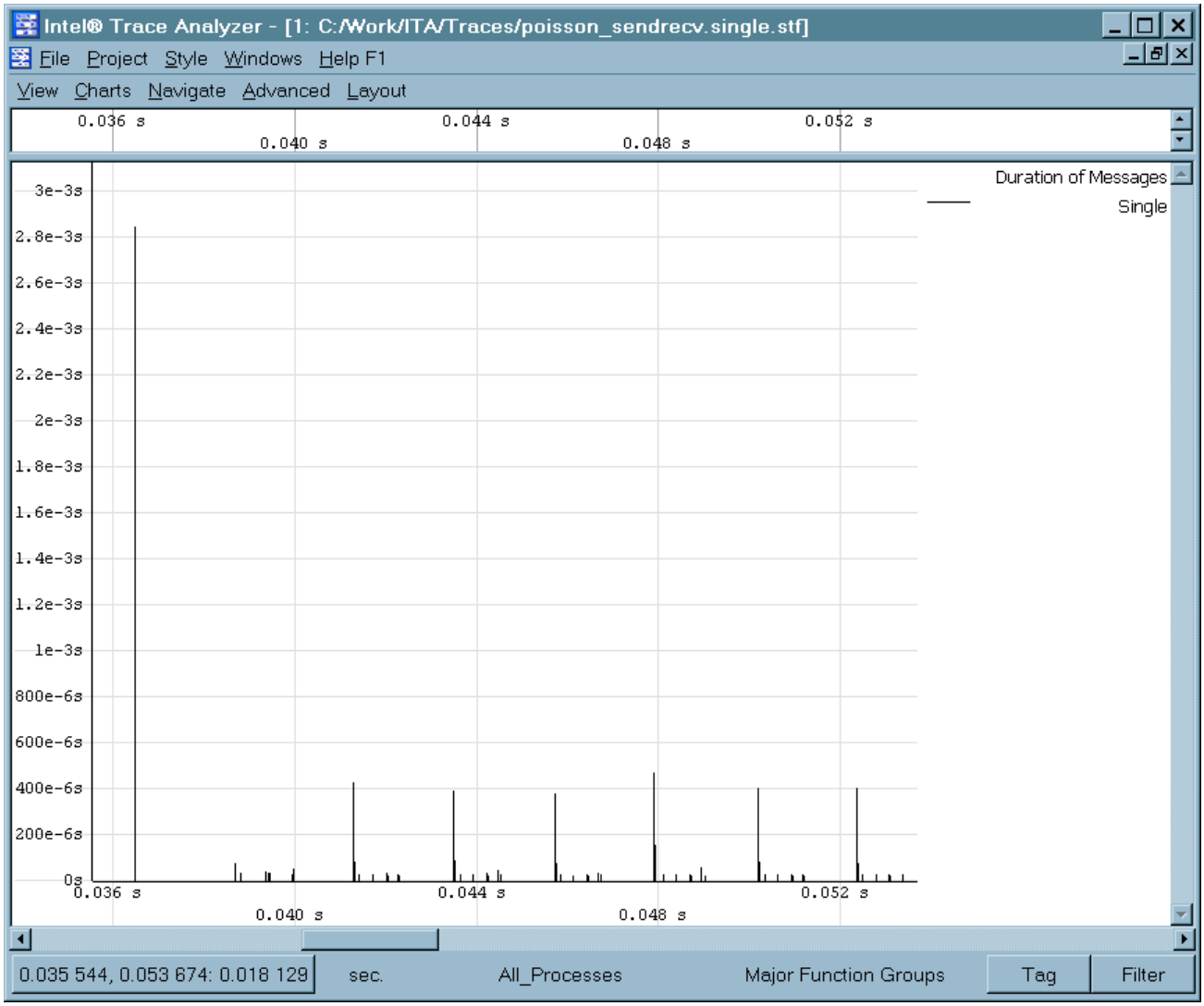
Figure 4.12 Qualitative Timeline with Tagged Messages



Similarly, to filter out all messages except those sent by P3 and P13, use the **Filtering** dialog box (section Filtering Dialog). It results in the following timeline:

Using tagging in the Qualitative Timeline is an efficient way to find specific events that occur infrequently. This is because it is guaranteed that a grouped multiple event is tagged if at least one of the singular events it represents matches the tagging filter expression.

Figure 4.13 Qualitative Timeline after Filtering



4.3 Quantitative Timeline

The Quantitative Timeline (**Views Menu > Charts > Quantitative Timeline**) gives an overview of the parallel behavior of the application. It shows over time how many processes or threads are involved in which function. Along the time axis, the different functions are presented as vertically stacked color bars. The height of these bars is proportional to the number of processes that are currently within the respective function.

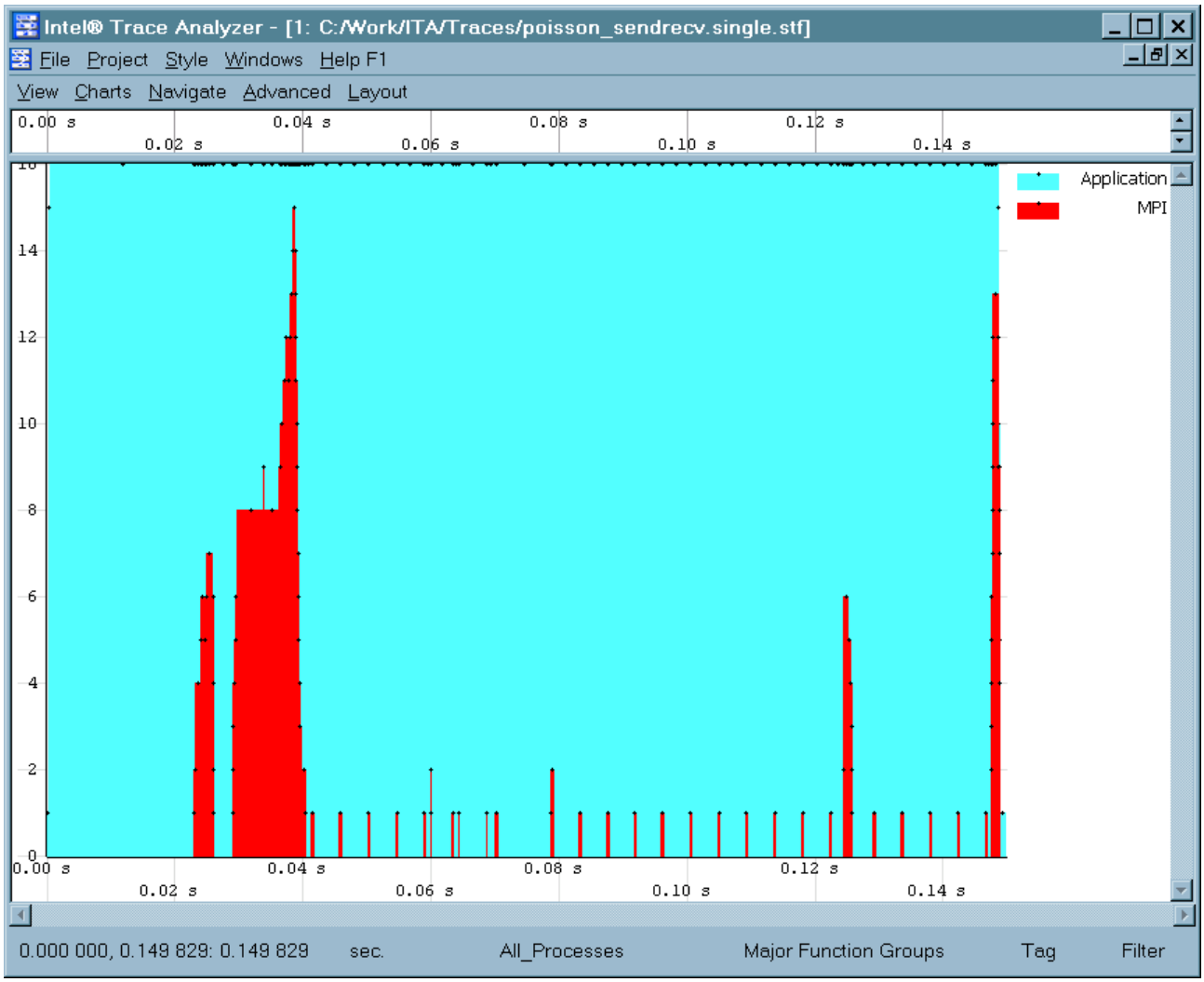
4.3.1 Mouse Hover

When the mouse hovers over the Quantitative Timeline, the status bar displays the present position of the cursor with respect to time. It also shows details pertaining to the function group under the cursor. *Figure* 4.14 shows that the mouse is currently at 0.0339729 seconds and that it hovers over MPI.

4.3.2 Quantitative Timeline Settings

The **Settings** dialog box in the Quantitative Timeline has two tabs: the **Preferences** tab and the **Functions** tab.

Figure 4.14 Quantitative Timeline without a Grid



Preferences Tab

Use the options in this tab to adjust the display options and the scaling of the timeline. It has a **Display** group and a **Vertical Scaling** group. In the **Display** group, there are six check boxes:

Time Scale: This displays a time scale along the x-axis.

Vertical Scale: This scale is on the y-axis. By default, this option is enabled so that the scale is shown.

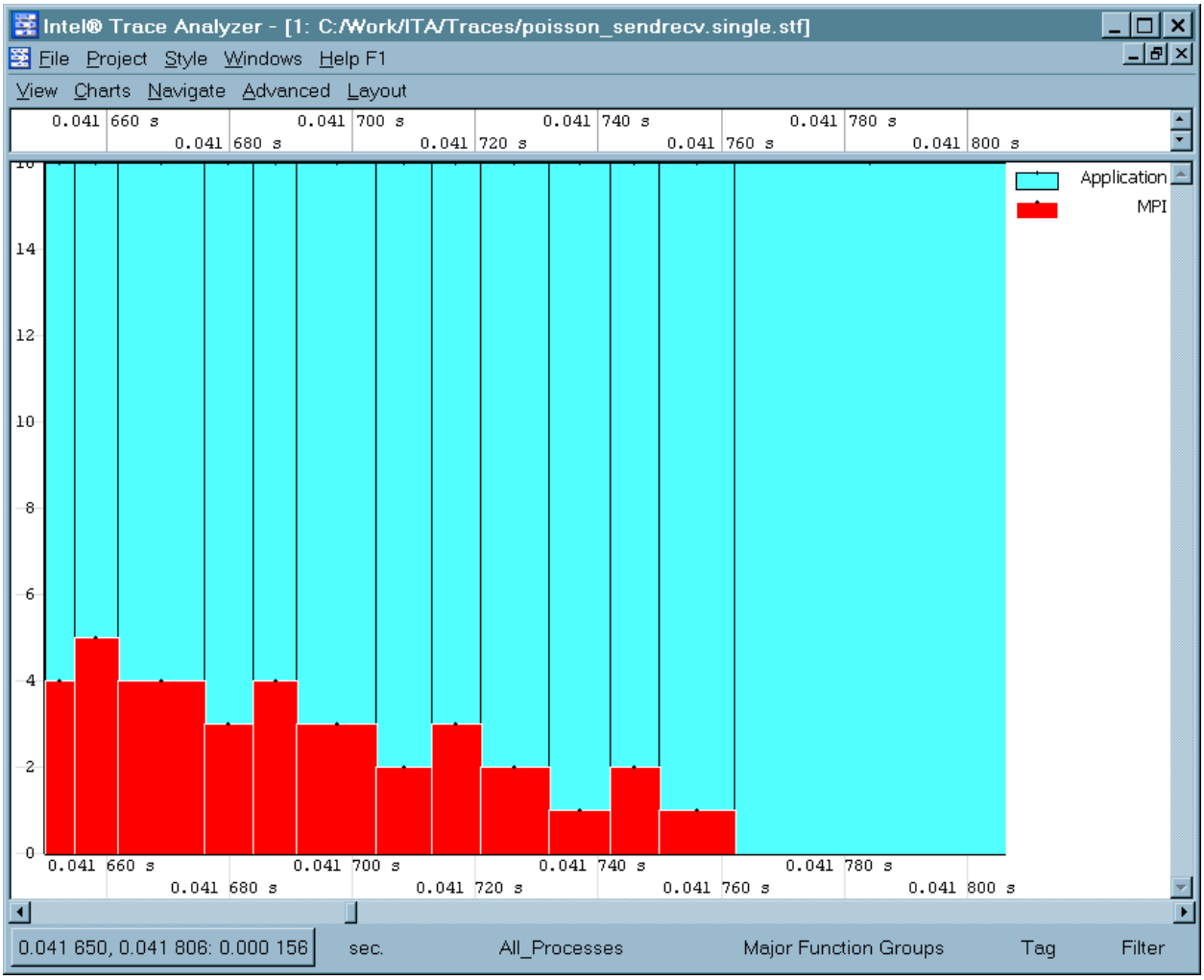
Legend: This shows the legend in the right margin of the timeline. This is also enabled by default.

Adjust Graphics to Legend Height: This forces the size of the diagram to be large enough to show all legend items.

Frames: Frames give an outline to the bars. The usage of frames becomes visible only when zooming in very closely such that each bar is separated from the other. *Figure* 4.15 illustrates the Quantitative Timeline with frames. In this figure, the Application function group is shown with a black outline, while the MPIs are shown with a white outline.

Grid: Use this checkbox to turn the grid on or off. The grid is drawn on top of the data and is aligned with the ticks on the scales. By default, this checkbox is activated.

Figure 4.15 Using Frames in the Quantitative Timeline



Vertical Scaling

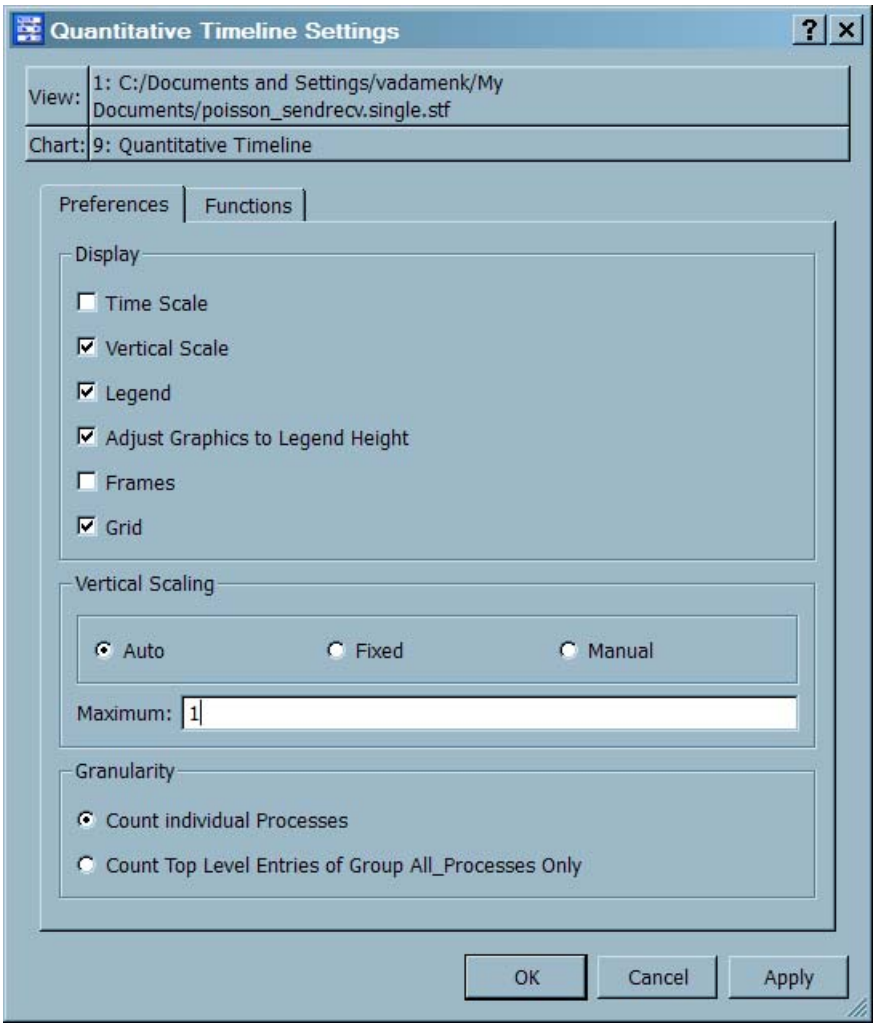
The vertical scaling option functions the same way as in the Qualitative Timeline (section Qualitative Timeline Settings).

Functions Tab

The **Functions** Tab selects the functions to be hidden/displayed. It is also possible to change the stacking order with this tab.

Radio buttons in this tab specify whether each individual process is counted (**Count individual Processes**) or whether merely the uppermost entries of Group All_Processes(**Count Top Level Entries of Group *All_Processes* Only**) are counted.

Figure 4.16 Quantitative Timeline Settings Dialog Box

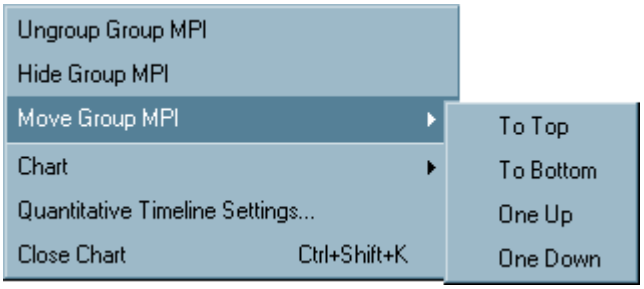


4.3.3 Context Menu

The context menu entry **Ungroup** in the Quantitative Timeline provides the option of ungrouping the given function group, like in the Function Profile (section Context Menu). Similarly, **Regroup** allows undoing a previous ungrouping.

The **Hide** option in the context menu conceals the chosen activity. To unhide all the hidden items, use the check boxes in the **Functions** tab of the **Settings** dialog box.

Figure 4.17 Quantitative Timeline: Context Menu



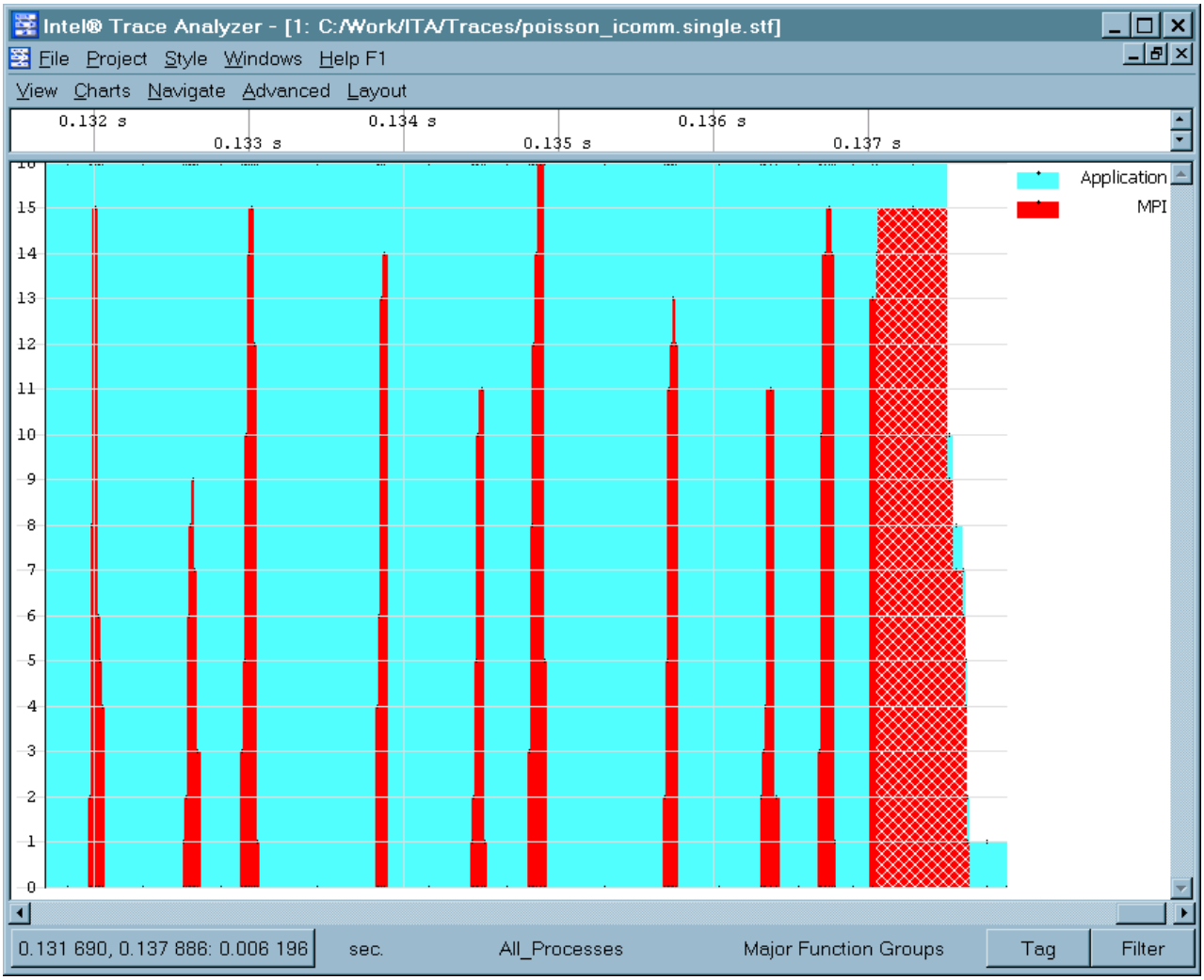
The **Move Group** entry in the context menu changes the position of the different groups. The opted group can be moved to the top, to the bottom, upward by one position or downward by one position. Click on the legend to obtain a context menu with these options.

The other context menu entries are explained in the section Common Chart Features.

4.3.4 Filtering and Tagging

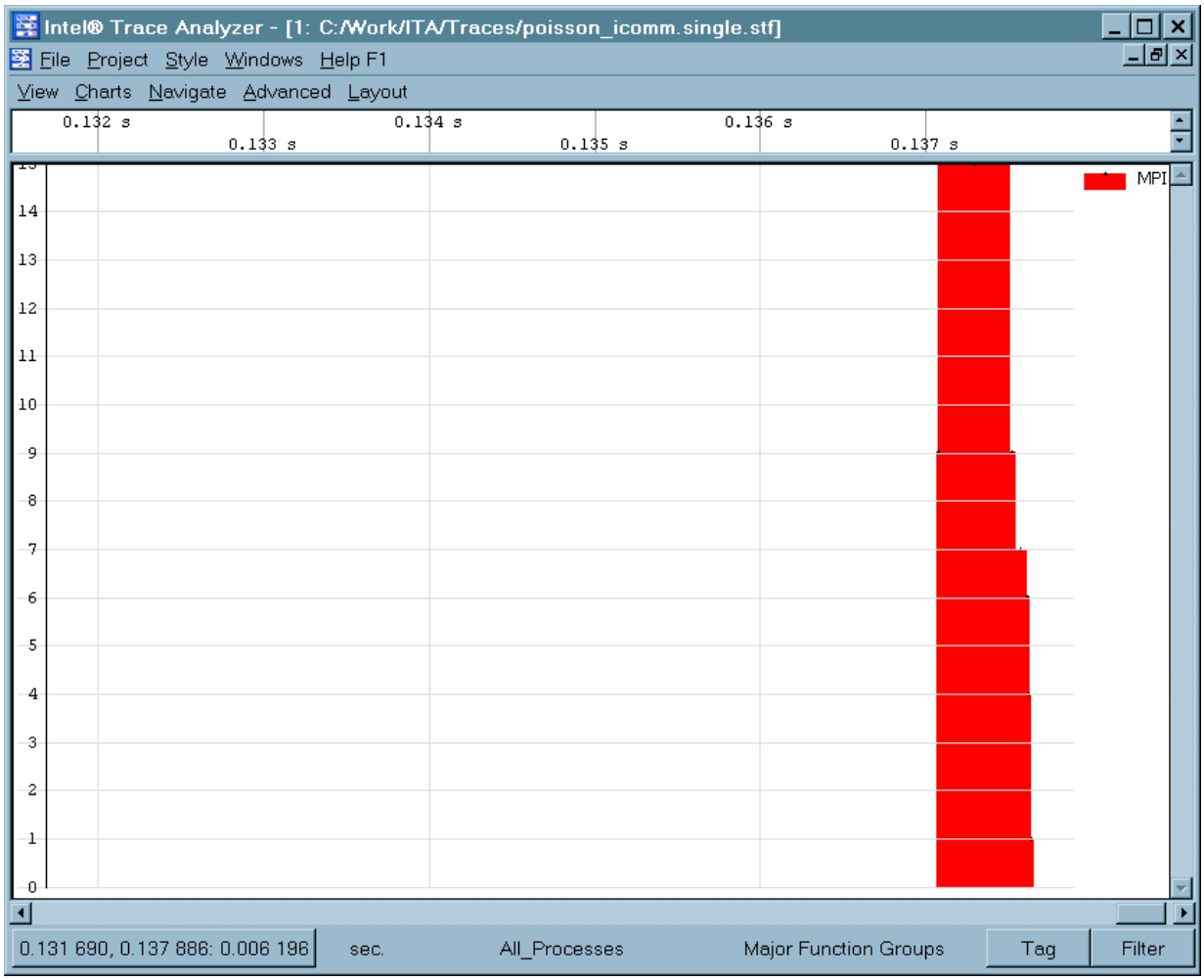
Tagging in the Quantitative Timeline is indicated by a mesh pattern being placed over the tagged item. *Figure* 4.18 shows a Quantitative Timeline for the file `poisson_icomm.single.stf` with the tagged function `MPI_Finalize` on the right-hand side of the display.

Figure 4.18 Tagging the *MPI_Finalize* function in the Quantitative Timeline



Filtering in the Quantitative Timeline works the same way as in any other Chart. *Figure* 4.19 shows the result when only `MPI_Finalize` passes the filter.

Figure 4.19 Quantitative Timeline after Filtering



4.4 Counter Timeline

The Counter Timeline shows the values of all counters that a given trace file provides. The Intel® Trace Collector records user-defined counters or counters provided by the operating system. See the Intel® Trace Collector documentation to learn more about tracing counters.

Figure 4.20 Settings Dialog Showing All Counters Available in the Trace with Their Type and Scope

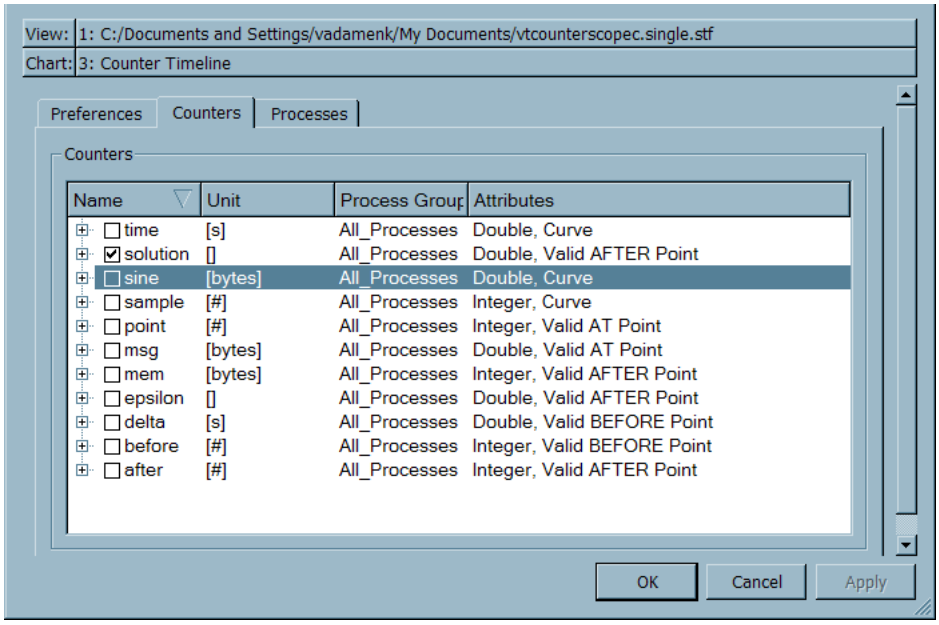
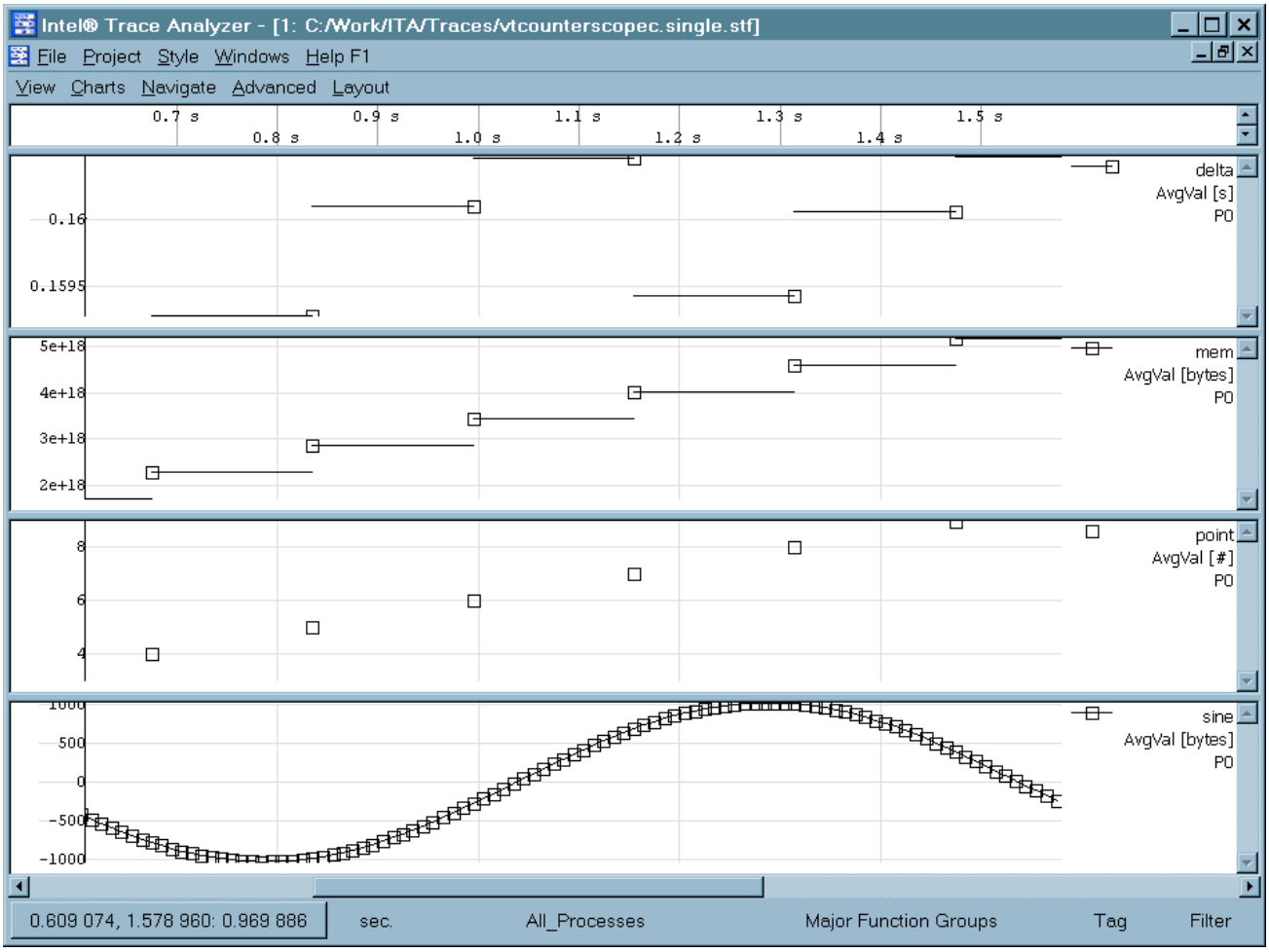


Figure 4.21 Some Counters with Differing Scopes, Zoomed in



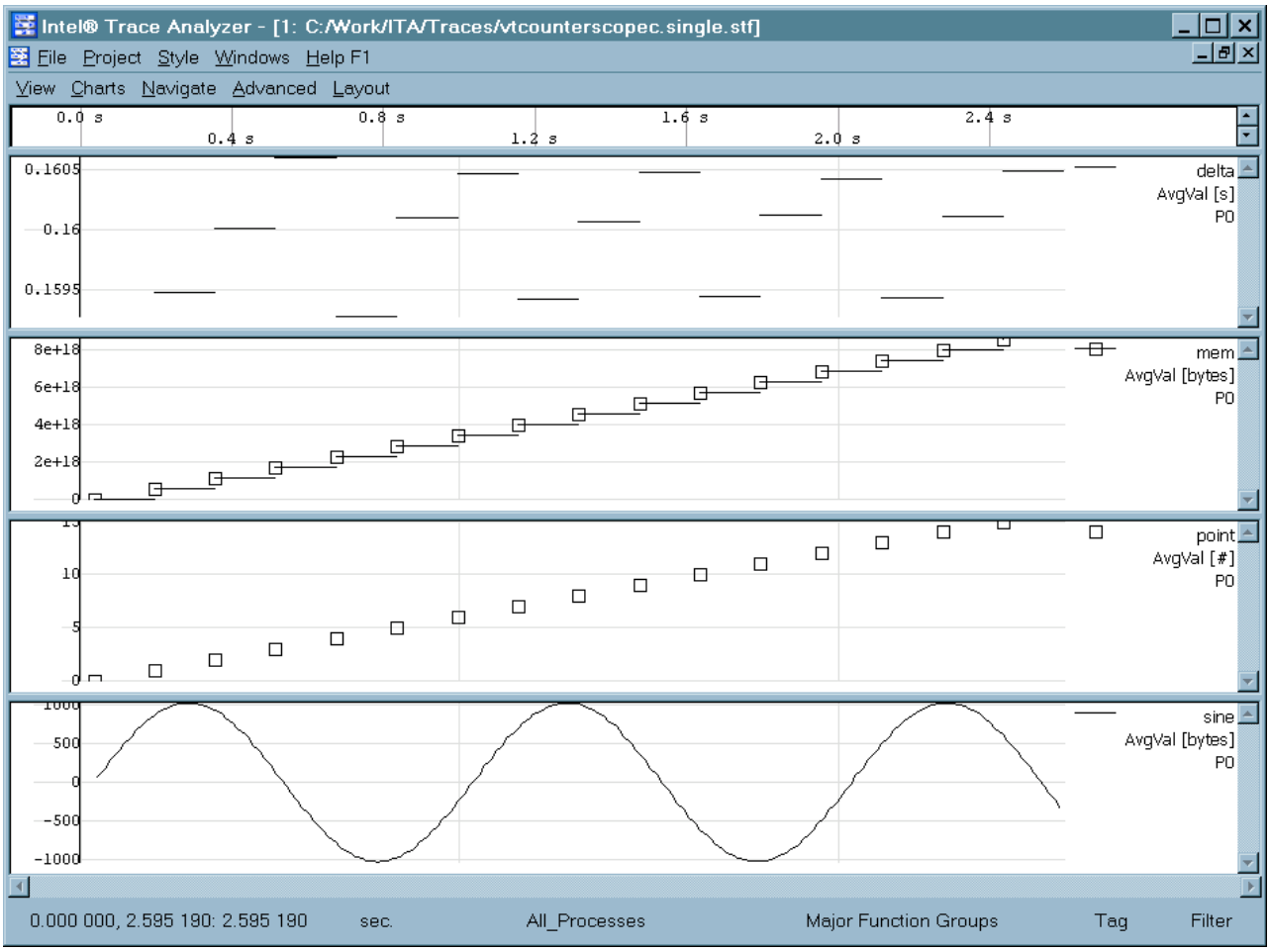
The Counter Timeline as shown in *Figure* 4.21 shows counters with the scopes Valid BEFORE point, Valid AFTER point, Valid AT point and Curve. The time interval is small enough so that no counter samples are merged over time (section Level of Detail). The sample points are indicated by markers.

The sample points for counters with the attribute Curve are connected by line segments. Samples with the attribute Valid AT point are shown by markers. Samples of counters with other scopes are shown by short, unconnected horizontal line segments before or after the sample.

The Counter Timeline as shown in *Figure* 4.22 shows the same counters as in *Figure* 4.21 but zoomed out.

NOTE: There are no markers anymore because the values shown represent samples that were merged over time. To force that effect the View was made very narrow.

Figure 4.22 Some Counters with Differing Scopes, Zoomed out



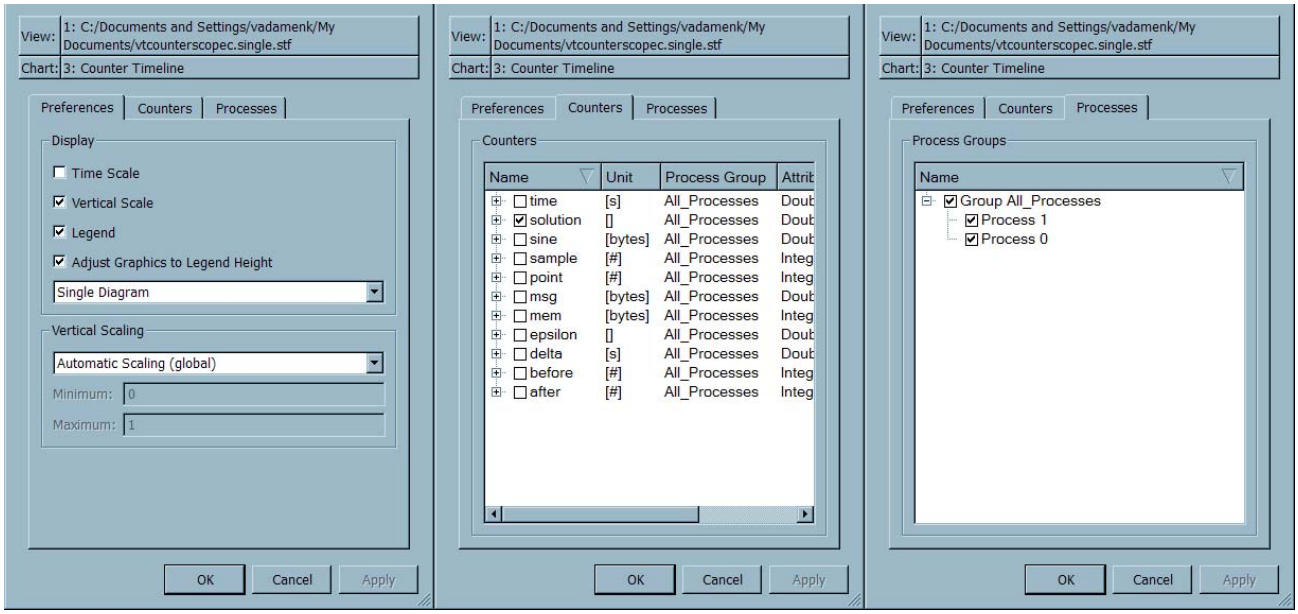
4.4.1 Mouse Hover

When the mouse pointer hovers over the Counter Timeline, its exact position in time is shown in the View status bar. If the mouse pointer is near a sample point then the respective value is shown.

4.4.2 Counter Timeline Settings

The **Settings** dialog box in the Counter Timeline has three tabs: the **Preferences** tab, the **Counters** tab and the **Processes** tab.

Figure 4.23 Counter Timeline Settings Dialog Box



Preferences Tab

Use the controls in this tab to adjust the display settings and the scaling of the Counter Timeline. The tab consists of the groups **Display** and **Vertical Scaling**.

The group **Display** contains the following controls:

Time Scale displays time scale along the x-axis.

Vertical Scale displays time scale along the y-axis. This option is enabled by default so that the scale is shown.

Legend shows the legend in the right margin of the timeline. It is also enabled by default.

Adjust Graphics to Legend Height forces the size of the diagram to be large enough to show all legend items.

The combo box showing either **Single Diagram** or **Many Diagrams** allows to have either a single diagram for all selected counters and all selected target group children or to create a separate diagram for each target group child.

Use the **Vertical Scaling** group to switch between the **Automatic Scaling (per Diagram)**, **Automatic Scaling (global)** and the **Manual Scaling** of the y-axis. To explicitly specify the minimum and maximum scale values, use **Manual Scaling**. To visually compare two or more Charts in the same or distinct Views, specify the same bounds for both charts.

Counters Tab

This tab holds a list with seven items for each counter provided by the trace file. The top level entry for each counter shows the counter Name, its Unit, its target Process Group and its Attributes (*Figure* 4.23). The name and the unit are arbitrary, free-format strings defined in the trace file.

The column **Process Group** contains the counter target group. A counter that has different values for each process will have the target group All_Processes, while a counter that has a distinct value for each SMP node will have the target group All_Nodes.

The column **Attributes** can contain the following attributes: either Integer or Double, indicating the counter type.

One of **Valid BEFORE Point**, **Valid AFTER Point**, **Valid AT Point** or **Curve** indicates the counters scope. Curve indicates that it is meaningful to interpolate values between two given counter values.

The attribute **Show Rate**, indicating that it is preferable to display the derivation to time instead of the plain counter values.

Nested under the top level entry for a counter, there are six entries for the minimum, average and maximum values and for the minimum, average and maximum rates that allow to switch each on and off independently. If you just use the top level entries check box then either the average value or rate will be chosen depending on the counters attributes.

Processes Tab

This tab contains one top-level entry for each target group used in at least one of the counters and a nested entry for each target group's child.

Using the top-level entry for a target group you can switch the entire target group on or off, or go back to the last subset that you selected using the second level entries.

Using the second level entries you can switch on and off arbitrary children of the target group.

The settings for a target group in this tab are applied to all counters with the respective target group.

4.4.3 Context Menu

The context menu of the Counter Timeline provides the common entries as defined in Common Chart Features.

4.4.4 Filtering and Tagging

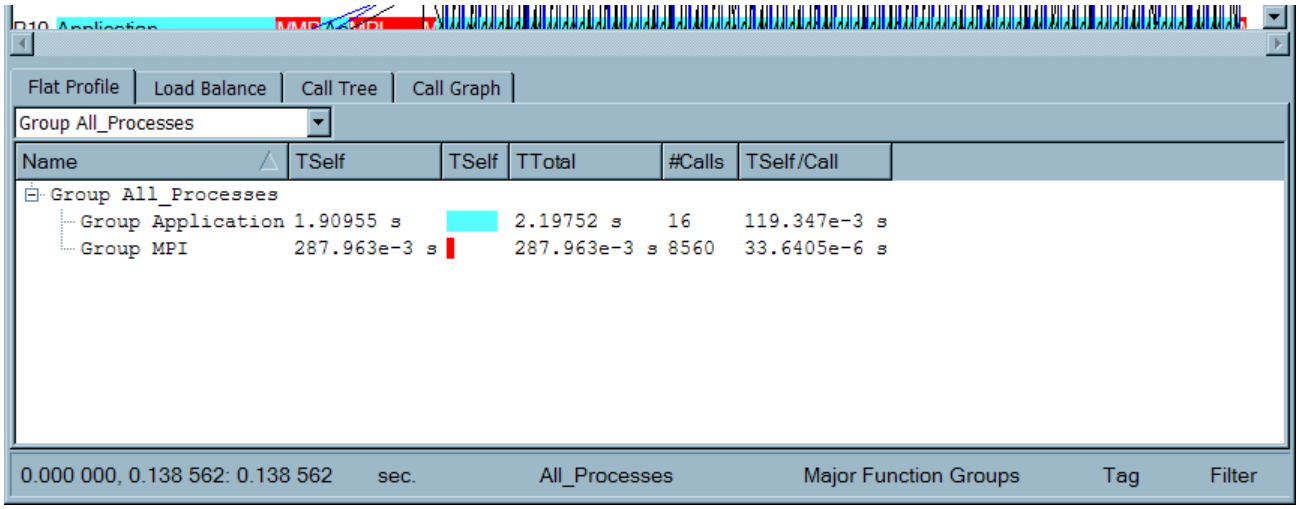
The filter mechanism in Intel® Trace Analyzer does not cover counters. Therefore the Counter Timeline is independent of the View filter settings.

4.5 Function Profile Chart

The Function Profile provides detailed profiling information on the performance data. It consists of four different tabs, namely the **Flat Profile** tab, the **Load Balance** tab, the **Call Tree** tab and the **Call Graph** tab.

All four of these tabs use the same column headers with the same semantics, and use the same raw data. The default column headers on display are Name, TSelf, TTotal, #Calls and TSelf/Call. For a detailed explanation of all available column headers refer to Function Profile Settings. The order of these columns is adjusted by dragging headers of columns.

Figure 4.24 Function Profile



To sort a list in ascending or descending order, click on a column header. To see which process spends the most time (or the least time) in a function, click **TSelf** and the entries are sorted by this column. The arrow symbol in the column header indicates whether it is arranged in ascending or descending order.

NOTE: Sorting by the name column does not sort alphabetically. Instead, it sorts in the order given by the layout of the current process or function group.

The number formatting options are preset globally through the **Number Formatting Settings** dialog (section Number Formatting Settings). To increase the number of digits locally by three (or one) digits press the key `[+]` (or `[Ctrl]+[+]`). Use the keys `[-]` (or `[Ctrl]+[-]`) to do the opposite.

NOTE: The exact effect of asking for additional digits depends on the format chosen in the **Number Formatting Settings** dialog for the respective unit.

4.5.1 Flat Profile

By default, the Flat Profile summarizes all major groups of functions and presents statistics over the processes. The exact contents of these groups depend on the group definitions stored in the trace file or as defined by the user; in the file `poisson_icomm.single.stf`, this is only MPI and Application.

The Chart in *Figure 4.24* shows that most of the time was spent in Application. To see the distribution of execution time over the individual MPI routines, right-click on the MPI entry and select **Ungroup Group MPI** from the context menu as shown in *Figure 4.25*.

This causes the single MPI entry to be replaced by several entries – one for each MPI function (see *Figure* 4.26). To regroup the children of MPI, right-click on a child and choose **Regroup MPI** from the context menu or select Major Function Groups from the Function Group Editor (**Views Menu > Advanced > Function Aggregation**).

Figure 4.25 Ungrouping the Function Group MPI through the Context Menu

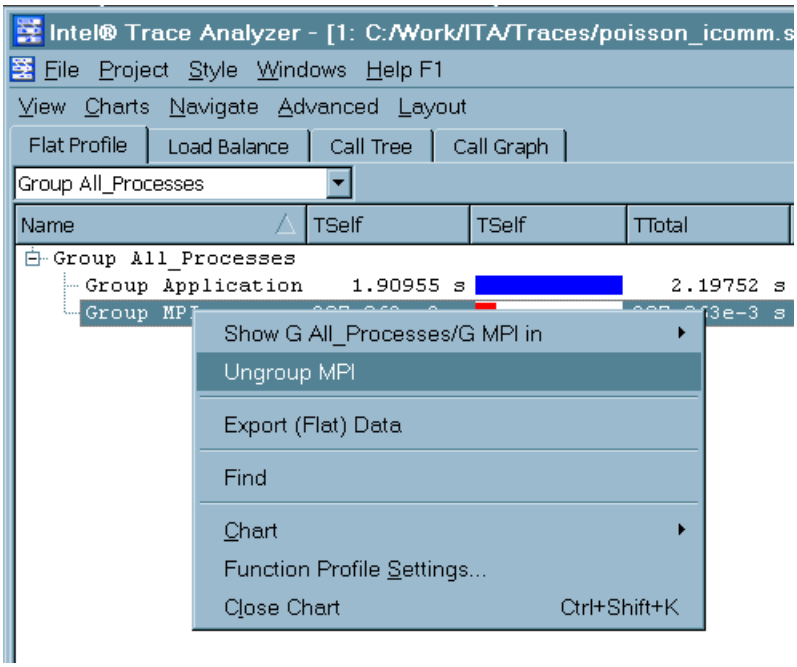
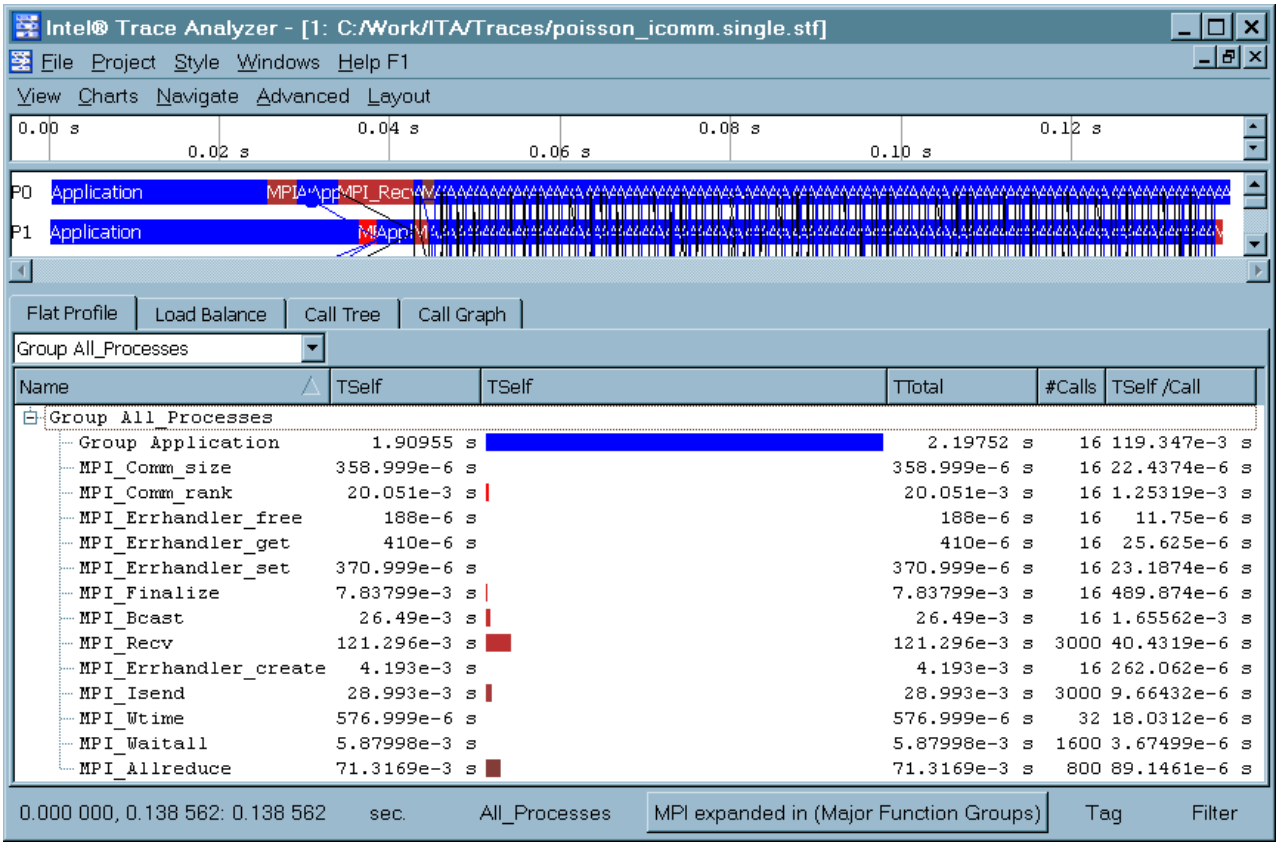


Figure 4.26 Flat Profile after Ungrouping MPI



The default settings ensure that all statistics are summarized over all threads into a single profile. All tabs provide the option of viewing the data for each process separately. To do this, use the combo box at the top of the tab as shown in *Figure 4.27*. For example, selecting **Children of Group All_Processes** results

in Figure 4.28. The processes are now listed as the top-level entries in the tree (first column). To expand and collapse the processes of interest, use the plus and minus handles (see Figure 4.28).

Figure 4.27 Selecting Profiles per Process

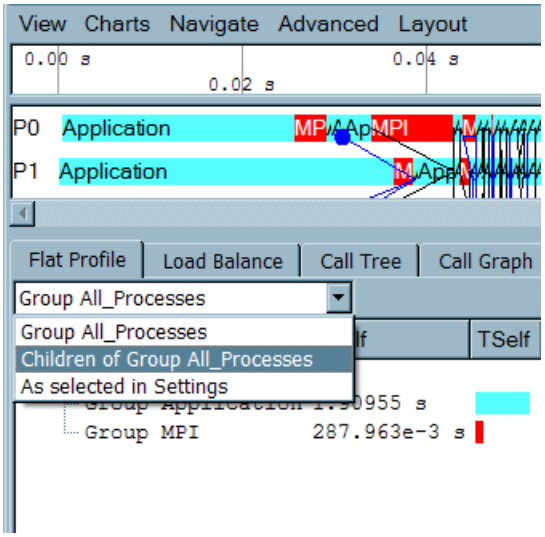
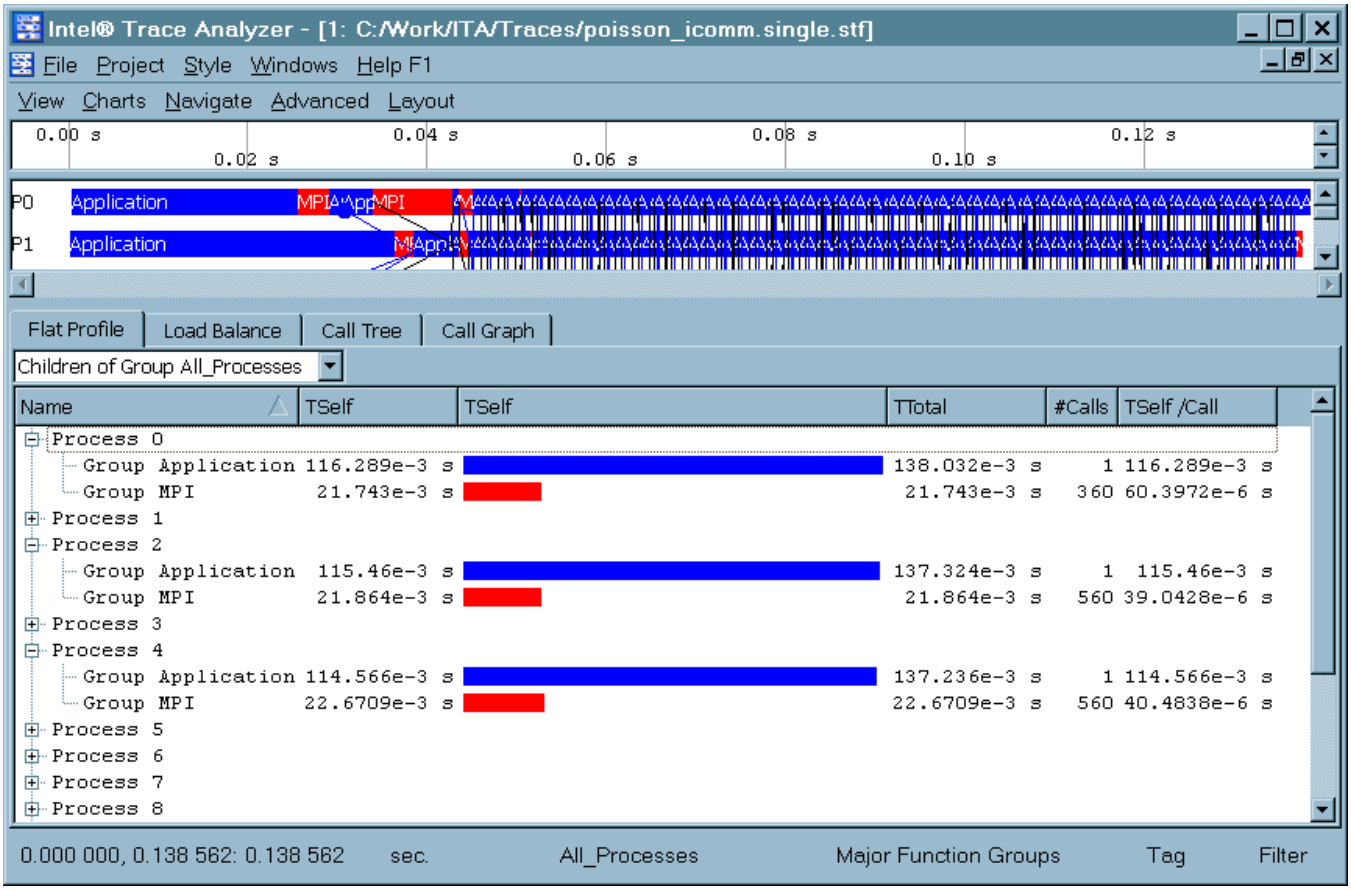


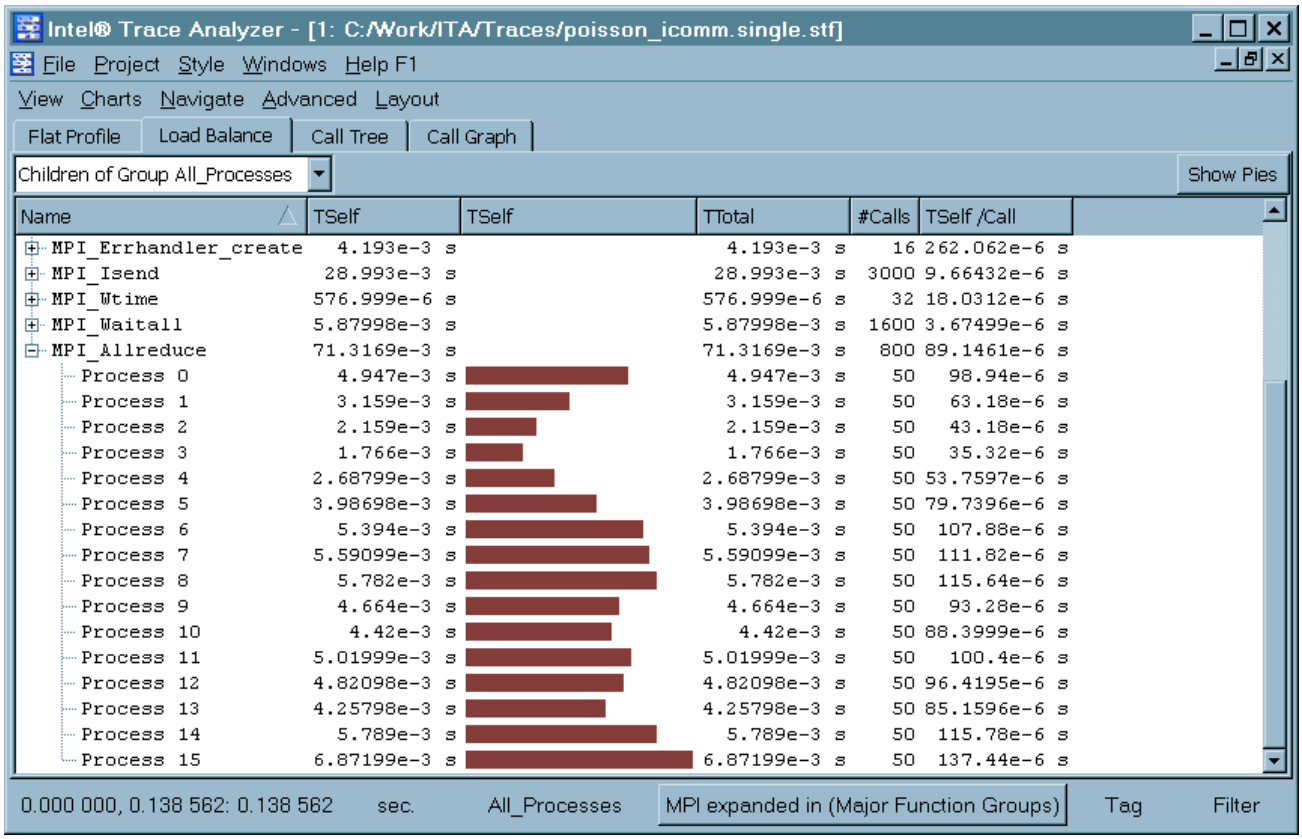
Figure 4.28 Showing Children of Process Group All Processes



4.5.2 Load Balance

The **Load Balance** tab displays the same data as the Flat Profile except that it is grouped by function instead of by process. The **Load Balance** tab compares the profiles of the same function across several processes. The top level entries of the tree given in the first column are functions. *Figure* 4.29 shows that TSelf for MPI_Allreduce is pretty unbalanced across processes.

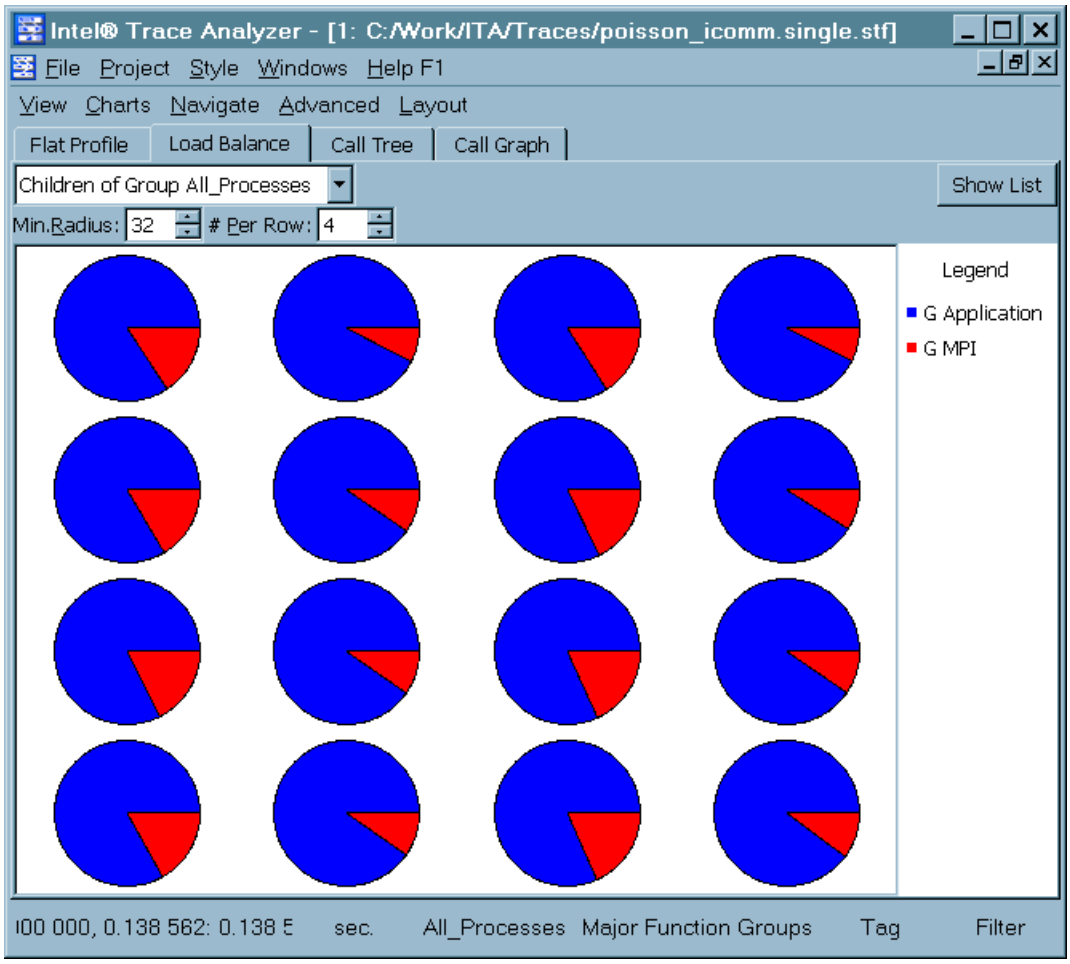
Figure 4.29 Load Balance for MPI_Allreduce



As in the Flat Profile, the Load Balance summarizes the statistics into a single profile using Group All_Processes. To view the data for each individual process in a given function, use Children of Group All_Processes. Similarly, the functions in the **Load Balance** tab are ungrouped using **Ungroup**/**Regroup** on the context menu. Ungrouping displays all major function groups. To group all processes together and to view it as a single profile, select **Group All_Processes**.

The **Load Balance** tab offers to display the data in form of pie diagrams (*Figure* 4.30). The button in the top right corner of the tab allows switching back and forth between the usual list and the pie diagrams. This allows to judge the overall load balance pattern (for TSelf) even among a huge number of processes in a relatively confined space. Above the pie diagrams there are two spin buttons. The left one controls the minimum radius of the pies and the right one controls how many pie diagrams appear in a row.

Figure 4.30 Pie Diagrams in the Load Balance Tab

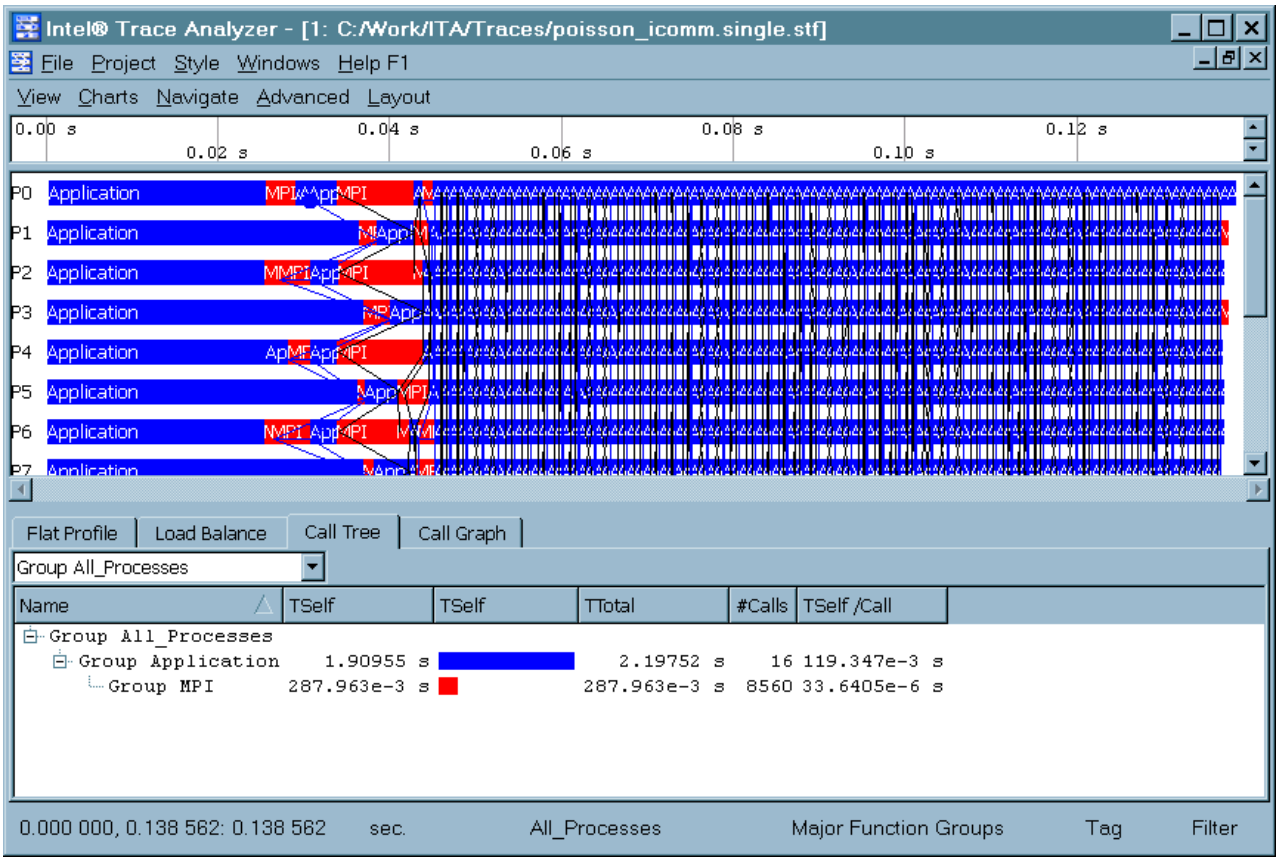


4.5.3 Call Tree

When **Load Balance** and **Flat Profile** tabs do not show enough detail, use the **Call Tree** tab to include calling dependencies in your analysis. The **Call Tree** tab shows the same information as the Flat Profile and Load Balance, but also includes the calling hierarchy.

Select a certain entry in the Call Tree to focus on it. The focus remains on this entry even when the time interval is changed due to scrolling or zooming. It stays selected and visible when possible. If a corresponding entry is absent for the new time interval, its parent is selected. This feature is very useful in large and deeply nested call trees.

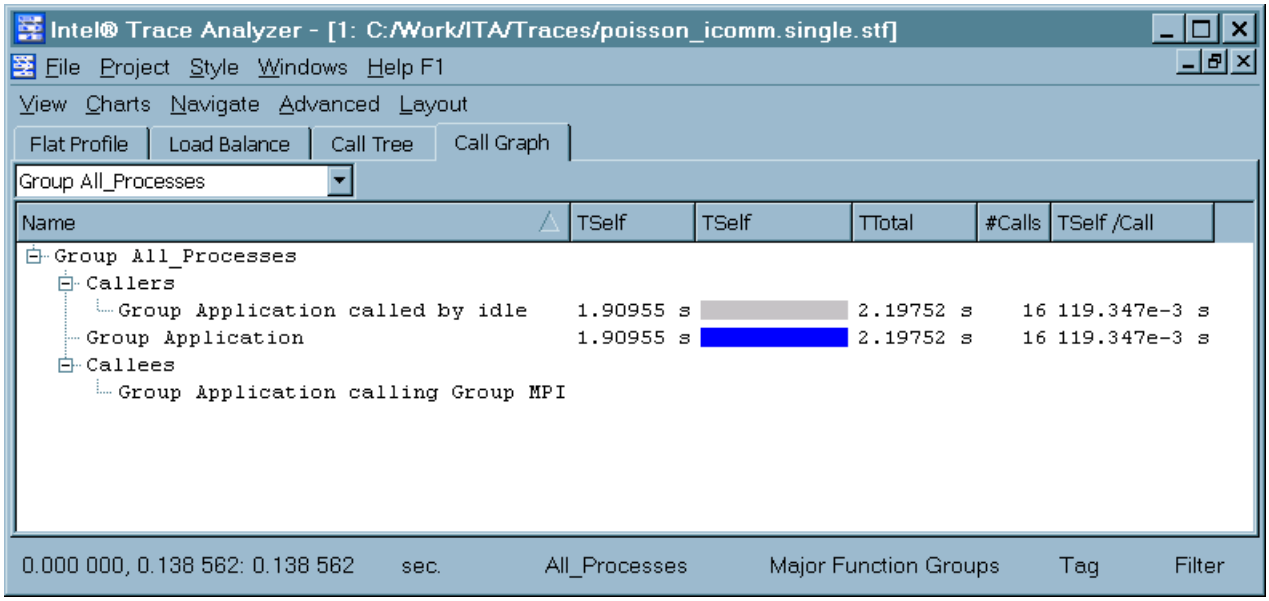
Figure 4.31 Call Tree Tab



4.5.4 Call Graph

The **Call Graph** tab shows a small part of the call graph for each process or process group: a single node (called central function) with its inbound and outbound edges. Each process entry has three children: the Callers, the central function and the Callees.

Figure 4.32 Call Graph Tab



To navigate through the Call Graph, double-click on a caller or callee and press the space bar or Enter key to make the respective function the central node.

The time shown for the central function is the same as shown in the **Flat Profile** tab and the **Load Balance** tab. The times shown for the callers represent the time spent in the central function when called from the respective function.

If a function is used within different contexts (by different algorithms for example), it can be observed which algorithm causes a function to consume more or less time. In *Figure* 4.32 it is seen which caller is responsible for most of the time spent in MPI: it is the function group Application (and not Forward, Adjoin, cg or Smoother). Using the Call Graph this way helps finding places in the code that cause expensive calls, even when the call tree gets too big to navigate through it.

4.5.5 Using the Function Profile

The following sections describe the columns headers of the Function Profile and how to define these headers using the Function Profile Settings dialog box.

4.5.5.1 Function Profile Settings

The **Function Profile Settings** dialog box enables customizing display options for all four tabs of the Function Profile Chart. To access the **Settings** dialog box, right click and select **Function Profile settings** from the context menu.

Preferences Tab

In the **Preferences** tab, there are four groups of options. The first one is the **Display Group**, which consists of check boxes. Use these check boxes to select the attributes to be displayed. There are a total of eight attributes available, out of which four are selected by default (**Time Self**, **Time Total**, **#Calls** and **Time Self per Call**). All eight attributes are described below:

Time Self (TSelf): Time spent in the given function, excluding time spent in functions called from it

Time Total (TTotal): Time spent in the given function, including time spent in functions called from it

#Calls: Number of calls to this function. This can be zero even if other attributes are non-zero, because the actual calls to the respective function can occur outside the current time interval.

Time Self per Call (TSelf/Call): Time Self averaged over #Calls

Time Total per Call: Time Total averaged over #Calls

#Processes: Number of processes in this function

Time Self per Process: Time Self: averaged over #Processes

Time Total per Process: Time Total averaged over #Processes

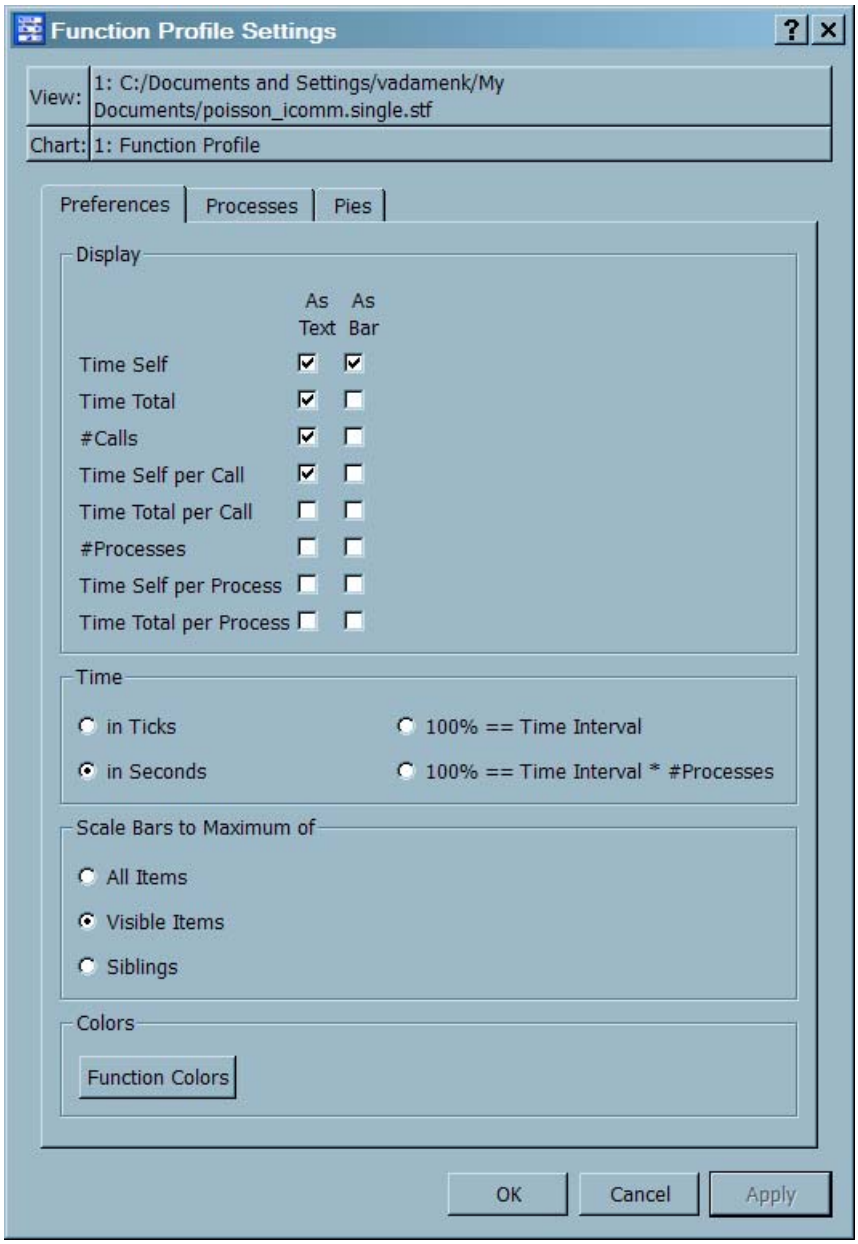
Using the given check boxes, the displaying of the above attributes either as text or as a bar graph can be switched on or off independently.

Use the radio buttons to specify the format for time (seconds or ticks) or to specify time as a percentage of the time interval.

There are three scaling modes in the **Preferences** tab given as radio buttons. The default (**Visible Items**) scales the bars to the respective maximum of all expanded items. **All Items** uses the global maximum of all values, regardless if they are expanded or not. **Siblings** uses only the maximum of the direct siblings. In all three scaling modes only values from the same column are taken into account.

At the bottom of the Preferences tab, there is a **Function Colors** command button. Clicking on this opens the **Function Group Color Editor** (section Function Group Color Editor).

Figure 4.33 Function Profile Settings Dialog



Processes Tab

In the **Processes** tab, select the processes to be displayed in the Chart by enabling the check box of the process. After selecting these in the **Processes** tab, the selected processes are shown in any of the **Function Profile** tabs by choosing the **As selected in Settings** option from the combo box (see *Figure* 4.27). An easy way to select all but one process is to choose the process not required and then using the

Invert All to reverse the selection. Doing this has no influence on the current process group of the View, it only allows to focus the Function Profile on a subset of all processes.

Pie Tab

The **Pie** tab contains check boxes to switch the individual diagram titles and the global legend on and off.

4.5.5.2 Context Menu

The context menu, obtained by right-clicking on an item, contains a set of operations that are performed on the clicked item and on the Chart as a whole. The context menu adjusts itself to suit the selected entry in the Chart.

The **Show All_Processes / xxx** entry in the context menu shows the given profile in a different tab. Here **xxx** stands for the Function group name. For the given example, this would be either the function groups MPI, Application or the function group Other.

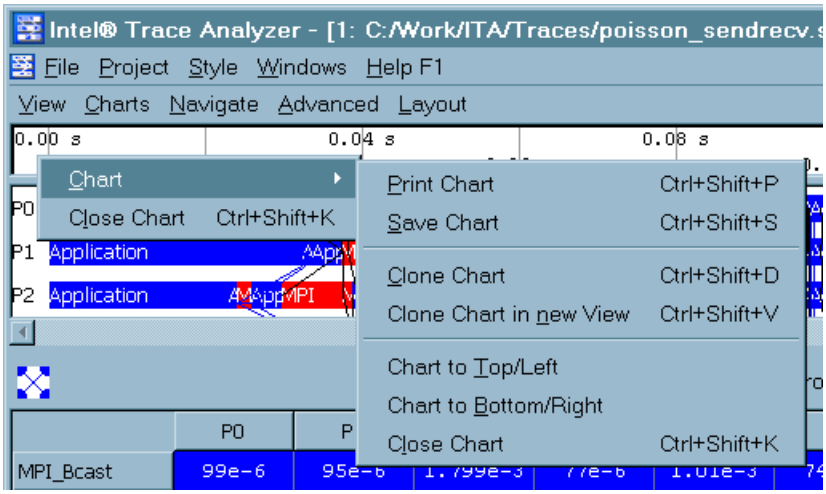
Another context menu entry is the **Ungroup** option. This ungroups the selected group and shows the distribution of execution time over the individual routines, as is illustrated in *Figure* 4.25 and *Figure* 4.26. To revert the ungrouping, right-click a child of a recently ungrouped function group and select **Regroup** from the context menu. To restore the summarized display after ungrouping a number of times, it is easier to open the Function Group Editor using **Views Menu > Advanced > Function Aggregation** and select **Major Function Groups**.

The **Find** entry searches for a process/function (section Find Dialog).

To save the flat profile data in text form, choose the context menu entry **Export (Flat) Data**. This opens a **File Save** dialog box. Specify the filename or choose the file in which to store the data here. This includes all data of the flat profile also taking into account the child processes. The default option is to save it as a .txt file.

Context Menu > Charts opens another sub-menu, which contains entries to print, save, clone and move the Chart (section Common Chart Features).

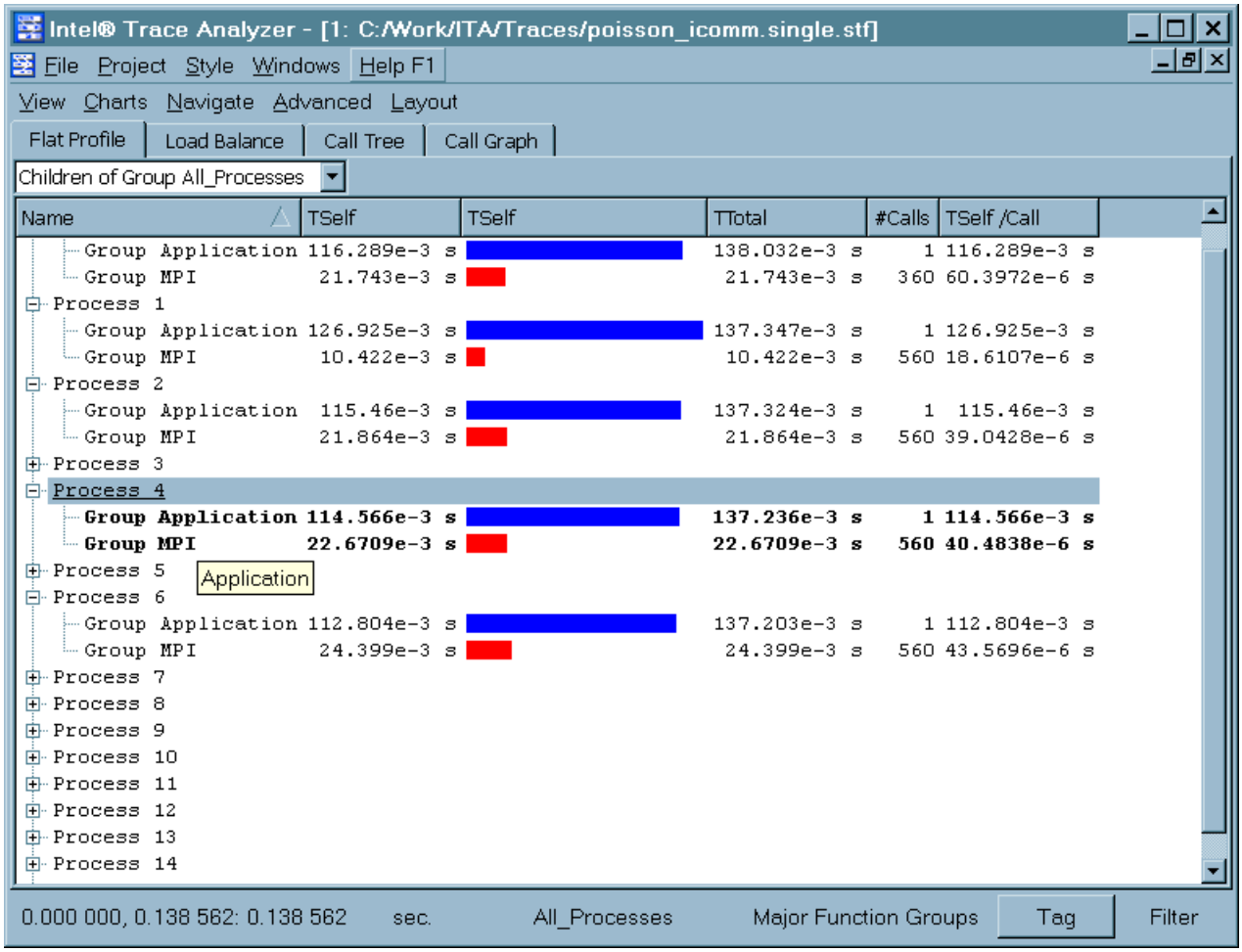
Figure 4.34 Context Menu with a Submenu



4.5.5.3 Filtering and Tagging

Tagged entries are shown using a bold font for the name column. Entries with tagged descendants are shown with underlined names. This helps to see or find the required entry, especially when the tree is large. For more details on tagging and filtering refer to Tagging and Filtering.

Figure 4.35 Tagged Entries in the Function Profile



4.6 Message Profile

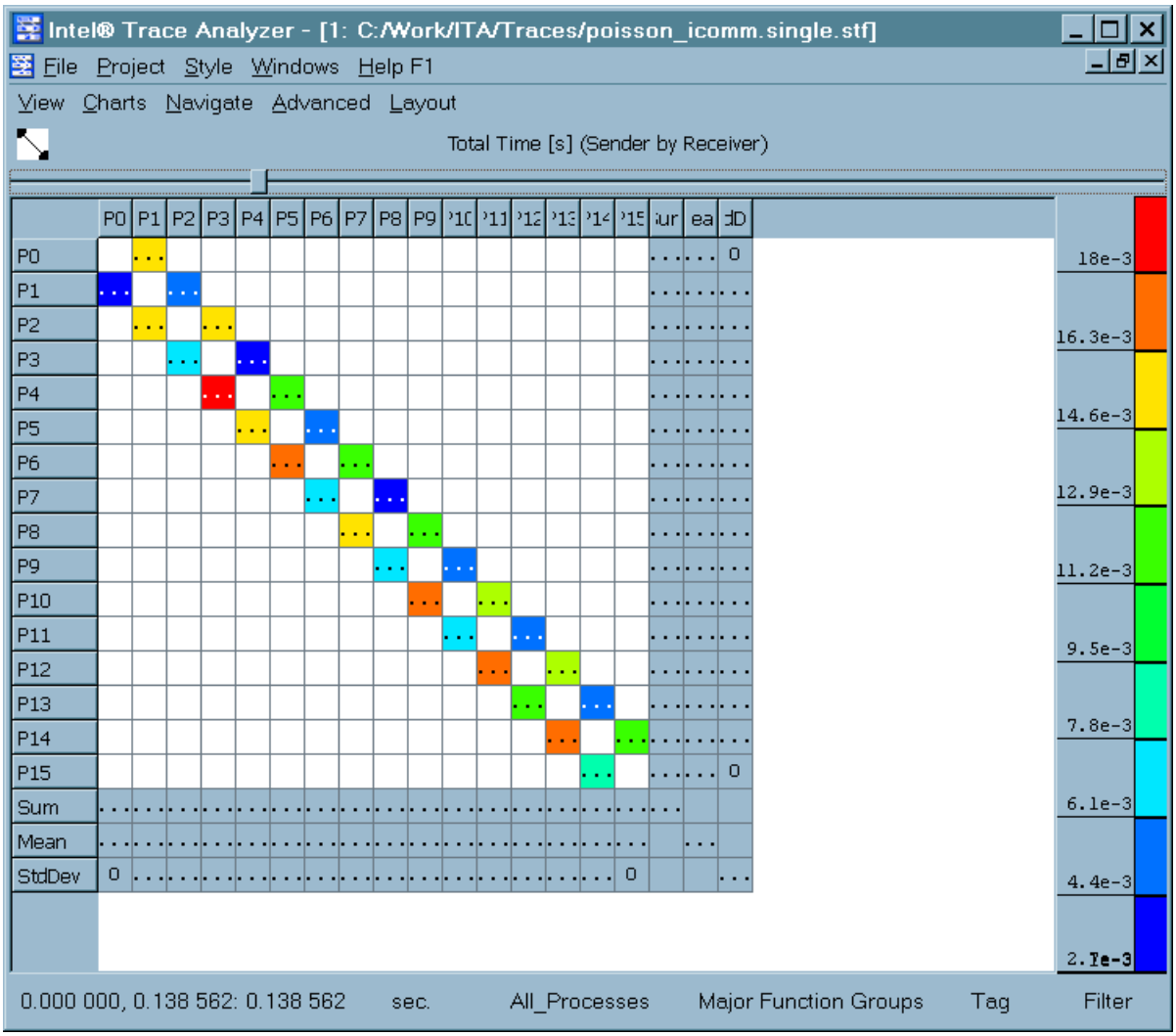
The Message Profile Chart categorizes messages by groupings in a matrix and shows the value of several attributes in each cell.

By default the matrix is square with the sending processes as row labels and the receiving processes as column labels. It shows in cell (i, j) the total time spent in transferring messages from sender i to receiver j.

This chart also includes per row and per column statistics, which give the sum, the average and the standard deviation for the respective row or column.

The grouping that defines the row and column headers of this matrix and therefore the categorization of the data are changed in the context menu and the settings dialog box. Available groupings in addition to Sender and Receiver are for example Tag and Communicator.

Figure 4.36 Message Profile



The attribute shown in the cells is chosen through the context menu or the **Settings** dialog box. Apart from the Total Time shown by default there are other time values, transfer rates, volumes, and counts.

The cell sizes can either be set automatically or manually. If manual sizing is selected you can change the size of the cell by using the slider above the matrix. If the cells are too small to display numeric data, hover your mouse over a cell and view data in the status bar.

The number formatting options are preset globally through the **Number Formatting Settings** dialog (section Number Formatting Settings). To increase the number of digits locally by three (or one) digits press `[+]` (or `[Ctrl]+[+]`). Use `[-]` (or `[Ctrl]+[-]`) to revert this action.

NOTE: The exact effect of asking for additional digits depends on the format chosen in the **Number Formatting Settings** dialog for the respective unit.

You can restrict the display to a rectangular area of the matrix. To select rows or columns click on row or column headers. For an arbitrary area of the matrix keep the mouse button pressed and select the respective cells. To restrict the display to the selection, right-click and select **Zoom to Selection** from the context menu. To change the position of the row and column headers, hold down the `[Ctrl]` key and drag the header to the required position.

4.6.1 Mouse Hover

When the mouse is positioned over any point in the matrix then detailed information for the current position is shown in the View status bar in the form `$AttributeValue ($RowLabel, $ColumnLabel)`. This allows getting exact attribute values even if the cells are configured to be very small or to show no alphanumerical entries at all.

4.6.2 Message Profile Settings

The **Settings** dialog box provides three tabs: **Preferences**, **Colors** and **Data**.

Preferences Tab

The **Display** group provides check boxes and radio buttons to configure some visual details. The check boxes **Row Labels** and **Column Labels** switch the respective row and column headers. The check box **Scale** switches the colored scale next to the matrix. The check box **Grid** shows/removes the black grid shown between cells.

The check box **Keep Empty Rows/Columns when using Sender/Receiver Groupings** switches a special feature on/off. This feature is only relevant for the Groupings Sender and Receiver. For these groupings, a checked state indicates that all processes should always be shown, like for example, showing even empty rows and columns. That keeps the form of the matrix constant and makes it easy to look for patterns in the data. An unchecked state means that empty rows and columns even for these groupings are suppressed. All other groupings suppress empty rows and columns to save screen space regardless of the state of this check box.

The radio buttons **Communicator Names** and **Communicator Ids** allow to either see helpful communicator names (if available in the trace file) that may take a lot of valuable screen space or to restrict the display to show only concise communicator ids.

The **Layout** group allows switching between two fundamentally different modes for the layout of the matrix. By default the mode is **Automatic Cell Sizes** and the cell sizes are adjusted to make all text readable. In this mode checking **Equal Cell Sizes** basically results in equal column widths and enables the check box **Square Cell Sizes** to get square cells. The other options of the **Layout** group are disabled by default.

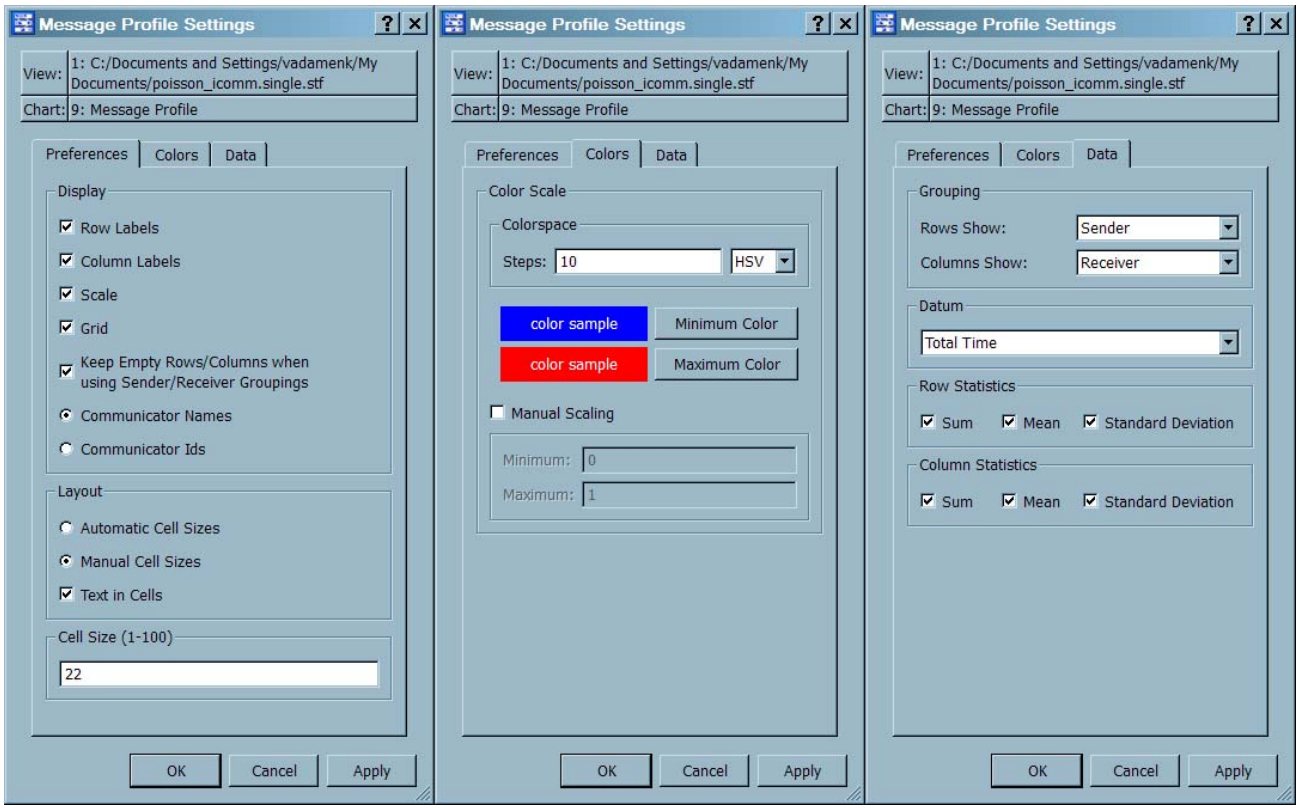
Choosing **Manual Cell Sizes** allows specifying the size of the cells in pixels either in the **Cell Size** group at the bottom of the tab or using the slider that is available on top of the matrix as soon as this setting is applied. In this mode, the alphanumerical data in the cells is displayed only if it fits or if it is switched off entirely by un-checking the check box **Text in Cells**. By default, **Manual Cell Sizes** is checked.

Colors Tab

The push buttons **Maximum Color** and **Minimum Color** allow choosing the colors for the maximum and minimum attribute values. The text input field allows specifying the number of color steps (1-255).

The chosen colors are considered as points in a color space and the colors of the scale are interpolated on a line through color space connecting these two points. The combo box to the right of the text input field allows using either the HSV or the RGB color space. HSV is more fancy and colorful, but RGB is often more useful and readable. For monochrome printing, it is advisable to choose a very light and a very dark color. Choosing white for the minimum and black for the maximum is not at all bad.

Figure 4.37 Three Tabs of the Message Profile Settings Dialog Box



Checking the box **Manual Scaling** enables you to specify the minimum and maximum values for the color scale in the two text input fields below. This is very convenient when comparing two Message Profile Charts that may live in different Views.

Data Tab

The **Grouping** group provides two combo boxes to choose the row and column headers or better said to choose how the data is grouped into categories. The groupings for rows and columns are chosen independently. However not all combinations are possible. It is not possible to have the same grouping for rows and columns and it is not possible to have Sender/Receiver at one axis and any one of Sender or Receiver on the other axis.

The available groupings are:

Sender

Categorizes the messages by Sender. The exact labels are defined by the current thread group that is given by the View (see Views).

Receiver

Categorizes the messages by Receiver. The exact labels are defined by the current process group that is given by the View (see Views).

Sender / Receiver

Categorizes the messages by Sender/Receiver pairs. The exact labels are defined by the current process group that is given by the View (see Views).

Tag

Categorizes the messages by the MPI tag assigned to the message by the program at the sender side.

Communicator

Categorizes the messages by the MPI communicator. The labels are either communicator ids or names. Names are displayed if they are available in the trace file and if they are chosen in the **Preferences** tab of the **Message Profile Settings** dialog box.

Volume

Categorizes the messages by their Volume; for example, size in bytes. This is seen in *Figure* 4.38, where only messages with a volume of 2000 bytes were sent.

Sending Function

Categorizes the messages by the function that sends them. Labels are names of MPI functions such as `MPI_Irsend`. This categorization is not influenced by the current Function Aggregation.

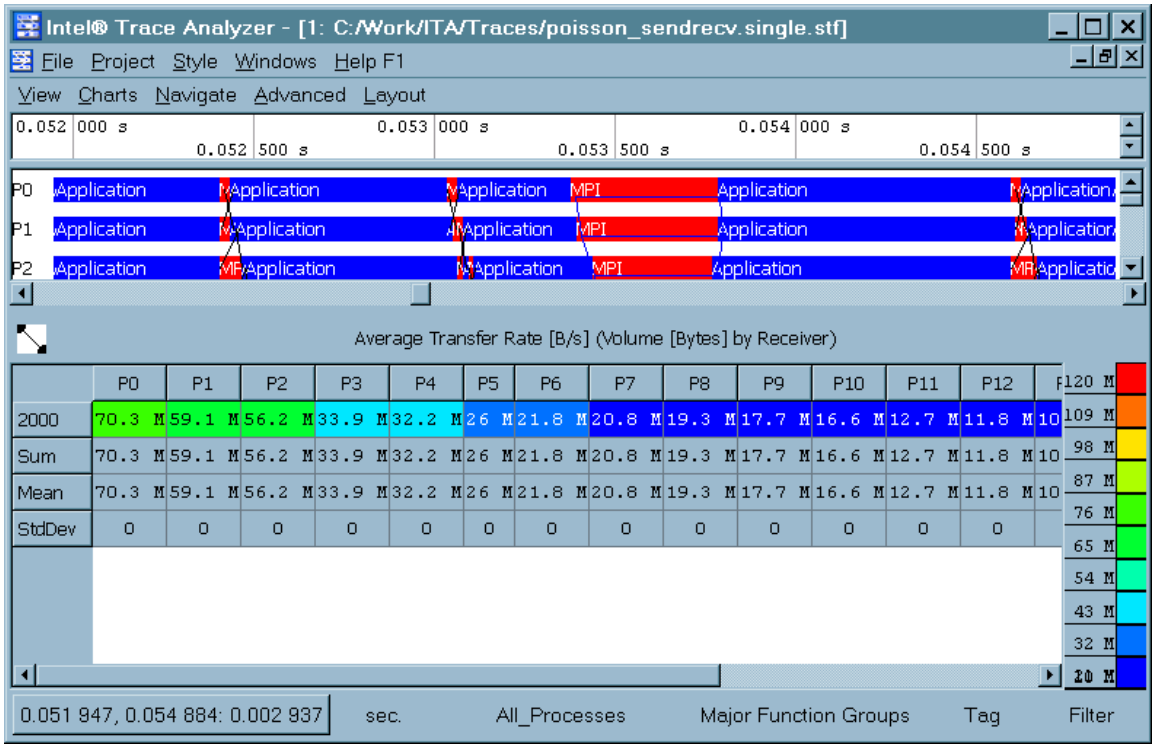
NOTE: This information is only available with traces created by the Intel® Trace Collector Version 6 and higher.

Receiving Function

Categorize the messages by the function that receives them. Labels are names of MPI functions like `MPI_Waitany`. This is not influenced by the current Function Aggregation.

NOTE: This information is only available with traces created by the Intel® Trace Collector Version 6 and higher.

Figure 4.38 Grouping Volume by Receiver



The **Datum** group allows choosing which attribute should be printed or painted in the cells. The available attributes are:

Total Time, [s] or [tick]

The total travel time of the messages, accumulated over all messages that fall into this cell. The unit is either [s] or [tick] depending on the View setting.

Minimum Time, [s] or [tick]

The minimum travel time of a message, minimized over all messages that fall into this cell. The unit is either [s] or [tick] depending on the View setting.

Maximum Time, [s] or [tick]

The maximum travel time of a message, maximized over all messages that fall into this cell. The unit is either [s] or [tick] depending on the View setting.

Average Transfer Rate, [B/s]

The average transfer rate, averaged over the transfer rates of all messages that fall into this cell. Messages are not weighted; for example, transfer rates of short messages have the same impact as transfer rates of long messages.

Minimum Transfer Rate, [B/s]

The minimum transfer rate, minimized over all messages that fall into this cell.

Maximum Transfer Rate, [B/s]

The maximum transfer rate, maximized over all messages that fall into this cell.

Total Data Volume, [B]

The total data volume, accumulated over all messages that fall into this cell.

Minimum Data Volume, [B]

The minimum data volume, minimized over all messages that fall into this cell.

Maximum Data Volume, [B]

The maximum data volume, maximized over all messages that fall into this cell.

Count, [1]

The number of messages that fall into this cell.

The group **Row Statistics** allows switching the individual columns on or off. These columns hold the statistics for the rows.

The group **Column Statistics** allows switching the individual rows on or off. These hold the statistics for the columns.

4.6.3 Context Menu

The context menu provides shortcuts with which the attributes and groupings are selected. To do this, use entries **Attribute to show**, **Columns show** and **Rows show**. These entries are the same as those explained in the section Message Profile Settings (**Message Profile Settings** dialog box).

The entry **Sort** allows to sort rows by the values of the column clicked on, or to sort columns by the values in a row clicked on and to switch back to the default order. Switching back to the default order is also useful if the columns or rows were rearranged by dragging the row or column headers around (hold the `[Ctrl]` key down while dragging to do that).

When a given area of the matrix is selected, then the context menu provides entries to either zoom into/out of the selected area or to suppress the display of the selected area (all row and columns that are partially selected are suppressed).

Figure 4.39 Selecting an Area in the Matrix and Zooming into

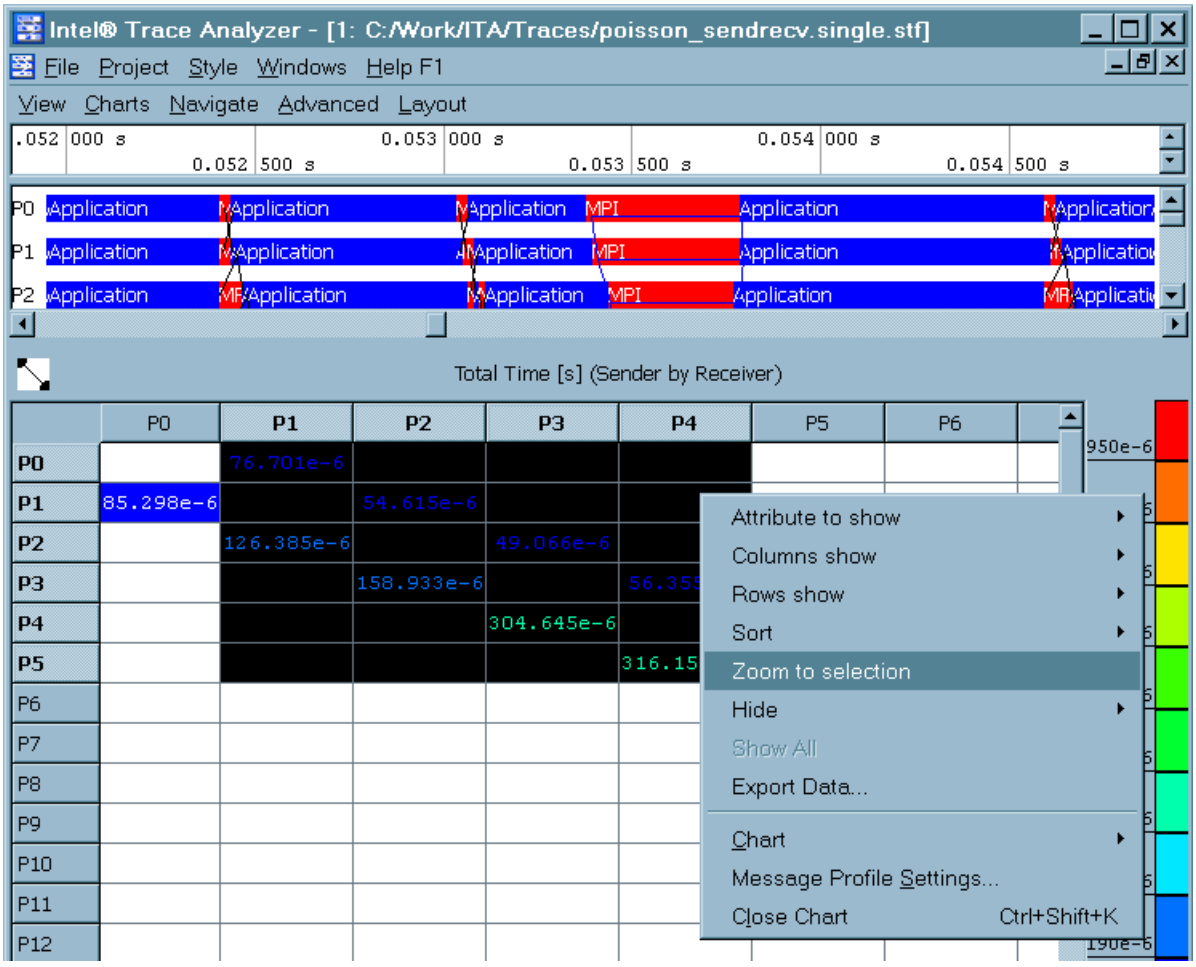
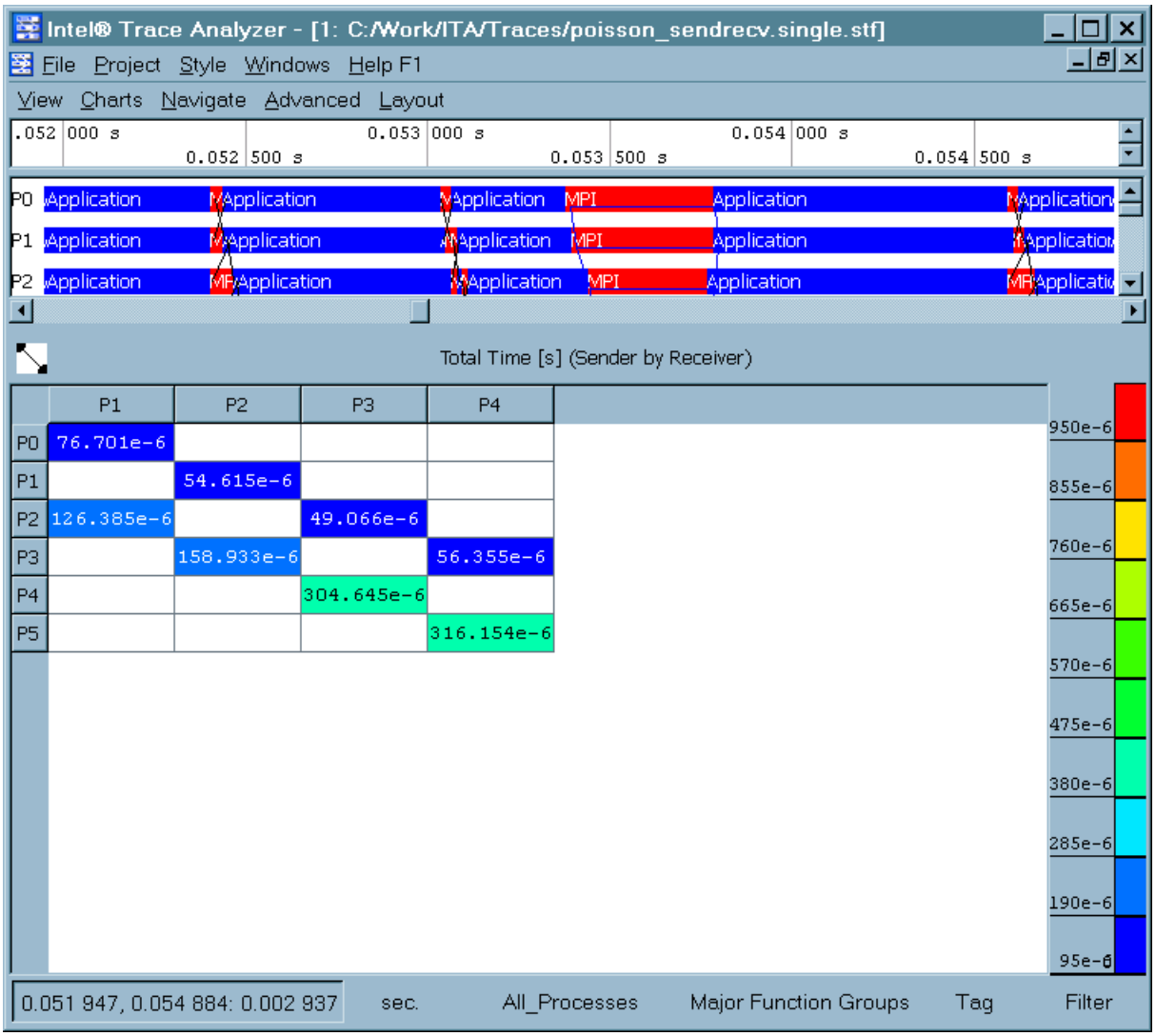


Figure 4.40 Zoomed into the Selected Area



If something is hidden then the context menu provides an entry **Show All** and the **Hide** submenu contains enabled entries to unhide all hidden rows or columns or all.

Actually the zoom feature of the Message Profile relies on storing the row and column labels to be suppressed. This can have surprising effects: if **Volume** is selected as row grouping and the rows with labels 17 and 19 are hidden, then scrolling into an area containing messages with volume 18 results in these messages being shown. To suppress all messages with certain volumes, use filtering (see Filtering Dialog).

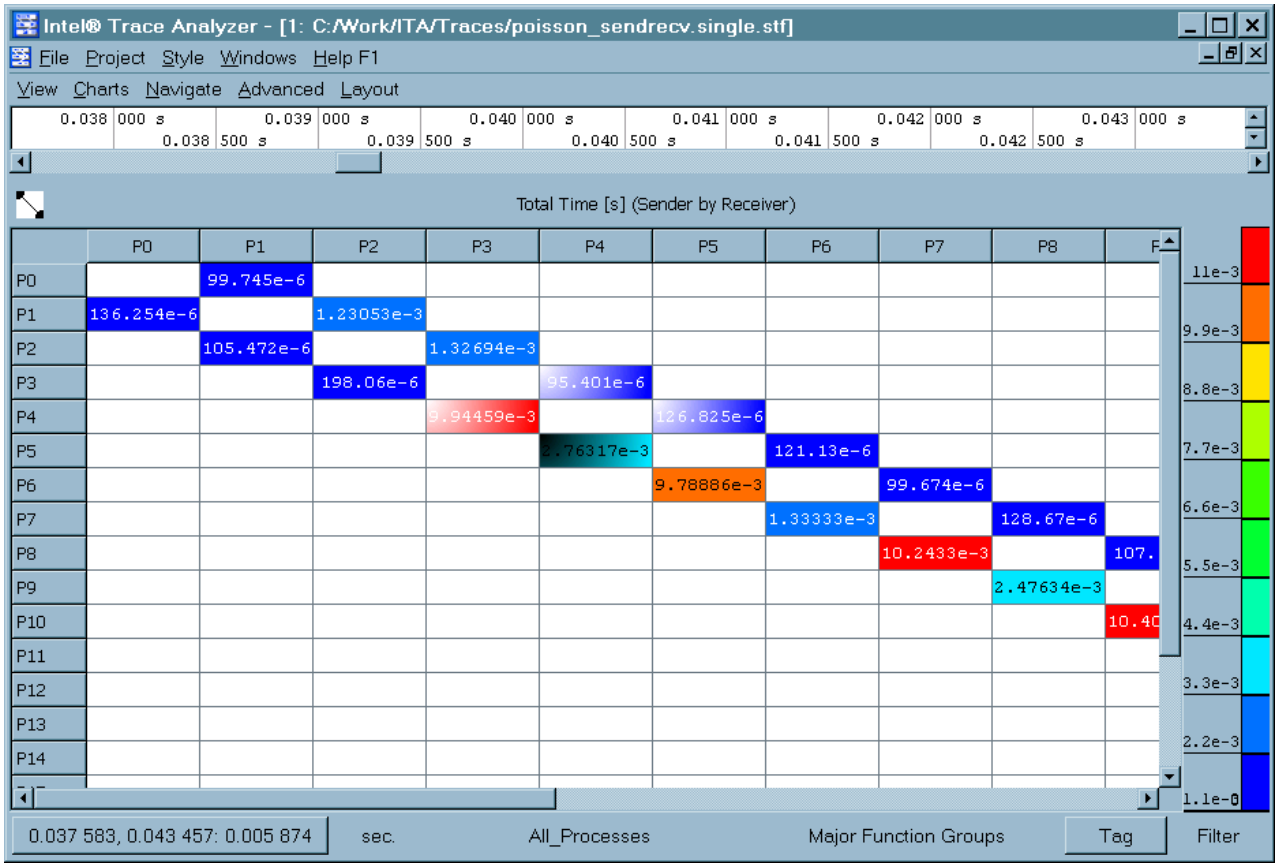
The entry **Export Data** opens a **File Save** dialog box to select a file to store textual data in. This includes all data cells that contain at least one message, even if they are currently hidden. It does not contain row or column statistics. For each cell, it stores all available attributes.

Additionally the context menu contains the usual operations as described in the section Common Chart Features.

4.6.4 Filtering and Tagging

Tagged cells are emphasized by a small additional frame around the cell in the color of the alphanumerical entry in the cell. A cell is tagged as soon as a single tagged message exists in that cell.

Figure 4.41 Tagging a Process in the Message Profile



Messages that do not pass a filter are not accounted for and may result in a smaller matrix when this occurs in empty rows and columns. For more information on filtering and tagging, refer to Tagging and Filtering.

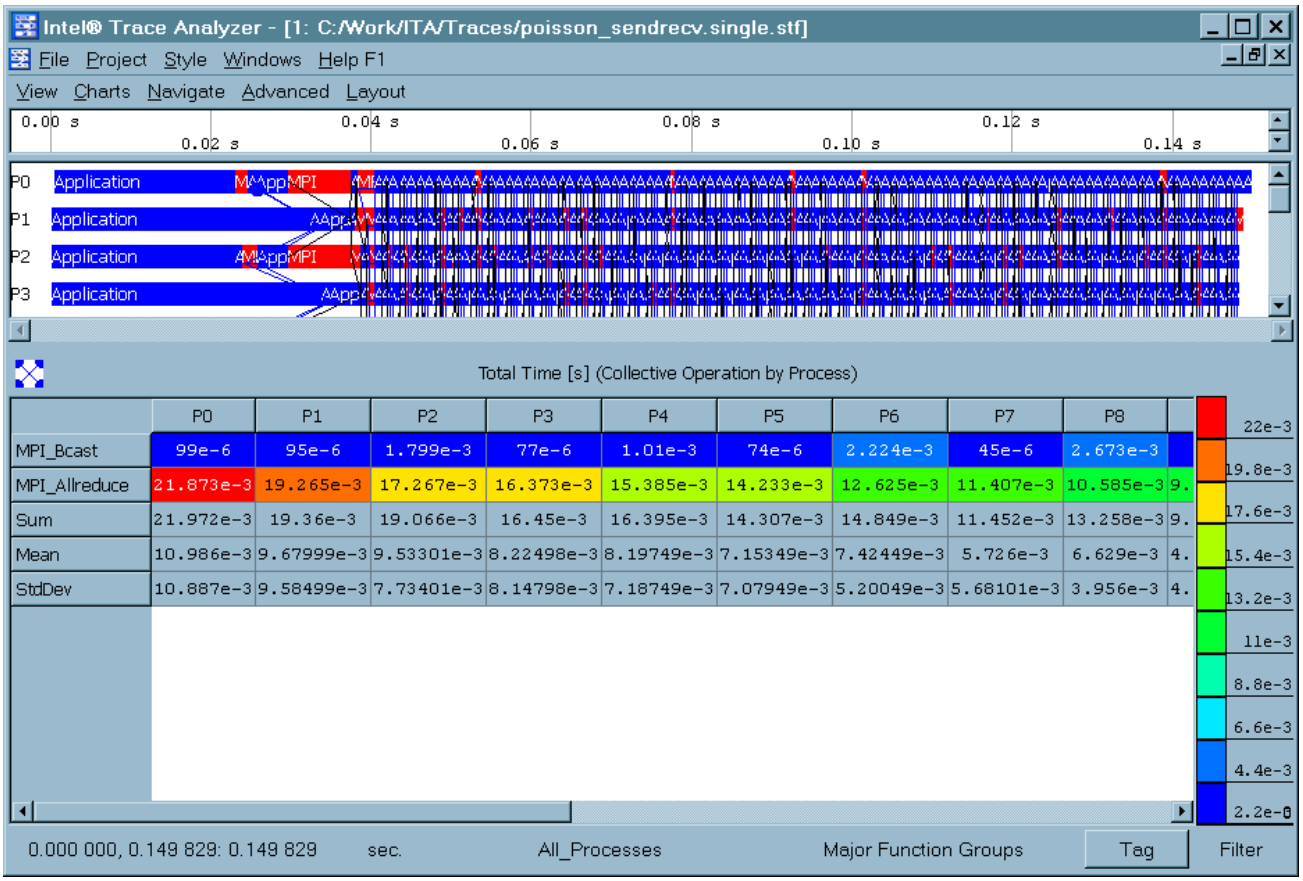
4.6.5 Aggregation

The View thread group influences the labels of the Sender, Receiver and Sender/Receiver groups. The View function group has no influence. If the View shows the thread group Other, then this results in additional rows and columns for the groupings Sender, Receiver and Sender/Receiver.

4.7 Collective Operations Profile

The Collective Operations Profile enables analyzing communication patterns that are done using MPI Collective Operations. Like the Message Profile (see Message Profile), the Collective Operations are also represented in a color-coded matrix format. The default matrix shows the type of the Collective Operation as the row label and the process as the column label.

Figure 4.42 Collective Operations Profile



The precision of the values shown can be adjusted as explained in the section Message Profile.

4.7.1 Mouse Hover

When the mouse is positioned over any point in the matrix then detailed information for the current cell is shown in the View status bar in the form `$AttributeValue ($RowLabel, $ColumnLabel)`. This allows getting exact attribute values even if the cells are configured to be very small or to show no alphanumerical entries at all.

4.7.2 Collective Operations Profile Settings

The **Collective Operations Settings** dialog adjusts the various attributes that affect how the Chart is displayed. This includes the colors, the layout and the statistical attributes. The **Settings** dialog box is divided into three tabs namely the **Preferences** tab, the **Colors** tab and the **Data** tab.

Preferences Tab

The **Preferences** tab adjusts the **Display** settings and the **Layout** settings.

Display Group: In this group, the visual aspects of the Chart are configured. Using the check boxes **Row Labels** and **Column Labels**, it is decided if the row/column headers should be displayed or not. The check box **Scale**, if enabled, displays the colored scale that is seen on the right-hand side of the matrix. The **Grid** checkbox displays/removes the black grid in which the cells are placed. The checkbox **Keep Empty Rows/Columns when using Sender/Receiver Groupings** switches a special feature that is only relevant for the Groupings Sender and Receiver. For these groupings, a checked state of this

box indicates that all processes should always be shown, like for example, showing even empty rows and columns. That keeps the form of the matrix constant and makes it easy to look for patterns in the data. An unchecked state means that empty rows and columns even for these groupings are suppressed. All other groupings suppress empty rows and columns to save screen space regardless of the state of this check box.

The radio buttons **Communicator Names** and **Communicator Ids** allow to either see helpful communicator names (if available in the trace file) that may take a lot of valuable screen space or to restrict the display to show only concise communicator ids.

Layout Group: This group allows switching between two fundamentally different modes for the layout of the matrix. In the **Automatic Cell Sizes** mode, checking **Equal Cell Sizes** basically results in equal column widths and enables the check box **Square Cell Sizes** to get square cells. The other options of the **Layout** group are disabled by default.

Choosing **Manual Cell Sizes** allows specifying the size of the cells in pixels. This is done either in the **Cell Size** group at the bottom of the tab or by sizing the cells manually with the slider that is available on top of the matrix as soon as this setting is applied. In this mode, the alphanumerical data in the cells is displayed only if it fits. Otherwise, it is switched off entirely by unchecking the check box **Text in Cells**.

Colors Tab

The push buttons **Maximum Color** and **Minimum Color** allow choosing the colors for the maximum and minimum attribute values. The text input field allows specifying the number of color steps (1-255).

The chosen colors are considered as points in a color space and the colors of the scale are interpolated on a line through color space connecting these two points. The combo box to the right of the text input field allows using either the HSV or the RGB color space. HSV is fancier and colorful, but RGB is often more useful and readable. For monochrome printing, it is advisable to choose a very light and a very dark color. Choosing white for the minimum and black for the maximum is not at all bad.

By checking the box **Manual Scaling** it is possible to specify the minimum and maximum values for the color scale in the two text input fields below.

Data Tab

The **Data** tab allows choosing how the data is analyzed. The **Data** tab is divided into the **Grouping** section, the **Datum** section, the **Row Statistics** and the **Column Statistics**.

Grouping section provides two combo boxes to choose the row and column headers or better said to choose how the data is grouped into categories. The groupings for rows and columns are chosen independently. However not all combinations are possible. It cannot have the same header for row and column. For example, a matrix cannot be plotted with both the row and column header being Communicator. All the headers are explained below:

Communicator: Categorizes the messages by the MPI communicator. The labels are either communicator ids or names. Names are displayed if they are available in the trace file and if they are chosen in the **Preferences** tab (see above) of the **Settings** dialog box.

Collective Operation: This shows the types of operations like `MPI_Allreduce` and `MPI_Bcast`.

Root: Shows the root used in the operation, if applicable. If there is no root, a label No Root is created.

Process: Categorizes the operations by the processes.

Datum Section: The Datum group allows choosing which attribute should be printed or painted in the cells. The available attributes are:

Total Time: The total time spent in operations, accumulated over all operations and all processes referred in this cell. For a single process and a single operation this is the time spent in the call to the operation. For cells referring to a process group this is the sum of the times all contained processes did spent in the operation. For many operations it is the sum over the times spent in each single operation. The unit can either be seconds [s] or ticks [tick] depending on the View setting.

Minimum Time: The minimum time spent in an operation, minimized over all operations and all processes that fall into this cell ([s] or [tick]).

Maximum Time: The maximum time spent in an operation, maximized over all operations and all processes that fall into this cell ([s] or [tick]).

Total Volume Sent: The total data volume that has been sent from all operations in this cell [bytes].

Minimum Volume Sent: The minimum amount of data volume that has been sent by an operation in this cell [bytes].

Maximum Volume Sent: The maximum amount of data volume that has been sent by an operation in this cell [bytes].

Total Volume Received: The total data volume that has been received by all operations in this cell [bytes].

Minimum Volume Received: The minimum amount of data volume that has been sent by an operation in this cell [bytes].

Maximum Volume Received: The maximum amount of data volume that has been received by an operation in this cell [bytes].

Total Data Volume: The total data volume, accumulated over all operations in this cell.

Count: The number of operations in this cell.

Row Statistics: it specifies whether the statistical values like the sum, the mean or the standard deviation should be displayed for the rows. Similarly, Column Statistics give the above mentioned statistical values for the given columns.

4.7.3 Context Menu

The context menu in the Collective Operations Profile mainly consists of the following entries:

Attribute to show

The attributes to be shown in the Collective Operations Profile is selected in this option. It contains all the attributes that are explained in the **Datum** Section.

Columns to show

This entry indicates if the Collective Operations Profile should be displayed by process, by root or by communicator.

Rows to show

This entry denotes whether the rows of the profile show the Collective Operation, the communicator or the root values.

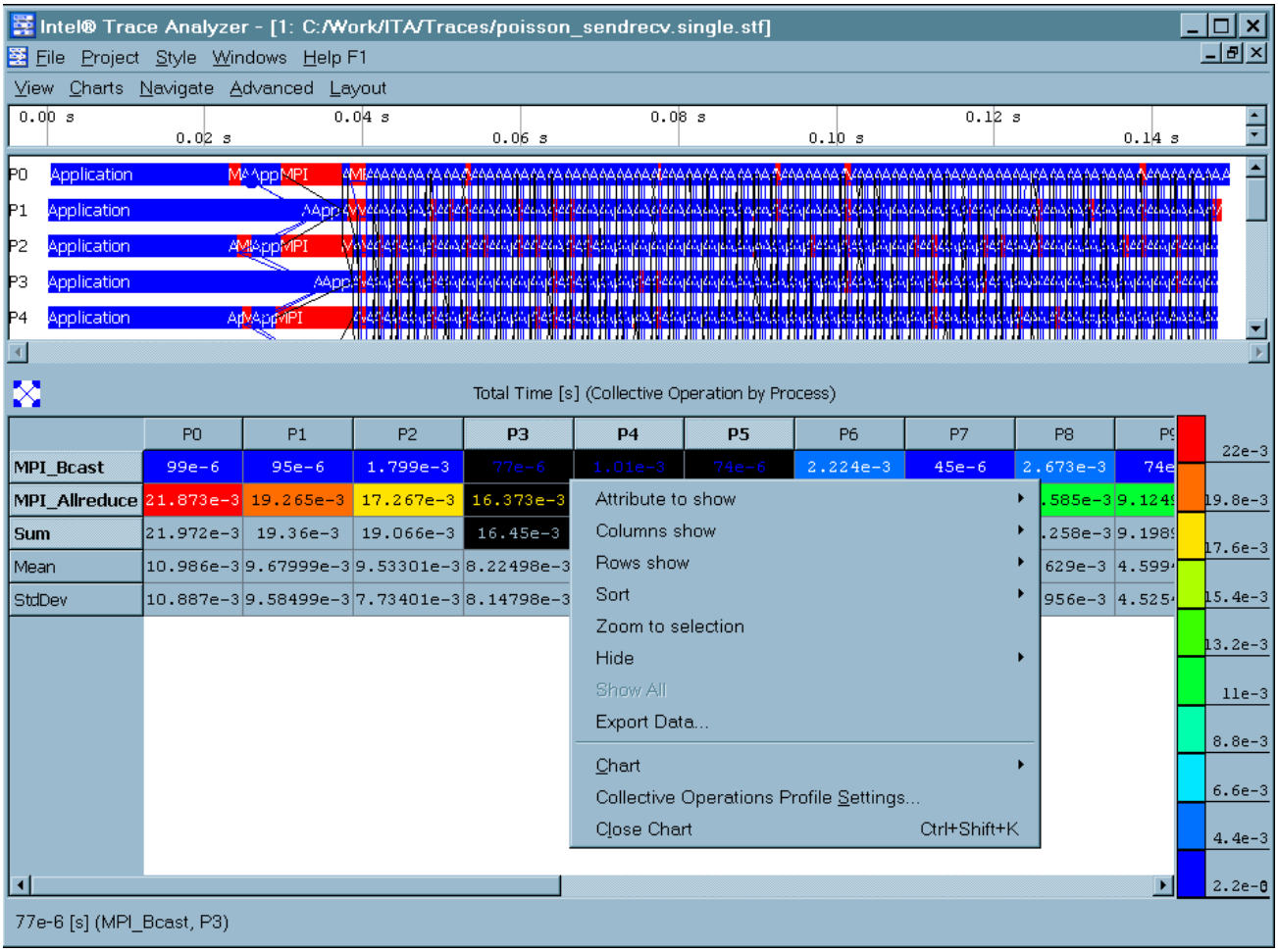
Sort

The entry sort enables sorting rows by the values of the column clicked on, or to sort columns by the values in a row clicked on and to switch back to the default order. Switching back to the default order is also useful if the columns or rows were rearranged by dragging the row or column headers around (hold the [Ctrl] key down while dragging).

Zoom to selection

Use this entry to focus on a particular region in the matrix. To do this, select the required region with the mouse as shown in *Figure* 4.43 and choose the entry **Zoom to selection** from the context menu (obtained by a right-click on the mouse). As a result, everything else other than the selected region, is removed from the display.

Figure 4.43 Zoom to Selection in the Collective Operations Profile



Hide

This hides all cells that are selected. Selection of cells is done by holding down the left mouse button and moving over the required region. This also automatically opens the context menu.

Show All

This entry shows all cells again. It is enabled only if cells have been previously hidden using the **Hide** entry.

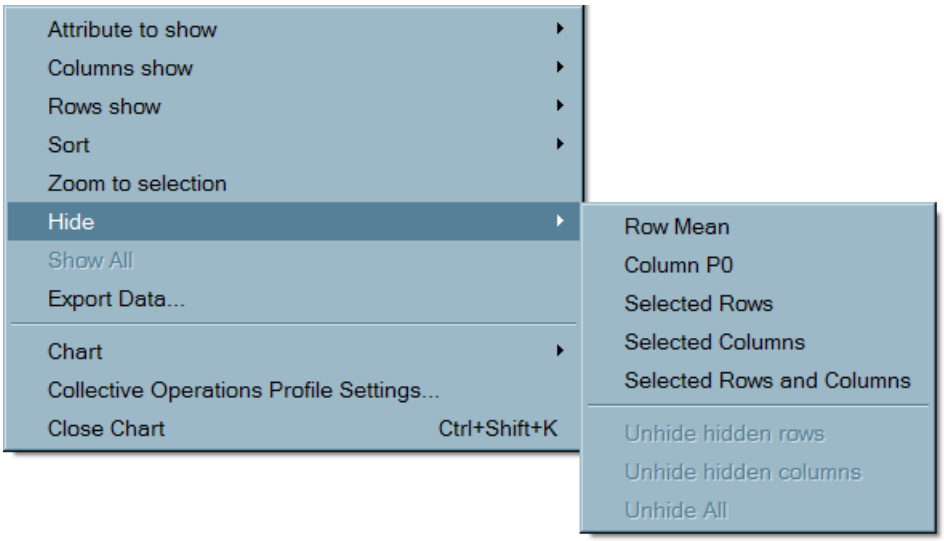
Export Data

This entry opens a **File Save** dialog box to select a file to store textual data in. This includes all data cells that contain at least one message, even if they are currently hidden. For each cell, all available attributes are given. It does not contain row or column statistics.

Collective Operations Settings Profile

This opens the **Settings** dialog box of the Collective Operations Profile.

Figure 4.44 Context Menu of the Collective Operations Profile



4.7.4 Filtering and Tagging

Tagged cells are emphasized by a small additional frame around the cell in the color of the alphanumerical entry in the cell. A cell is tagged as soon as a single tagged message falls into that cell. For more information on tagging and filtering, refer to Tagging and Filtering.

4.8 Common Chart Features

All Charts share some common features that are available through common context menu entries:

Print Chart (`Ctrl+Shift+P`)

Prints a hard copy of the Chart.

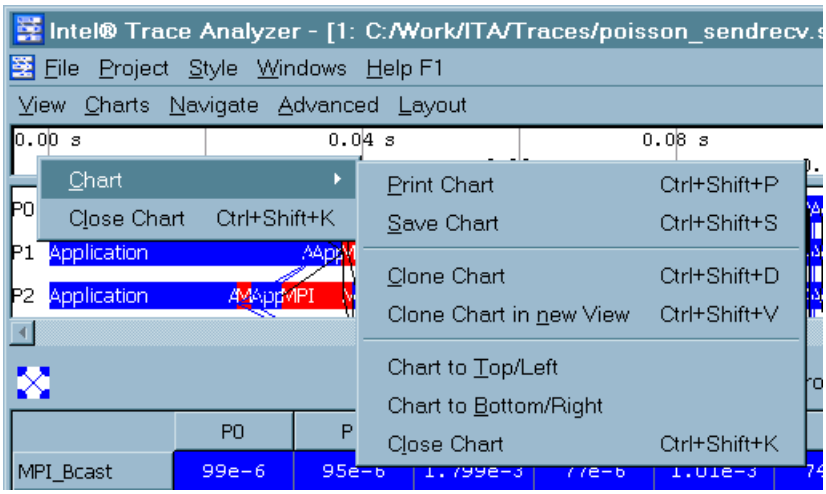
Save Chart (`Ctrl+Shift+S`)

Saves the chart as a picture.

Clone Chart (`Ctrl+Shift+D`)

Opens exactly the same Chart within the present View. In other words, it clones the current Chart.

Figure 4.45 Common Context Menu Features



Clone Chart in New View (`Ctrl+Shift+V`)

Creates a new View and opens a copy of the current Chart in the new View.

Chart to Top / Left

Useful when two or more Charts of the same kind (Timelines or Profiles) are open. If the Charts are placed one on top of each other, then this entry moves the selected Chart to the top. If they are placed next to each other, then it moves the selected Chart to the left.

Chart to Bottom / Right

This entry is also only useful when two or more Charts of the same kind (Timelines or Profiles) are open. If the Charts are placed one on top of each other, then this entry moves the selected Chart to the bottom. If they are placed next to each other, then it moves the selected Chart to the right.

Close Chart (`Ctrl+Shift+K`)

Closes the Chart.

When a Function Group A is right-clicked then all Charts show the entry **Context Menu > Ungroup A**. When a child of a recently ungrouped Function Group A is right-clicked then all Charts show the entry **Context Menu > Regroup A**.

To move the Chart to different positions in the View, use **Context Menu > Chart > Chart to Top / Left** or **Context Menu > Chart > Chart to Bottom / Right**.

5 Dialogs

Apart from the Settings dialog boxes of each chart, there are a number of other dialog boxes in the Intel® Trace Analyzer. All dialog boxes have the same semantics regarding the buttons **OK**, **Cancel** and **Apply**.

In case the current settings of the dialog boxes are inconsistent or out of bounds, the OK and Apply buttons are both disabled.

5.1 Filtering Dialog

The filtering dialog box is accessed through the **Advanced** Menu (**Views Menu > Advanced > Filtering**). This dialog box allows specifying filter expressions that describe which function events, messages and collective operations are to be analyzed and shown. The two radio buttons in the group **Definition of Filter Expression** at the top switch between two fundamental modes: **Using GUI Interface** is chosen by default and allows generating the filter expression through a graphical interface, while **Manually** allows to type in the filter expression directly. Building the expression with the graphical interface is a lot easier and recommended for beginners or infrequent use.

An expression is built using either the point and click interface or using the manual mode. Either way the resulting expression is parsed upon each change.

If the current expression can not be converted into a proper filter definition then the dialog shows a red warning at the bottom which indicates the reason.

If the expression makes use of filter attributes that require to bypass the inmemory trace cache and to process detailed data from the trace file then a warning message is shown. In this case it can be expected that the analysis using this filter expression will need considerably more time than usual.

5.1.1 Building Filter Expressions Using the Graphical Interface

The **Filtering** dialog box allows specifying filter expressions separately for function events, messages and collective operations through three separate tabs. It provides a fourth tab labeled **Processes** to additionally restrict the events to only a subset of processes.

Some of the text fields in the filter dialog box can take triplets that are of the form `start:stop:incr` or the short form `start:stop` if the increment is one. Such a triplet describes all numbers greater than or equal to start and smaller than or equal to stop and that are of the form `start+incr*n`. The stop value is optional, too. When stop is omitted, all numbers, beginning from start, match.

Processes Tab

This tab specifies the processes that may pass the filter. It has effects on all three sub-expressions.

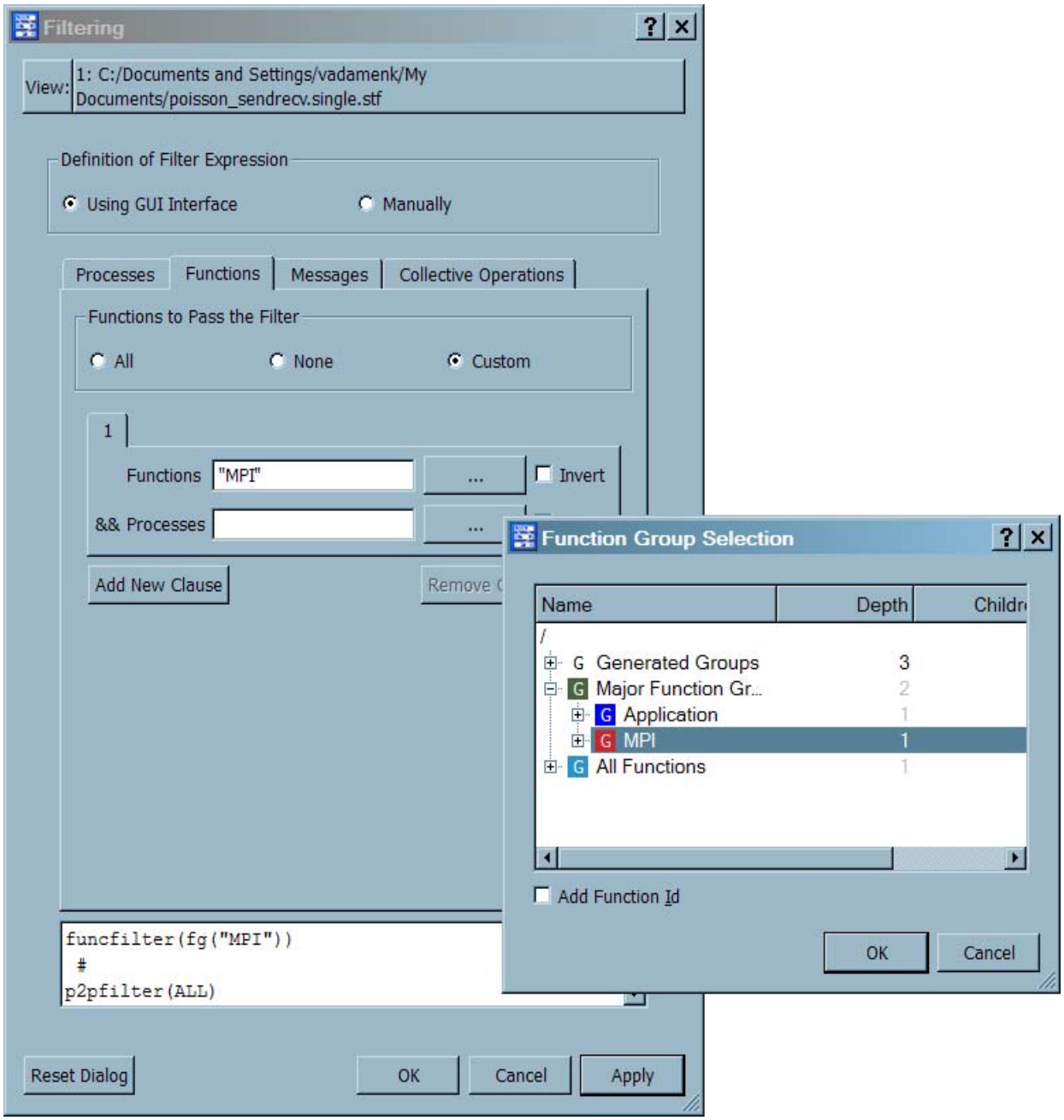
The **Show Processes** text field defines the processes that are filtered according to the settings of the other three tabs. Processes not listed in the text field are filtered out completely unless the text field is empty: this is the default and indicates that the **Processes** tab has no effect. The text field can take a comma separated list of process ids, process group ids, triplets thereof, unquoted or double-quoted names of threads, processes or process groups. Names given match all equally named processes/groups.

The button labeled **...** opens the dialog box **Process Group Selection** where one or more processes or process groups are selected (see below). The **Invert** check box swaps the selection. For example, if all processes except one pass the filter, select the process to be filtered out and use the check box **Invert**.

Checking **Invert** means to let only events pass that do not match this predicate. It is a logical NOT and shown as an exclamation mark before the predicate in the filter expression.

Below the filter clauses, the resulting filter expression is shown. To reuse the expression elsewhere, select and copy the expression in the manual mode of the filtering dialog box. The selection is done using the mouse. The context menu of the filter expression contains a Copy (`Ctrl+C`) entry and a Select All (`Ctrl+A`) entry.

Figure 5.1 Function Group Selection Opened through the Filter Dialog Box



Functions Tab

In the **Functions** tab, the mode of filtering is selected by means of radio buttons: **All**, **None** and **Custom**.

Again, on selecting the **Custom** radio button, the filter clause tab is enabled. This consists of the following entries:

Functions: The text field can take a comma-separated list of functions or function group ids, triplets thereof, unquoted or double-quoted names of functions or function groups that describe functions passing the filter. Names given match all equally named functions/groups. The button **...** opens the **Function Group Selection** subdialog which allows choosing the function names from a list. In addition, the subdialog provides a checkbox **Add Function Id**: if checked, not only is the name of a selected function written into the text field, but also its Id. This is useful to resolve ambiguities of function/group names: if only the name is written into the text field, all functions with this name pass the filter; if the name is replaced by an id, only the function with this id passes the filter, not other functions with the same name.

Processes: the text field can take a comma-separated list of process ids, process group ids, triplets thereof, unquoted or double-quoted thread, process or process group names that describe processes and threads in the functions passing the filter. Names given match all equally named processes/groups. The button **...** allows choosing from a list. The corresponding **Process Group Selection** subdialog provides a checkbox **Add Process Id** which is similar to the above **Function Group Selection** subdialog.

The button **Add New Clause** specifies another filter clause. To remove an existing filter clause tab, use the Remove Current Clause button. Clauses are connected by a logical OR, while attributes from the same tab are connected by a logical AND; they form a so-called And Clause.

Messages

On selecting the **Custom** radio button (see *Figure* 5.1), the filter clause tab in the **Messages** tab is enabled. It has the following entries:

Communicator: The text field can take a comma-separated list of communicator ids, unquoted communicator names or communicator names in double quotes that pass the filter. The button **...** allows choosing from a list.

Tag: The text field can take a comma-separated list of non-negative integers and triplets that describe tag values that pass the filter.

Message Size: The text field can take a comma-separated list of non-negative integers and triplets that describe message sizes (in bytes) that pass the filter.

Sender: The text field can take a comma-separated list of process ids, process group ids, triplets thereof, unquoted or double-quoted names of threads, processes or process groups that describe processes and threads in the message sender that make the message pass the filter. Names given match all equally named processes/groups. The button **...** allows choosing from a list.

Receiver: Analogous to **Sender**.

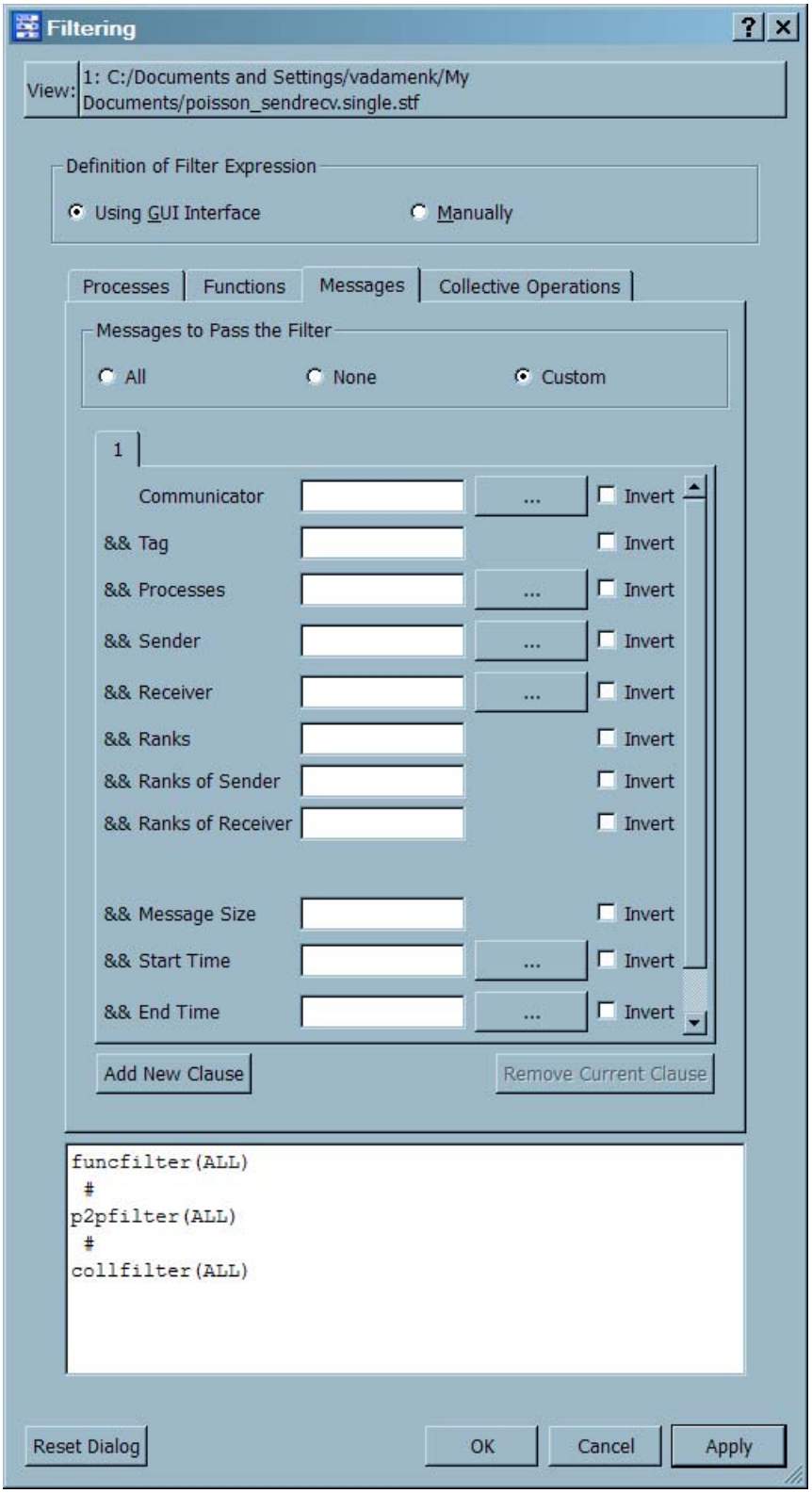
Processes: Makes a message pass the filter if either the sender or the receiver matches. Analogous to the logical OR of **Sender** and **Receiver**.

Ranks of Sender: The text field can take a comma-separated list of nonnegative integers and triplets that describe sender ranks (in the MPI communicator) that can pass the filter.

Ranks of Receiver: Analogous to **Rank of Sender**.

Ranks: Makes a message pass if either the sender or the receiver matches. Analogous to the logical OR of **Rank of Sender** and **Rank of Receiver**.

Figure 5.2 Filtering Dialog Box Showing the Messages Tab



Sending Function: The text field can take a comma-separated list of functions or function group ids, triplets thereof, unquoted or double-quoted names of functions or function groups that describe functions from which the message was sent. Names given match all equally named functions/groups. The button **...** allows choosing from a list.

Receiving Function: The text field can take a comma-separated list of functions or function group ids, triplets thereof, unquoted or double-quoted names of functions or function groups that describe functions which received the sent message. Names given match all equally named functions/groups. The button **...** allows choosing from a list.

Start Time: The text field can take a comma-separated list of non-negative integers and triplets that describe start time (in ticks) of the message that make the operation pass the filter. The button **...** allows to enter/edit the time in ticks or seconds (default depending on the View current time unit).

End Time: Analogous to **Start Time**.

Duration: The text field can take a comma-separated list of non-negative integers and triplets that describe the duration (in ticks) of the message that make the operation pass the filter. The button **...** allows to enter/edit the duration (shown as a time interval) in ticks or seconds (default depending on the View current time unit).

Collective Operations

On selecting the **Custom** radio button the filter clause tab in the **Messages** tab is enabled. It has the following entries:

Communicator: The text field can take a comma-separated list of communicator ids, unquoted communicator names or communicator names in double quotes that pass the filter. The button **...** allows choosing from a list.

Collective Operation: The text field can take a comma-separated list of unquoted or double-quoted names of collective operations like `MPI_Allreduce` that pass the filter. The button **...** allows choosing from a list.

Transferred Volume: The text field can take a comma-separated list of nonnegative integers and triplets that describe all volumes (in total bytes per operation) that make a collective operation make pass the filter.

Processes: The text field can take a comma-separated list of process ids, process group ids, triplets thereof, unquoted or double-quoted names of threads, processes or process groups that describe processes and threads participating in the operation that make the operation pass the filter. Names given match all equally named processes/groups. The button **...** allows choosing from a list.

Root: The text field can take a comma-separated list of process ids, process group ids, triplets thereof, unquoted or double-quoted names of threads, processes or process groups that describe processes and threads serving as Root in the operation that make the operation pass the filter. Names given match all equally named processes/groups. The button **...** allows choosing from a list.

Rank of Root: The text field can take a comma-separated list of non-negative integers and triplets that describe root ranks of the operation that make the operation pass the filter.

Start Time: The text field can take a comma-separated list of non-negative integers and triplets that describe start time (in ticks) of the operation that make the operation pass the filter. The button **...** allows to enter/edit the time in ticks or seconds (default depending on the View current time unit).

End Time: Analogous to **Start Time**.

Duration: The text field can take a comma-separated list of non-negative integers and triplets that describe the duration (in ticks) of the operation that make the operation pass the filter. The button **...** allows to enter/edit the duration (shown as a time interval) in ticks or seconds (default depending on the View current time unit).

5.1.2 Building Filter Expressions Manually

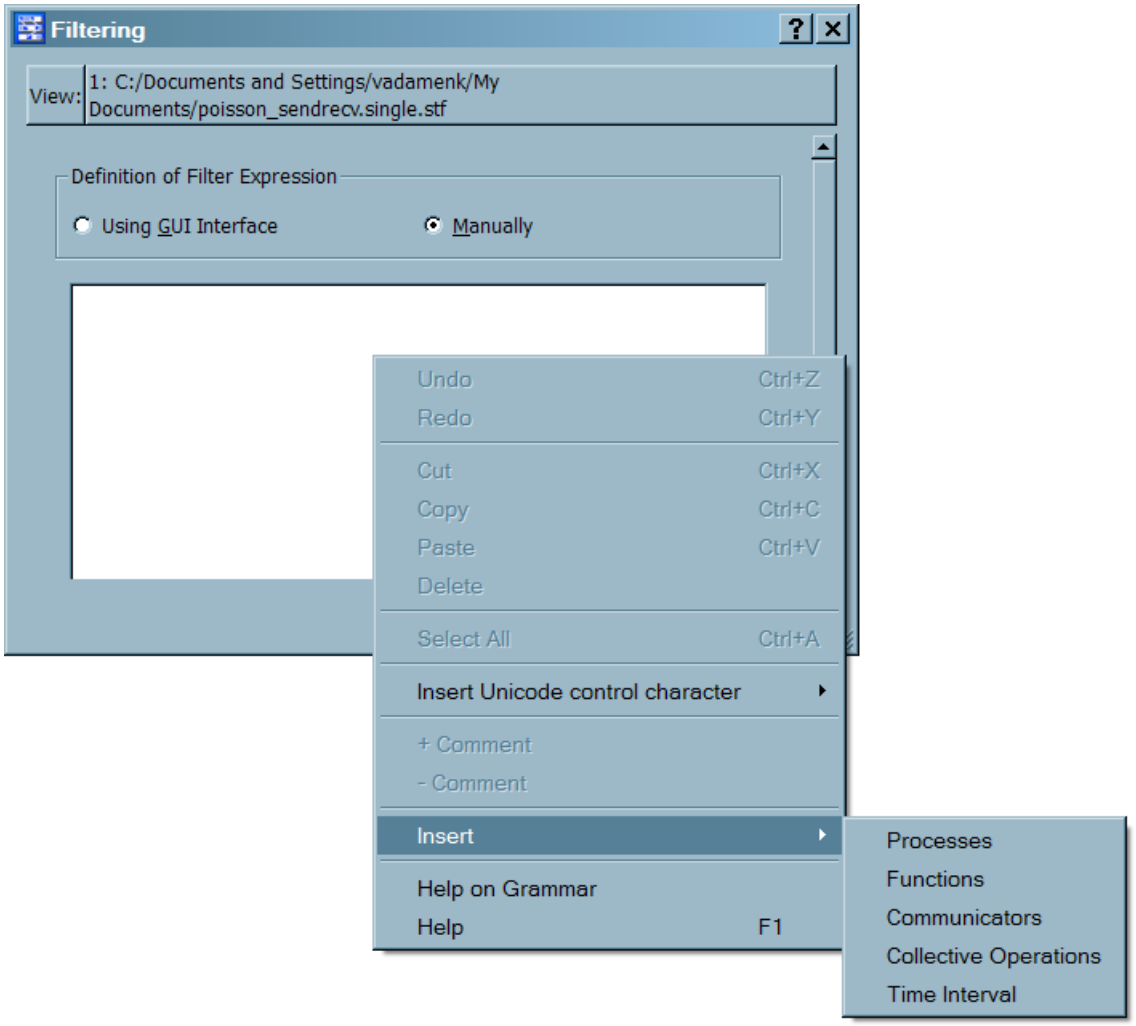
Manual mode allows constructing any filter expression that is valid as described by the expression grammar in the section Filter Expression Grammar.

For convenience, there are context menu entries that allow to select processes, functions, communicators and collective operations from a dialog box in the same way as from the point and click interface and to insert them into the expression at the current cursor position.

The percentage sign **%** inserts single line comments and there are context menu entries to comment out (or in) selected text blocks.

There is no operator precedence in the Intel® Trace Analyzer; the expressions are evaluated from left to right. However, you can use parentheses if needed.

Figure 5.3 Filtering dialog box in manual mode showing its context menu



5.1.3 Filter Expression Grammar

The filter expression grammar creates a filter, which a set of function (`funcfilter`), message (`p2pfilter`), and collective operation (`collfilter`) filters, each defining a filter for its respective kind of data. These sub-filter specifications are separated by a `#` and come in any order. Each filter class is specified once, more than once (in which case a Boolean AND is created from all subfilters for a given class), or not specified at all. An example where three classes of filters are specified is the expression generated by the graphical interface.

Each filter class specifier (`funcfilter`, `p2pfilter`, or `collfilter`) is followed by an expression put in parentheses. That expression can consist of any number of predicates that are different for each filter class and correspond to the entries described in the section Building Filter Expressions Using the Graphical Interface. (See also Formal Description of the Grammar) These predicates are joined by using Boolean AND (`&&`) and OR (`||`) operators. Boolean expressions are parenthesized as needed. Also, a Boolean NOT (`!`) operator in front of any predicate or parenthesized expression negates the predicate/expression.

A filter class allows defining a special expression for a filter that lets all or no data of that kind pass through it. For example, `p2pfilter(NONE)` filters out all messages, while `collfilter(ALL)` lets all collective operations pass. When the keywords `ALL` or `NONE` are used, ensure that it is the only argument to `funcfilter`, `p2pfilter`, or `collfilter`.

Keyword specification in the filter expression grammar is case-insensitive. Specifying names (for functions, processes and communicators, amongst others), however, is case-sensitive. Double quotes are needed for names that consist of several words or do not start with a letter or an underscore character (for example, Major Function Groups). Use double quotes for single word names (for example, MPI) if necessary. White space (space and tab characters, as well as newlines) is ignored, unless it is part of a quoted name. If a process/group or function/group name is ambiguous then it is evaluated as if all matching groups were given.

5.1.3.1 Formal Description of the Grammar

Here is a formal description of the filter expression grammar:

```
# The filter itself

FILTER ::= AFILTER

| FILTER # AFILTER

AFILTER ::= funcfilter ( FUNCFILTARG )

| collfilter ( COLLFILTARG )

| p2pfilter ( P2PFILTARG )

# Specifying functions

FUNCFILTARG ::= FUNCEXPR

| all

| none

FUNCEXPR ::= FUNCATOM
```

```
| FUNCEXPR && FUNCATOM

| FUNCEXPR || FUNCATOM

FUNCATOM ::= TG

| FG

| STARTTIME

| ( FUNCEXPR )

| ! FUNCATOM

# Specifying messages

P2PFILTARG ::= P2PEXPR

| all

| none

P2PEXPR ::= P2PATOM

| P2PEXPR && P2PATOM

| P2PEXPR || P2PATOM

P2PATOM ::= DURATION

| COMM

| TAG

| P2PVOLUME

| TGSENDER

| TGRECEIVER

| COMMSENDER

| COMMRECEIVER

| TGSRPAIR

| COMMSRPAIR

| TG

| COMMSR

| STARTTIME

| ENDTIME

| SENDER_FG
```

```
| RECEIVER_FG

| ( P2PEXPR )

| ! P2PATOM

# Specifying collective operations

COLLFILTARG ::= COLLEXPR

| all

| none

COLLEXPR ::= COLLATOM

| COLLEXPR && COLLATOM

| COLLEXPR || COLLATOM

COLLATOM ::= DURATION

| COMM

| COLLOPTYPE

| COLLVOLUME

| TGROOT

| COMMROOT

| TG

| STARTTIME

| ENDTIME

| ( COLLEXPR )

| ! COLLATOM

# Specifying times

STARTTIME ::= start ( TRIPLETS ; INTEGER )

| start ( TRIPLETS )

ENDTIME ::= end ( TRIPLETS ; INTEGER )

| end ( TRIPLETS )

DURATION ::= duration ( TRIPLETS )

# Specifying TGroups and FGroups

TG ::= tg ( NAMES )
```

```
FG ::= fg ( NAMES )

SENDER_FG ::= send_fg ( NAMES )

RECEIVER_FG ::= recv_fg ( NAMES )

# Specifying collective operation and message properties

COMM ::= comm ( TRIPLETS )

TAG ::= tag ( TRIPLETS )

COLLOPTYPE ::= type ( COLLNAMES )

COLLVOLUME ::= volume ( TRIPLETS )

P2PVOLUME ::= volume ( TRIPLETS )

# Specifying root, sender, or receiver, either by a TGroup name

# or by position in the communicator. If the operation has no

# root, then root() is always false.

TGROOT ::= root ( NAMES )

TGSENDER ::= sender ( NAMES )

TGRECEIVER ::= receiver ( NAMES )

# The predicate sr specifies both sender and receiver, separated

# by a semicolon.

TGSRPAIR ::= sr ( NAMES ; NAMES )

COMMROOT ::= root@ ( TRIPLETS )

COMMSENDER ::= sender@ ( TRIPLETS )

COMMRECEIVER ::= receiver@ ( TRIPLETS )

# The predicate sr@ specifies both sender and receiver ranks,

# separated by a semicolon.

COMMSRPAIR ::= sr@ ( TRIPLETS ; TRIPLETS )

COMMSR ::= tg@ ( TRIPLETS )

# Names containing fancy characters have to be double-quoted.

# Names map to TGroup and thread names, FGroup and function

# names, or collective operation types, depending on the context.

NAMES ::= NAME
```

```
| TRIPLET

| NAMES , NAME

| NAMES , TRIPLET

COLLNAMES ::= COLLNAMELIST

| TRIPLETS

COLLNAMELIST ::= NAME

| COLLNAMELIST , NAME

NAME ::= [_a-zA-Z][_a-zA-Z0-9.]*

| \"[^\"]*\"

# Specifying triplets and numbers

TRIPLETS ::= TRIPLET

| TRIPLETS , TRIPLET

TRIPLET ::= INTEGER

| INTEGER :

| INTEGER : INTEGER

| INTEGER : INTEGER : INTEGER

INTEGER ::= [0-9]+
```

5.1.3.2 Examples of Advanced Usage of the Grammar

This section includes several examples of manually using the filter expression grammar and how it provides advanced capabilities in filtering trace data and speeding up the process of selecting exactly what the user would like to be analyzed.

For the first example, consider a parenthesized structure that can not be built by the point and click interface as easily (messages sent by process 0 and starting or ending between 70000 and 80000 ticks):

```
p2pfilter( sender( 0 ) && ( start( 70000:80000 ) || end( 70000:80000 ) ) )
```

The following example uses the predicate `sr`, which is not available in the point and click interface, to efficiently filter out all messages between processes 0 and 1:

```
p2pfilter( ! sr( 0:1; 0:1 ) )
```

Finally, consider the following scenario. With the point and click interface, a complicated filter is specified for a certain filter class with a large number of predicates and Boolean operators (both AND and OR, the latter added by using the **Add New Clause** button). Now, to negate everything that has been specified so far (that is, to get exactly the trace data that was previously being filtered out), use `!` in front of the whole expression when in the manual mode. For example, the filter below specifies `MPI_Barrier`

collective operations that last no longer than 2000 ticks, plus all collective operations with process 0 as the root:

```
collfilter( type( MPI_Barrier ) && duration( 0:2000 ) || root( 0 ) )
```

while this filter specifies all the collective operations that do not match the description above:

```
collfilter( ! ( type( MPI_Barrier ) && duration( 0:2000 ) || root( 0 ) ) )
```

5.1.4 Filter Expressions in Comparison Mode

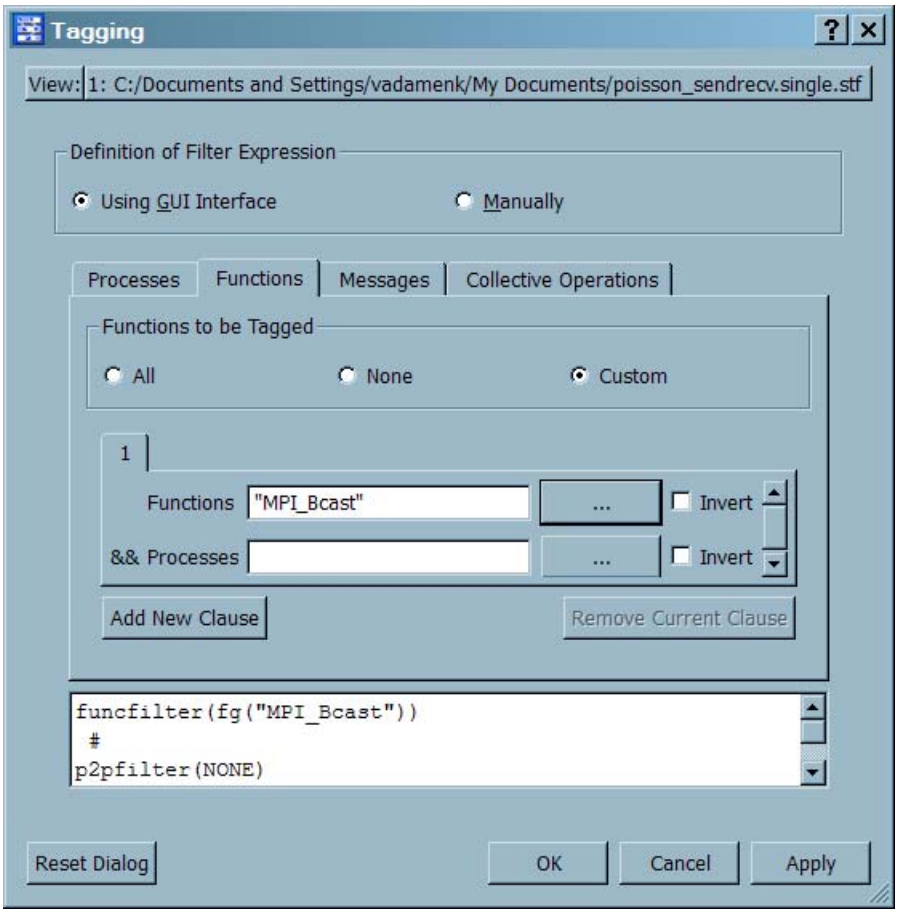
In Comparison mode (see Comparison of two Trace Files) the dialog is extended by a label and a combo box that allows to apply the chosen filter expression to the other trace file shown in the View. Choosing the option **If expression is valid in other file** will check if the resulting expression is valid in the name space of the other file and if so then apply the expression as if you had manually typed it in the other file filter dialog. Any previous input in the target dialog will be overwritten.

5.2 Tagging Dialog

For explanation of the filtering concept refer to Tagging and Filtering. The usage of this dialog box is as for the **Filtering** dialog box and described in Filtering Dialog.

The only difference is that the default **Filtering** dialog box lets all events pass while the default **Tagging** dialog box does not tag any event at all.

Figure 5.4 Tagging Dialog



5.3 Trace Idealizer Dialog Box

The Trace Idealizer dialog produces an ideal trace. The dialog contains the following elements:

File displays the path the filename of the project input file to be converted.

Output file name contains the name of ideal trace output file. By default, the output file is placed in the same folder of the input file, with the suffix `ideal` added before the .sfx extension. Type in, or browse to a different location as needed.

Trace conversion parameters specifies the time range of conversion, based on the original trace file times. The time range is specified in seconds, in the **start time** field and **end time** fields. The default values are obtained from the original trace.

The progress bar indicates the percentage of conversion done. Before clicking the **Start** button, the progress label is **Ready to Start**. After clicking the **Start** button, the label changes to **Converting**.

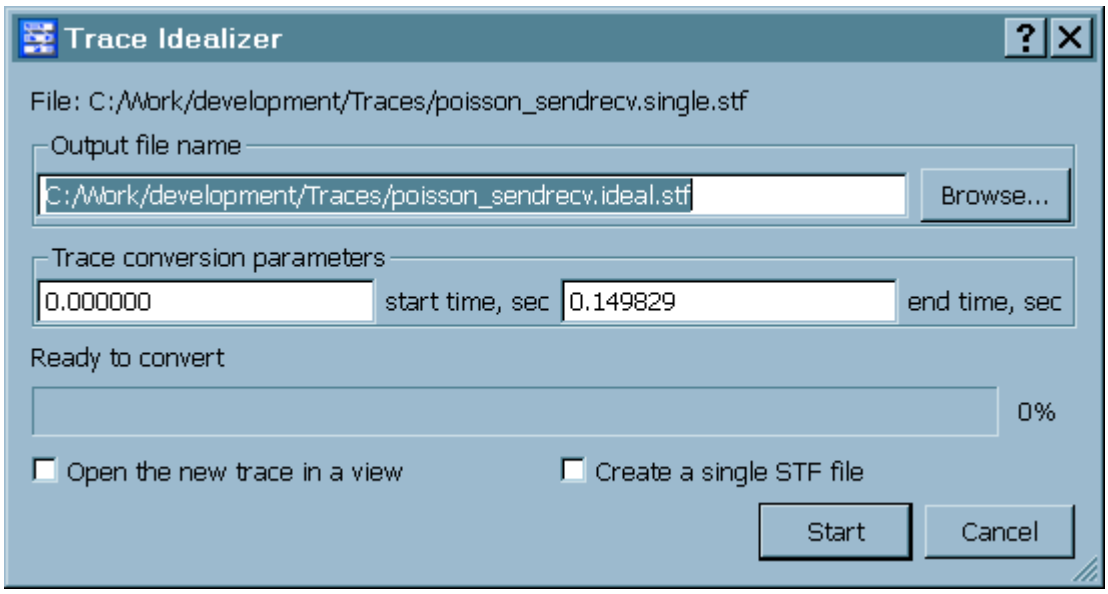
Open the trace in a view

Check this box to open the newly produced trace in a separate Intel Trace Analyzer view.

Create a single structured trace format (STF) file

Check this box to store the resulted trace as a single file. By default, the idealizer creates ?? format output files.

Figure 5.5 Trace Idealizer Dialog



5.4 Application Imbalance Diagram Dialog Box

The Application Imbalance Diagram enables you to compare the performance between a real trace and an ideal trace.

Use the Application Imbalance Diagram total mode to see the time spent on calculation, interconnect, and imbalance; as well as the time distribution between them. The diagram displays the distribution with different colors.

Use the Application Imbalance Diagram breakdown mode to see:

- how three most time-expensive functions of the application work
- how much time have they spent on three different kinds of messages
- what is the proportion between interconnect and imbalance for each kind

To compare a real trace with an ideal trace, do the following steps:

1. Go to **Views Menu > Advanced > Application Imbalance Diagram**.
2. In the **Choose Idealized Trace** dialog box, choose a pair of real and ideal traces that you want to analyze. You can change the following parameters, and then click **OK**. See Figure 5.6:

- Breakdown message borders
- Rebuild the BDI files

  If you have changed the BDI files manually or the files were partially corrupted, select this checkbox.

  The BDI files are ASCII* files and contain all the needed information to create an Application Imbalance View. The file name is <original trace name>.bdi. The file format is the following:

  <version string>

  <short message border> <medium message border>

  <Function ID> <Collective ID> <Function Name> <Total Function Time> <Short Messages Time> <Medium Messages Time> <Large Messages Time>

NOTE: Do not change the BDI file manually. It may result in incorrect feature functioning. The BDI file can be imported into an Excel table.

3. The **BDI File Building** dialog box displays the progress of building BDI files. To stop the BDI building, click **Cancel**. See Figure 5.10.

NOTE: The **Application Imbalance Diagram** dialog box does not display if you stop the BDI building.

4. In the **Application Imbalance Diagram** dialog box, specify the mode to display (Total mode or BreakDown mode) and whether to display the application time. See Figure 5.7.

   To change the colors used in the Application Imbalance Diagram, do the following steps:

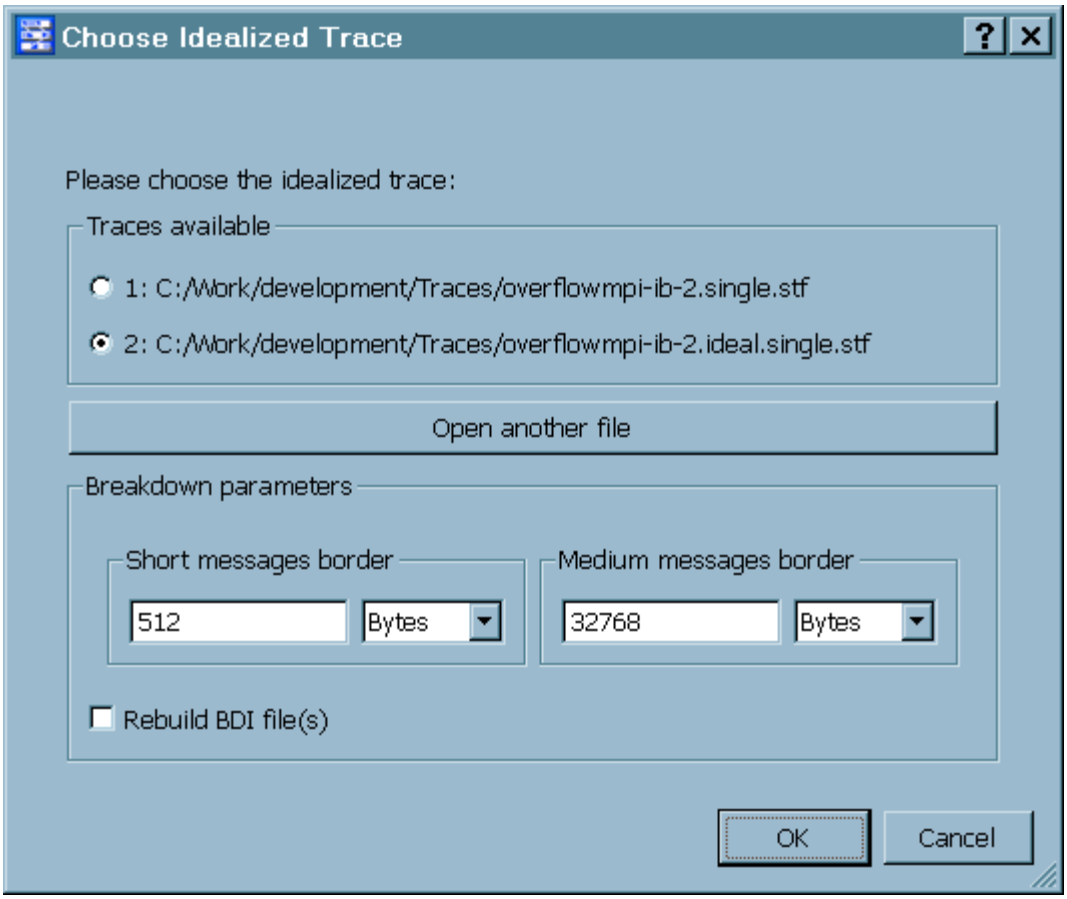
   a. In the **Application Imbalance Diagram** dialog box, click **Colors.**

   b. In the **Application Imbalance Diagram Colors** dialog box, click **Choose Colors**. See Figure 5.9.

   c. Click **OK** to save the changes.

NOTE: The new colors are stored in the Intel Trace Analyzer Resource file so you don't need to change them every time you start the Analyzer. To restore the default colors for all elements, click **Reset all colors**.

Figure 5.6 Choose Idealized Trace Dialog



The Breakdown (Figure 5.8) mode displays the following information:

- three most time-expensive functions
- the time differentiation between several kinds of messages. The messages can be:
  - short, 0-512 bytes
  - medium, 512-32767 bytes
  - large, > 32767 bytes

  You can change the default borders by using the Step 2.
- the information that describes how these functions are balanced for these kinds

Figure 5.7 Application Imbalance Diagram View

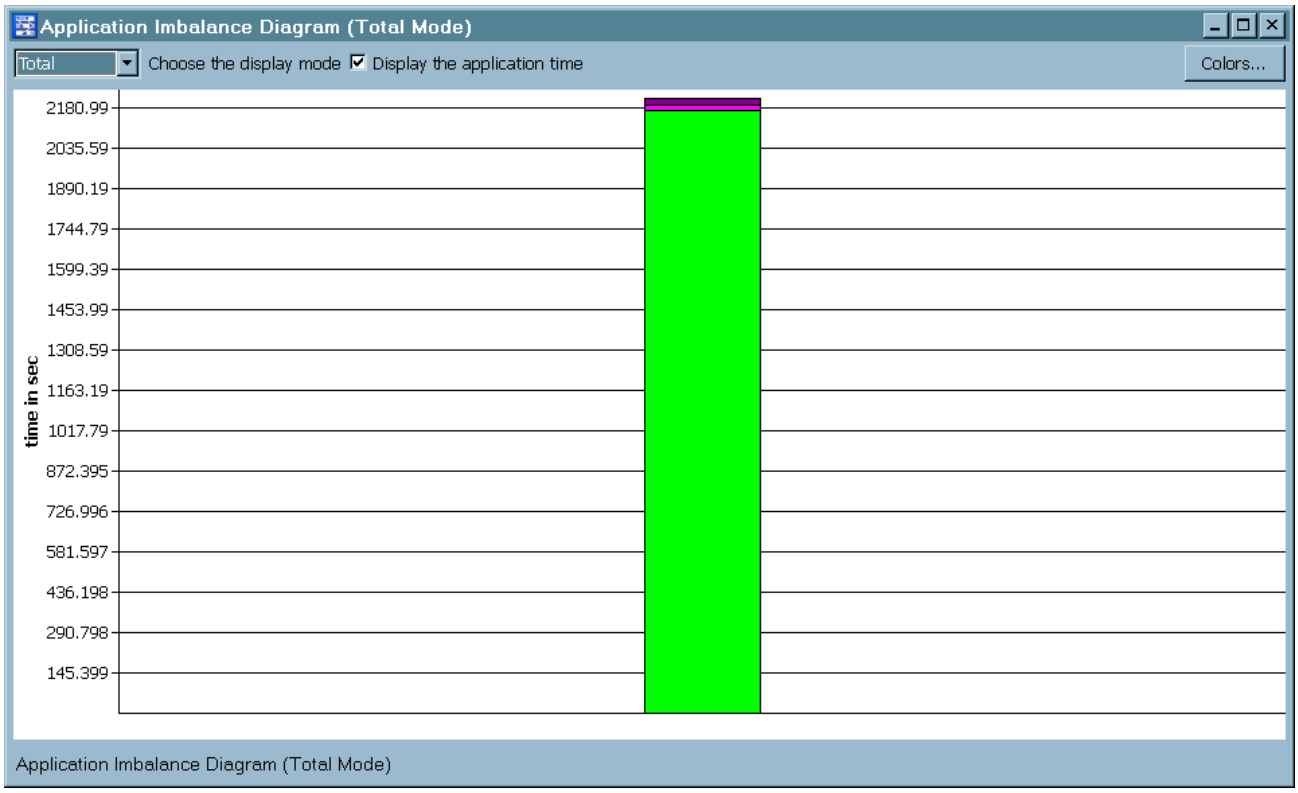


Figure 5.8 Application Imbalance Diagram View (Breakdown Mode)

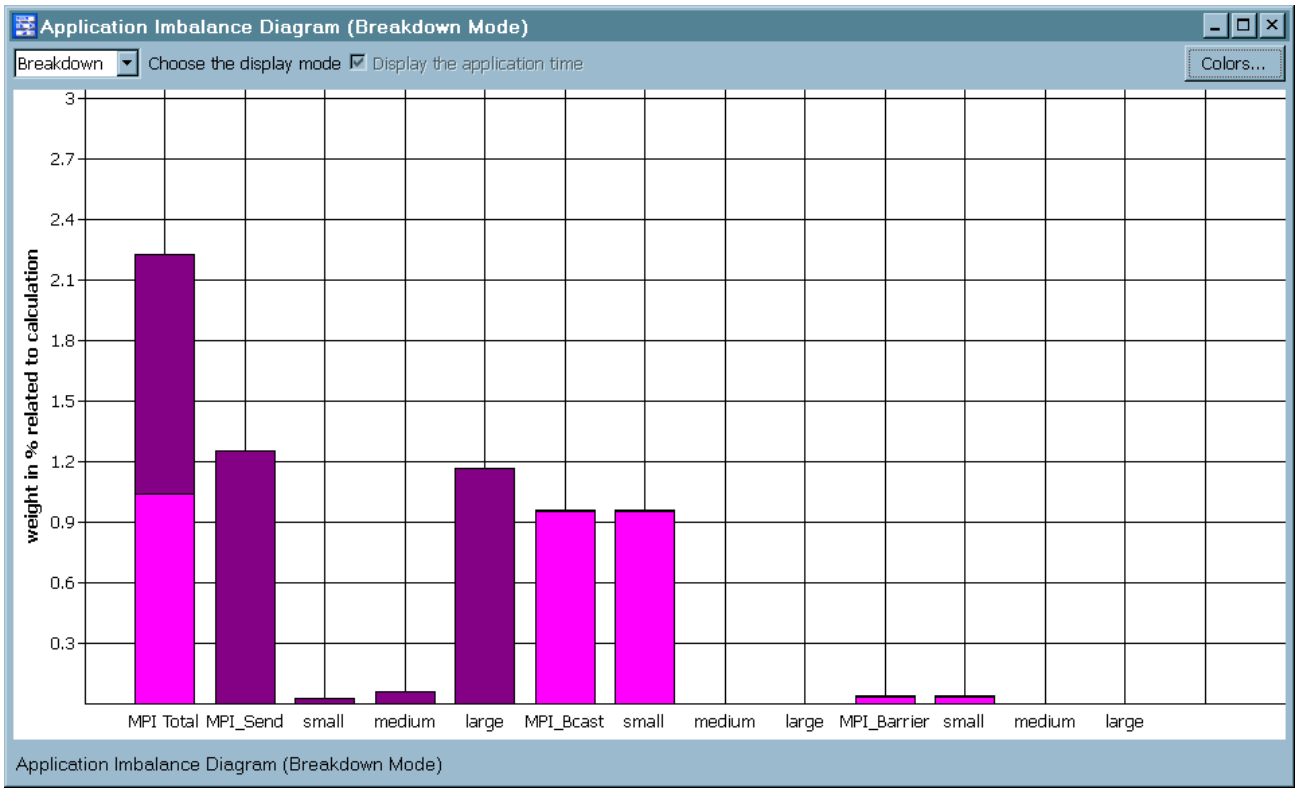


Figure 5.9 Application Imbalance Diagram Colors

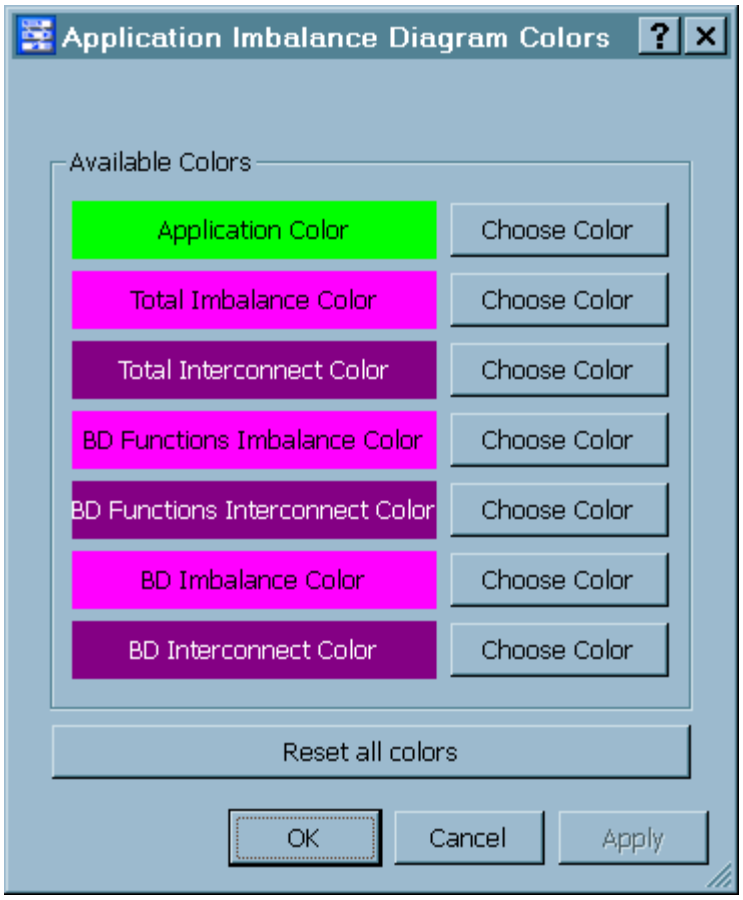
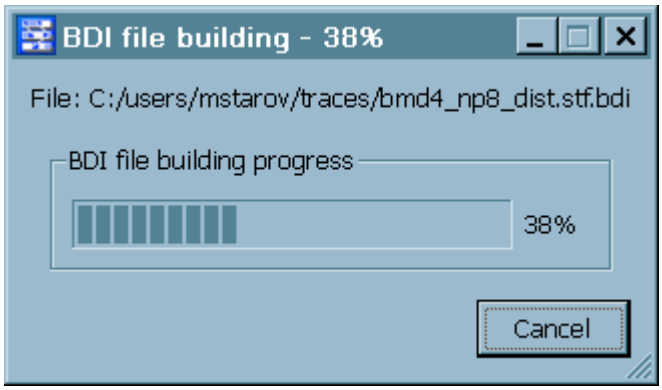


Figure 5.10 BDI File Building Progress Dialog Box

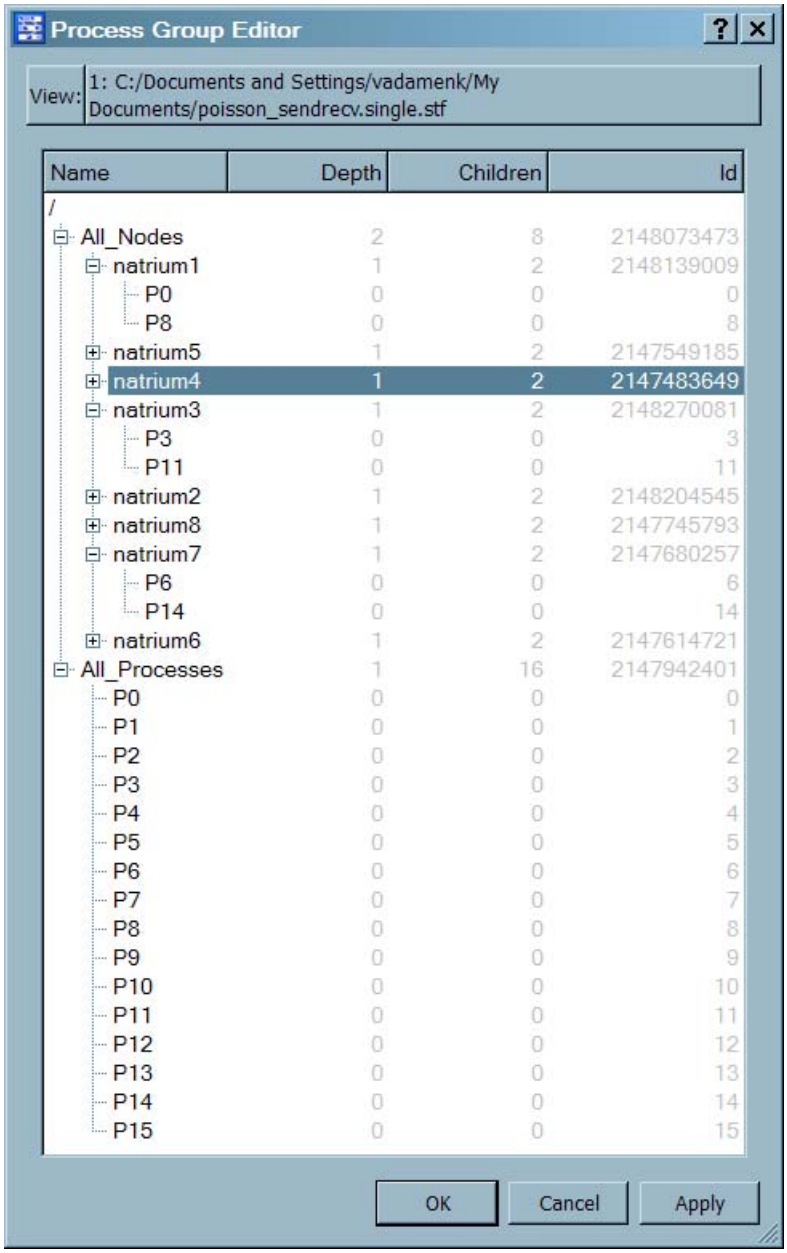


5.5 Process Group Editor

The Process Group Editor is found at **Advanced > Process > Aggregation**. The Process Group Editor provides two functions. One is to select a process group (or function group in general) for process

aggregation. The other is to create new groups beyond the ones that are provided by default. Group definitions are stored in the file `.itarc` in your home directory (for an English Microsoft Windows XP* installation this will be something like `C:\Documents and Settings\%username%\.itarc`). See Configuration Dialogs for other ways to save, edit and load configuration information.

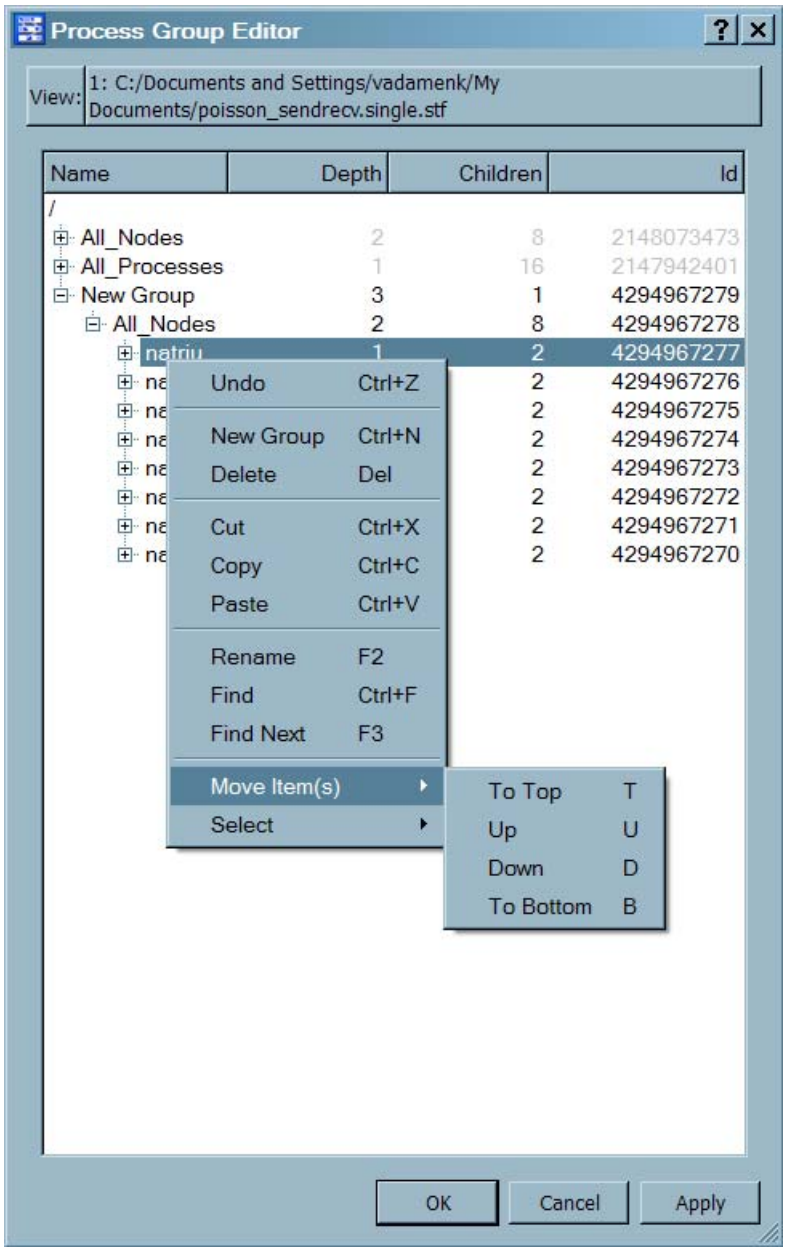
Figure 5.11 Process Group Editor



To select a process group for aggregation (see Aggregation) select the group by using the mouse or the cursor/arrow keys and press **Apply** or **OK**.

NOTE: The dialog box only accepts a single, non-empty process group that contains each of its functions no more than once.

Figure 5.12 Process Group Editor's Context Menu



Right-click an item in the editor to bring up a context menu with several entries.

The entry **Undo** (`Ctrl+Z`) allows to revert the last actions (delete, cut, move, copy and others).

The entry **New Group** creates a new group as a child of the clicked item. The entry **Delete** removes an item. This entry is disabled for items that originate of the trace file. Only user-created items are deleted.

The entry **Rename** (`F2`) allows to rename user-created items. The entry **Find** (`Ctrl+F`) opens the **Find** dialog box (see Find Dialog) and **Find Next** (`F3`) searches for the next match if a search was started before.

The entry **Move Item(s)** allows to move several selected siblings up or down relative to their unselected siblings through a submenu. It is much more convenient to use the keyboard shortcuts shown in the submenu instead of the menu itself.

The entry **Select** opens a submenu that allows conveniently selecting subsets of the tree under the clicked item. The entries provided distinguish between selecting processes, selecting process groups or both, and if the selection should include only the direct children or all descendants.

Another way of editing is to drag a group, a process, or the entire current selection and to drop it into a target group. This can result in groups that contain the same function twice or more. In the group hierarchy, such groups are accepted for storage but not for aggregation. Empty groups are deleted automatically when the dialog box is closed.

5.5.1 Comparison Mode

In comparison mode (see Comparison of two Trace Files) the dialog is extended by a label and a combo box that allows to apply the chosen aggregation to the other trace file shown in the View. Basically using the option **If match found** will try to find a process group of the same name in the other file and choose it.

For many usage scenarios that allows to choose a new aggregation in comparison mode for both trace files at once.

5.6 Function Group Editor

The Function Group Editor is accessed at **Advanced > Function Aggregation**. In most respects the Function Group Editor works exactly like the Process Group Editor (see Process Group Editor). It allows to edit function group definitions and to choose a function group for aggregation in a View. The only addition is that functions and function groups are assigned colors through the context menu (see Function Group Color Editor).

When the mouse hovers over the first column of a group entry a tooltip window with the function group name is shown. For a function a two-lined tooltip with the function name and the full name of the function is shown. The full name (or static path) reflects the original definition of the function or group as stored in the trace file.

To change the color of any of the functions, select the **Color** option from the context menu shown in *Figure* 5.13. A **Color** dialog box is opened where the preferred color for the given function is selected. The context menu also has the option of assigning the color of the parent node to all the children, using the **Assign Color to Children** entry. This entry is enabled only if a parent node is selected.

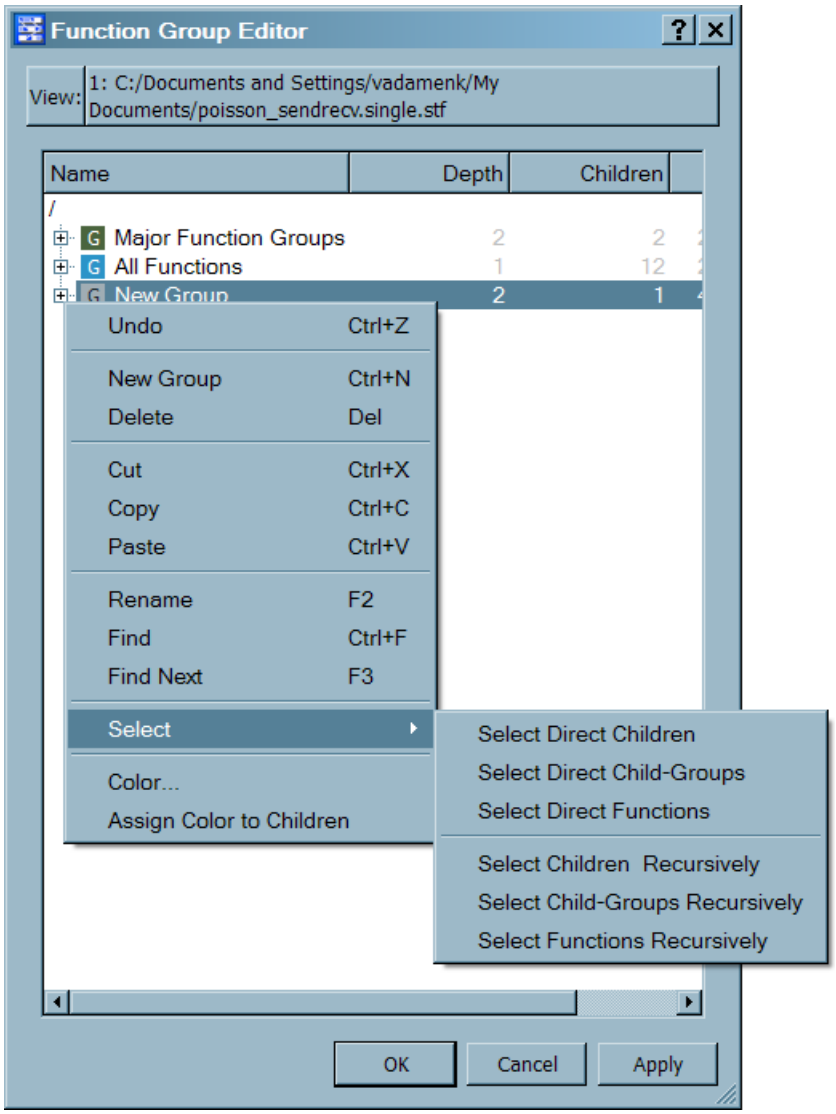
5.6.1 Comparison Mode

In comparison mode (see Comparison of two Trace Files) the dialog is extended by a label and a combo box that allows to apply the chosen aggregation to the other trace file shown in the View.

Basically using the option **If match found** will try to find a matching function group according to the matching rules explained in Mapping of Functions and choose it. This works pretty well for predefined groups. If no match is found the aggregation for the other file remains unchanged.

Using the option **Always (Create matching FGroup)** first tries to find a match. If a match is not found then a matching function group in the other files name space is created. Beware that the outcome of this operation will be a function group that will mimic the original group's hierarchical structure but it will only contain functions that are present in both trace files. Use this option with great care.

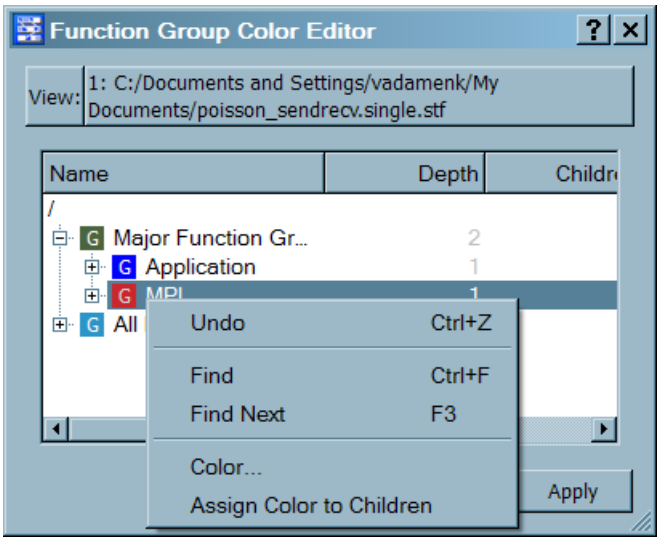
Figure 5.13 Function Group Editor's Context Menu



5.7 Function Group Color Editor

The Function Group Color Editor is the Function Group Editor (see Function Group Editor) in a restricted mode that only allows editing the colors of functions and function groups. The Function Group Color Editor can only be accessed when choosing to change the function colors in the Event Timelines settings dialog box (see Event Timeline). The context menu of the Function Group Color Editor has a **Find** entry and a **Find Next** entry, both of which are explained in Process Group Editor and also a **Color** entry and an **Assign Color to Children** entry, which are explained in Function Group Editor.

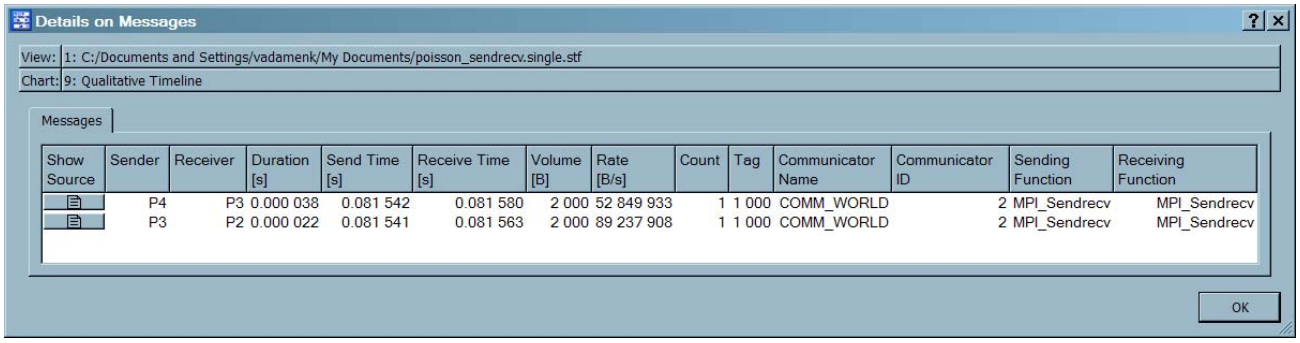
Figure 5.14 Function Group Color Editor



5.8 Details Dialog

This dialog box is available from the context menus of the Event Timeline (see Event Timeline) and the Qualitative Timeline (see Qualitative Timeline).

Figure 5.15 Details on Messages Shown in the Qualitative Timeline



The dialog box shows detailed attributes of the clicked events. Function events, messages and collective operations are shown in separate tabs. Each tab shows a list of event entries.

If the column **Count** shows a value greater than one, the event was created by merging several atomic events (see Level of Detail). Each entry representing a merged event shows a View, which focuses on the data that went into this entry using the drill down button shown next to the entry. This is called a Detail View; it is a full blown View without restrictions. In the dialog box below the list is a check box that allows to filter out the other event categories in the Detail View to be opened.

NOTE: A left click on reuses an existing Detail View so that the screen is not cluttered so easily. If desired, right-clicking the drill down button opens a new Detail View.

If source code location information is available for an entry, then a button **Show source** appears next to the entry. This button opens a **Source View** dialog box (see Source View Dialog). The source code location is only available up to a certain time interval. If the time interval is set in such a way that there is no enter event for a function, then the **Details** dialog is not aware of the source code location and consequently there will not be a **Show Source** button.

When a **Details** dialog is opened, it preserves all the settings of its View (aggregations, filters, ticks vs. seconds) and the event results for which it is opened. Further changes to the View, like a change in Aggregation, do not affect the dialog contents. When such a change is made to the View, the dialog View label is made bold and an asterisk character ***** appears in front of it. This is to indicate that the details shown are not updated to match the View/Chart. When the Chart or the View from which the Details Dialog originated are closed, the dialog is also closed.

5.8.1 Detailed Attributes of Function Events

Name: This attribute specifies the name of the selected function. If it represents a Function Group, then it is prefixed with the Group name.

Process Group: Specifies the Process, Function or Group in which the selected event occurred

Duration: This indicates the time spent in a given Function/Group. For coarser resolutions (Count ≥ 1), the value does not reflect the actual time spent in this function but the length of the time interval over which several function events were merged.

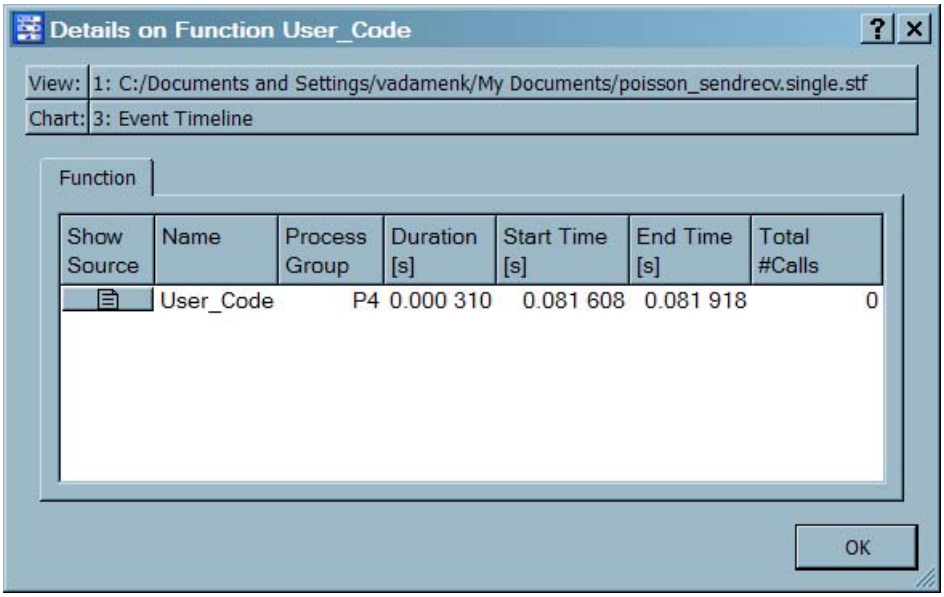
Start Time: This shows the time when the event entered the Function/Group. For coarser resolutions (Count ≥ 1) the value represents the start of the time interval over which several function events where merged.

End Time: This shows the time when the event left the Function/Group. On coarser resolutions (Count ≥ 1) the value represents the end of time the interval over which several function events where merged.

NOTE: Sometimes a Function starts before and/or ends after the displayed time interval. In these possible cases, you may see in the Details dialog a less **<** character preceding the numeric value listed under the column Start Time, and a greater **>** character preceding the numeric values listed under the columns **End Time** and **Duration**. These numeric values are the boundaries of the current zoom interval.

Total #Calls: Total #Calls gives the total number of function calls. It contains all calls covered by the function in the given time interval, including calls to functions other than the function clicked.

Figure 5.16 Details on Functions Shown in the Event Timeline



5.8.2 Detailed Attributes of Message Events

Sender: The Sender is the Process, Function or Group which sent the message.

Receiver: Process, Function or Group which received the message.

Duration: The duration specifies the time taken by the merged operation. It is the difference between Send Time and Receive Time.

Send Time: Specifies the time when the message was sent. If more than 1 message is represented (Count ≥ 1), then the first Send Time of any member in the merge is specified.

Receive Time: This indicates when the message was completely received. If more that 1 message is represented (Count ≥ 1), then the last Receive Time of any member in the merge is specified.

Volume: The total number of bytes sent with selected message(s).

Rate: This indicates the rate at which the bytes are transferred. It is calculated using Volume/Duration.

Count: This specifies the number of messages that are merged into the selection.

Tag: This attribute specifies the MPI tag of the message. If more than one message is merged together, then the tag of the first message is shown.

Communicator Name: The name of the MPI communicator on which the message(s) was (were) sent is specified in this attribute.

Communicator ID: The plain ID of the MPI communicator on which the message(s) was (were) sent is given by this attribute.

Sending Function: The name of the MPI function from which the message(s) was (were) sent is given in this attribute.

Receiving Function: This specifies the name of the MPI function which received the sent message(s).

5.8.3 Detailed Attributes of Collective Operation Events

Each possibly merged collective operation has a header entry which describes the collective operation as a whole. The plus handle gets a detailed list of the same information per Process/Group. The exact processes or process groups shown depend on the current process aggregation.

Name/Process: On the per-operation row this column lists the name of the selected operation (**Mixed** if different operation types were merged). On per-process rows it shows the name of the Process/Group.

Duration: Last Time minus First Time

First Time: First time, one of the merged operations was entered.

Last Time: Last time, one of the merged operations was left.

Volume Sent: Number of bytes sent. It is the sum of all bytes sent on all merged operations for the per-process rows. The per-operation row sums up all per-process rows.

Volume Received: Number of bytes received. It is the sum of all bytes received on all merged operations for the per-process rows. The per-operation row sums up all per-process rows.

Count: Number of collective operations that are merged into the selection.

Root: The root process.

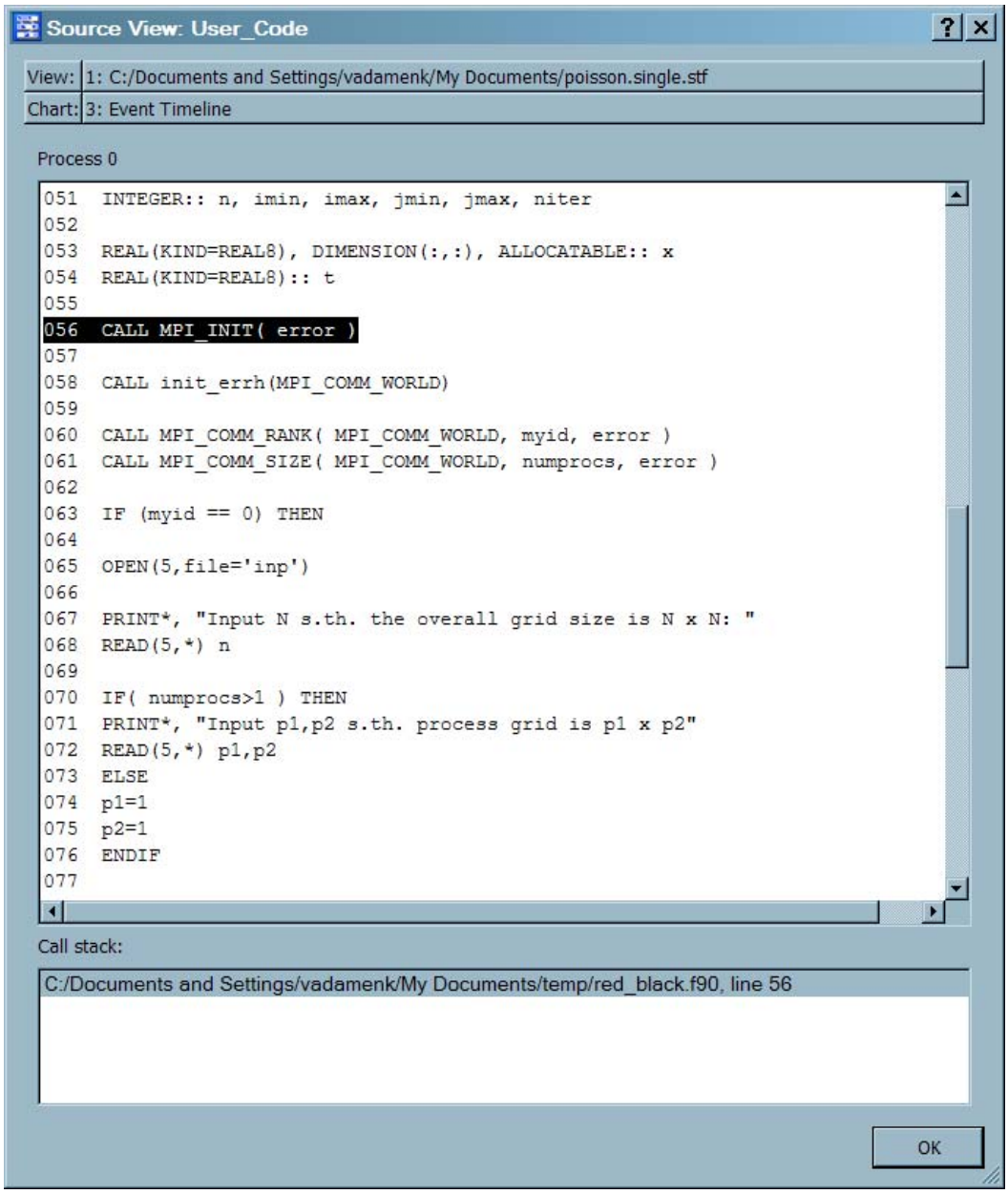
Communicator Name: The name of the MPI communicator on which the collective operation(s) was (were) executed.

Communicator ID: The plain ID of the MPI communicator on which the collective operation(s) was (were) executed.

5.9 Source View Dialog

The Source View dialog box is opened from the **Details** dialog box (see Details Dialog) if source code location information is available in the trace file.

Figure 5.17 Source View Dialog Box



The dialog box consists of a combo box to choose processes, a text browser in the center and a list box representing the call stack at the bottom.

The combo box allows selecting from several processes (functions) if the dialog box was opened for a collective operation, from two processes if it was opened for a message and it degenerates to a label if the dialog box was opened for a function event.

The text browser shows a source file. The line corresponding to the current stack level is shown highlighted in reverse video.

The list at the bottom allows selecting from the stack levels that were stored with the source code location information. Selecting an entry from this list switches the text browser to the file and line number matching the stack level.

The default is to search source code files in the current directory and the directory of the current trace file. If no source files are found, then a file open dialog box is shown to manually specify the source file to load. The specified path is added to the directory search path. Use the entry **File Default / UserDefines SCLSearchPaths** in the configuration dialog box to specify additional directories to be searched. Refer to Edit Configuration Dialog for details.

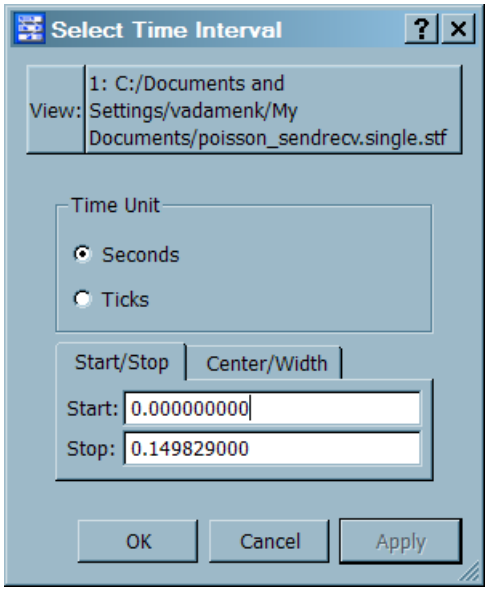
5.10 Time Interval Selection

When opened from **Views Menu > Navigate > Goto (G)**, this dialog box allows entering a new time interval for the whole View. This interval is pushed onto the zoom stack (see Zoom Stack) and the View is updated accordingly.

When opened from the filter dialog boxes (see Filtering Dialog and Tagging Dialog), this dialog box enters or edits a time interval or duration in a filter expression.

The time interval is specified in ticks or seconds. The interval is entered either by giving the start and stop or by giving the center and width. Entering a value that is greater than the maximum value of the trace file's time interval is possible; this value is automatically reduced to the maximum value (`tmax` of the trace).

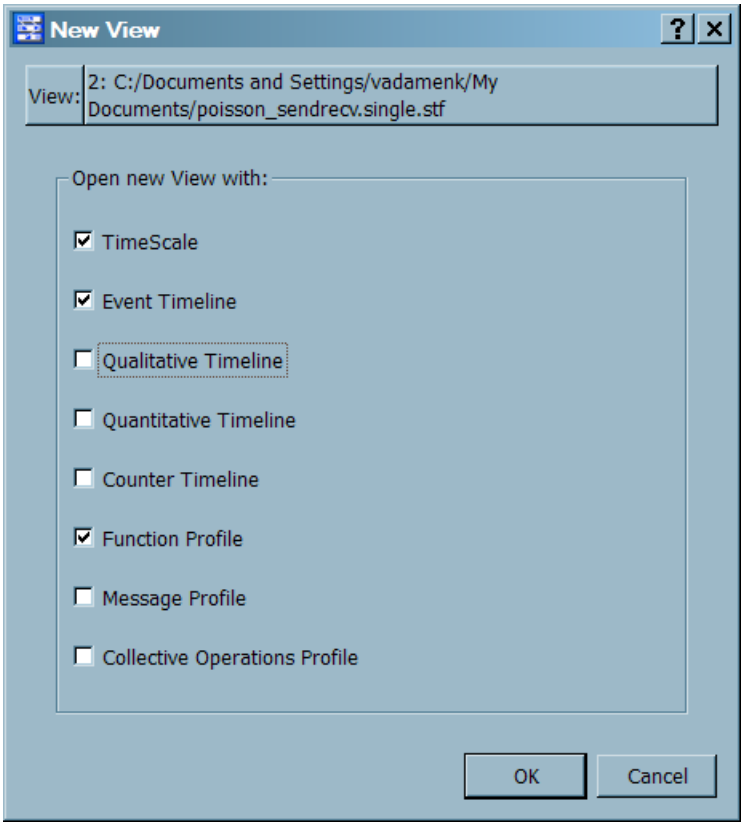
Figure 5.18 Selecting a Time Interval



5.11 New View Dialog

This dialog box appears when **Views Menu > View > New View** is chosen and allows specifying which charts are made visible in the new View.

Figure 5.19 New View Dialog Box



5.12 Configuration Dialogs

These dialog boxes enable manipulating the configuration that is usually stored in the file `.itarc` in your home directory upon program exit. The configuration consists of the global information found under the option **File Default** and the per file information found under the respective file name.

The global information consists of the recent file list and the list of search paths that find the source code files. The latter is stored in the option **File Default / UserDefines SCLSearchPaths** and contains a list of directories, separated by semicolons **;**. These directories are searched in the given order for source code files to be shown in the **Source View** dialog box (see Source View Dialog).

The per-file information holds all user-defined process groups, function groups and function group colors.

5.12.1 Edit Configuration Dialog

This dialog box allows editing the configuration; for example, by changing values or removing entries from the configuration.

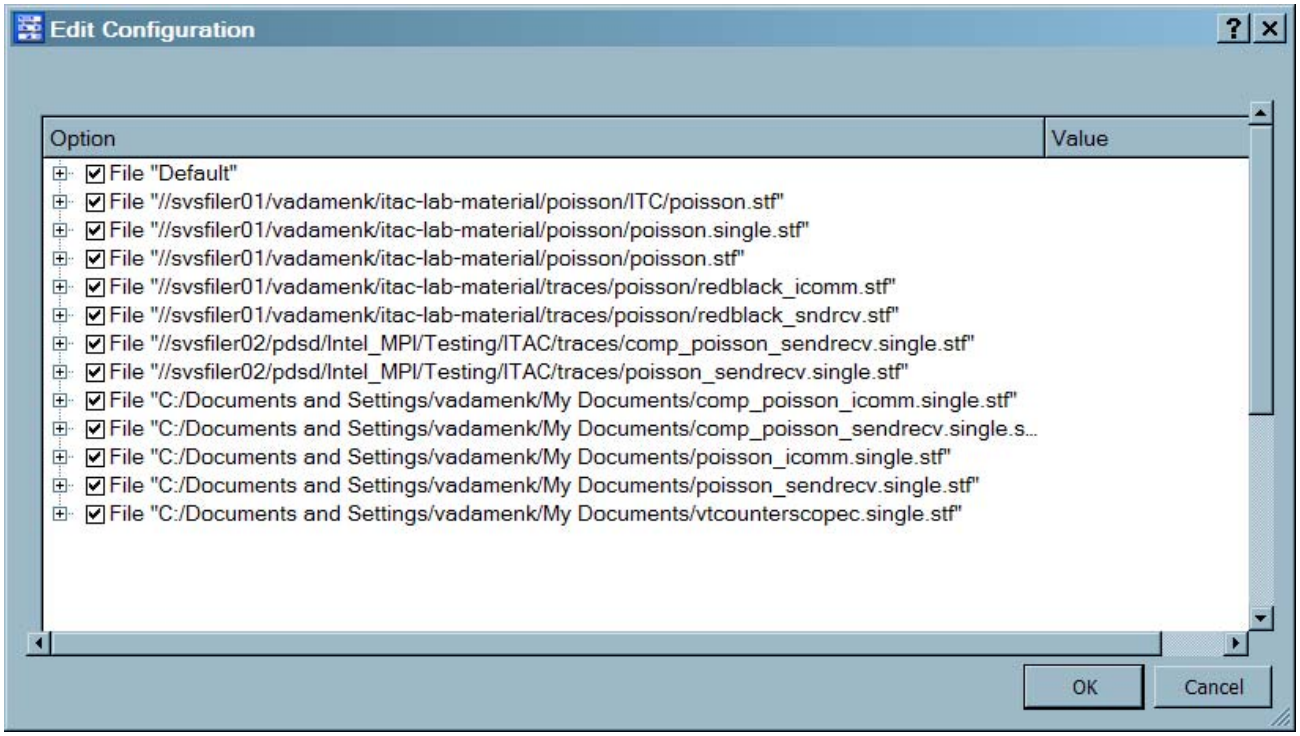
This dialog box together with the **Load Configuration** dialog box allows storing per-trace file settings together with the trace file so that definitions of process groups and function groups are shared in a work group.

The entries are shown with check box like handles that are all checked by default. Unchecking entries removes them from the current configuration when the button **OK** is pressed. To edit a value, double-click on the respective entry.

Group definitions and color assignments are dragged and dropped from one trace file branch to another or even onto the branch **File Default** to make them available for each new trace file. The dialog box allows dragging and dropping every child node in the configuration tree, and thus the configuration is easily manipulated in a way that leads to surprising results. In such cases, start it again with a well-known configuration.

Generally, it is best to avoid moving options whose values are files or lists of files. For example, a cache file usually corresponds to a particular trace file, and hence, moving a **UserDefines Cachefile** to another trace file or the default section would not make sense, unless the intent is to reuse that cache file with a different trace. On the other hand, color definitions can typically be dragged and dropped from trace to trace or to default (the latter would define that color for all trace files, unless they provide their own definition). Moving group definitions, however, requires more attention, since group ids in one trace might not make sense in another. Examination and possible value editing can solve such issues.

Figure 5.20 Edit Configuration Dialog



5.12.2 Load Configuration Dialog

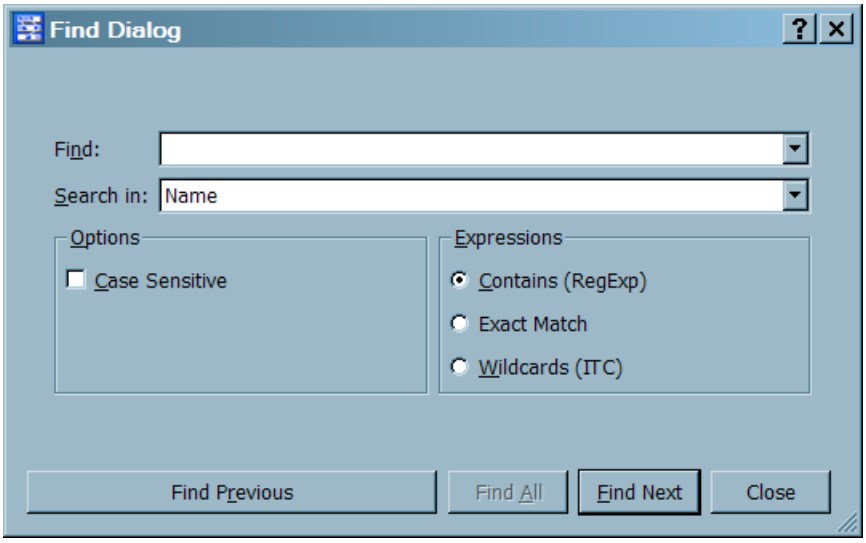
This dialog box allows reading configuration data from files into the program. The information to be loaded is chosen on a file per-file basis. Directly saving a configuration in Intel® Trace Analyzer is not possible. To save portions of the `.itarc`, use a text editor and save in the usual manner.

The check box **Merge** with the current configuration allows choosing if the eventually present configuration in memory for chosen files should be replaced by or merged with the configuration from the selected file.

5.13 Find Dialog

The **Find** dialog box searches for a process/function. This dialog box is found in the context menu of the Function Profile and in the context menus of the Function Group Editors and the Process Group Editors.

Figure 5.21 Find Dialog



Searching is made case-sensitive using the given check box in the **Find** Dialog box. A pull-down menu provides the option of searching by Name, by All Columns or by Dynamic Path.

The **Find** dialog box provides a few options with regard to the type of expression that optimize the search process. These are:

Contains (RegExp)

This entry searches for a phrase that contains a regular expression. A regular expression (RegExp) provides a way to find patterns within text. Regular expressions are built up from expressions, quantifiers and assertions. The simplest form of an expression is a character, like **x** or **5**. An expression can also be a set of characters. For example, `[ABCD]` would match an **A** or a **B** or a **C** or a **D**. More on the valid regular expressions is found at http://doc.trolltech.com/3.3/qregexp.html#details.

Exact Match

This entry searches for the exact match of a given character or set of strings.

Wild Cards (Intel Trace Collector)

This option defines a search for any function that matches the given pattern. Syntax and semantics are the same as in the regular expressions used in Intel® Trace Collector.

The wild card characters in use are *****, ******, **?** and **[]**. These match any number of characters except for the colon **:**. Pattern matching is not case-sensitive.

The state or function name that the pattern is applied to consists of a class name and the symbol name, separated by a colon. The colon is special and is not matched by the ***** or **?** wildcard. To match it use ******. Examples of valid Wild Card patterns are:

`MPI:*` (all MPI functions)

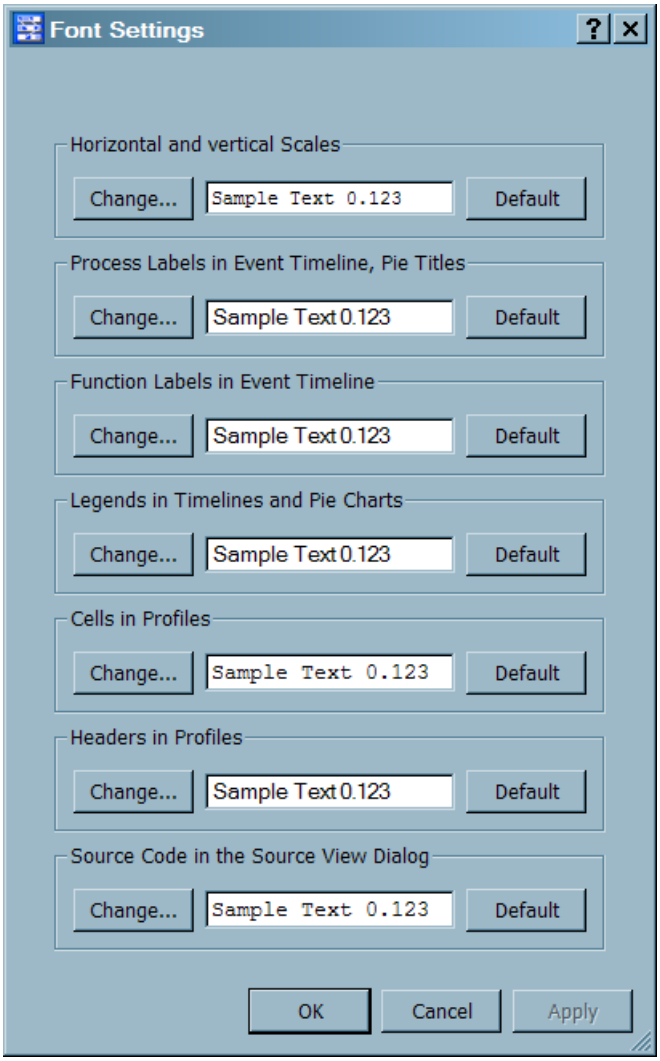
`**:*send*` (any function that contains "send" inside any class)

`MPI:*send*` (only send functions in MPI)

5.14 Font Settings

To change the fonts in any part of Intel® Trace Analyzer, select the **Fonts** dialog box from **Main Menu > Style > Set Fonts**. In the **Fonts** dialog box, you can change the fonts of the labels, legends or headers in the different Charts, for instance. Do this by clicking on the **Change** button. Another dialog box appears where the size, style and font can be changed.

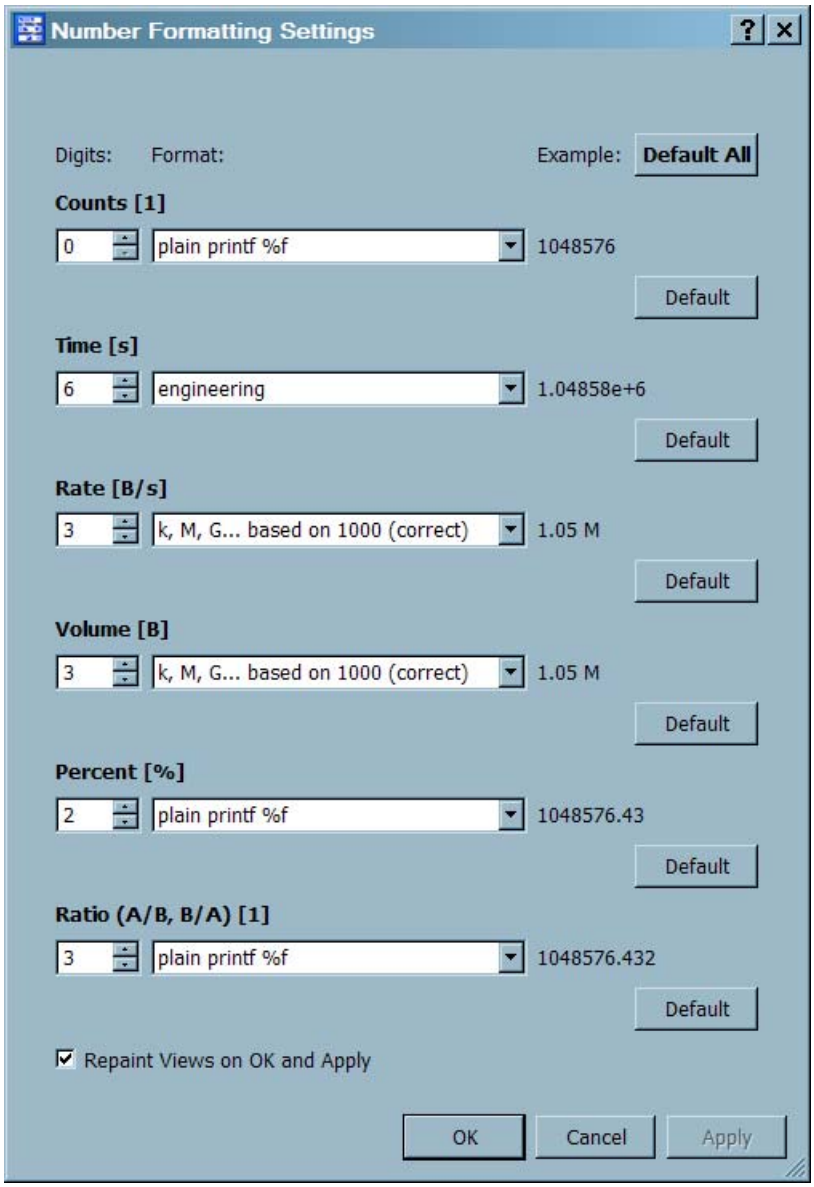
Figure 5.22 Font Settings



5.15 Number Formatting Settings

Use **Number Formatting Settings** to change the way numerical values of different units are represented. This dialog allows you to change the number of digits shown and the format in which the numerical value is shown for each unit. The number of digits is either interpreted as digits after the decimal point or as the number of valid (non zero) digits. The exact interpretation is dependent on the fact if the chosen format bases on the plain printf-format %f or %g.

Figure 5.23 Number Formatting Settings



At the bottom of the **Number Formatting Settings** dialog box, there is a check box that controls repainting. In the checked state, it ensures that all open Views are repainted upon **Apply** or **OK** to take on the new values. Unchecked, the changes in the format will only be visible with the next update. All values can be restored to their default by using the **Default All** button or by using the individual **Default** buttons for each number type.

6 Correctness Checking Reports

6.1 Appearance

The Correctness Checking Reports (CCRs) are shown on three charts:

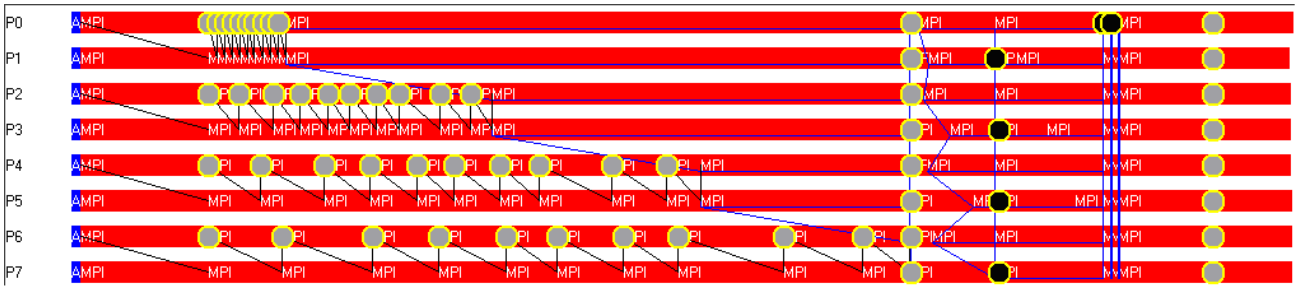
- Event Timeline
- Qualitative Timeline
- Debug EventAnalyzer

On the first two charts you can get a detailed dialog by clicking the right mouse button. The detailed dialog also contains details on the CCRs.

6.1.1 Event Timeline

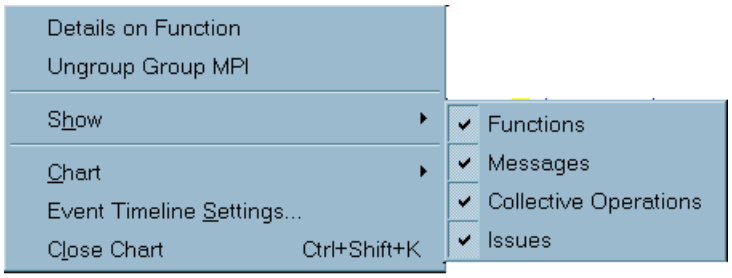
On the Event Timeline chart CCRs are displayed as yellow-bordered circles. The color of each circle depends on the type of the particular report: if it is error then the color is black, if it is warning – the color is grey.

Figure 6.1 Event Timeline with CCRs of Both Types



You can turn on/off the display of CCRs by clicking the right mouse button and checking/unchecking the **Issues** item in the **Show** submenu. The display is on by default.

Figure 6.2 Context Menu in Event Timeline



6.1.2 Qualitative Timeline

On the Qualitative Timeline chart CCRs are displayed as vertical lines representing the frequency of the CCRs occurrence. The height of each line is constant.

You can turn the display of CCRs on/off by clicking the right mouse button and selecting the **Correctness Reports** item in the **Events to show** submenu. The height of each CCR line depends on the level of the report. The lines representing errors are twice higher than the lines representing warnings.

The default event is Duration of Collective Operations.

Figure 6.3 Qualitative Timeline with CCRs

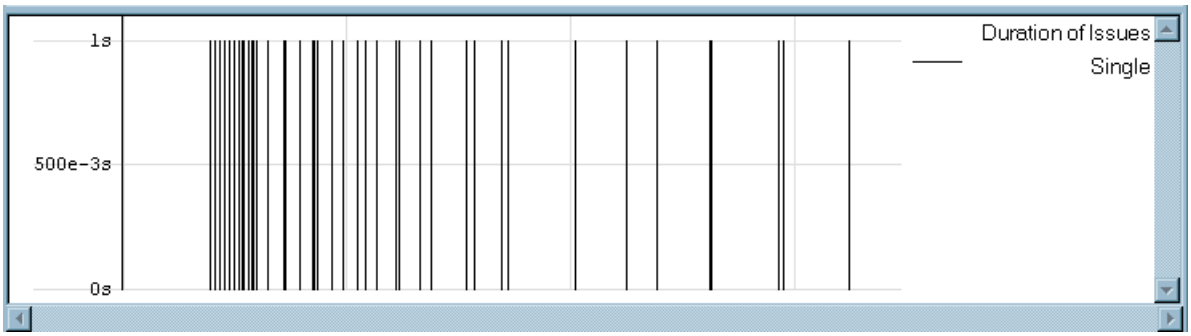


Figure 6.4 Submenu "Events to show" in Qualitative Timeline Context Menu

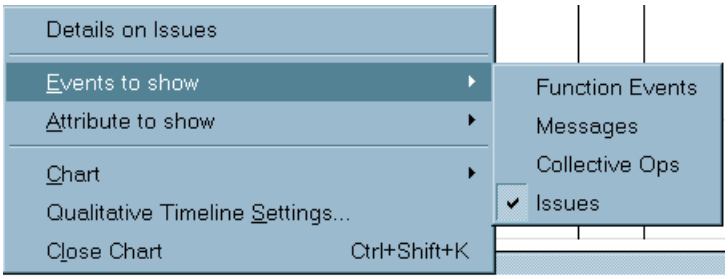
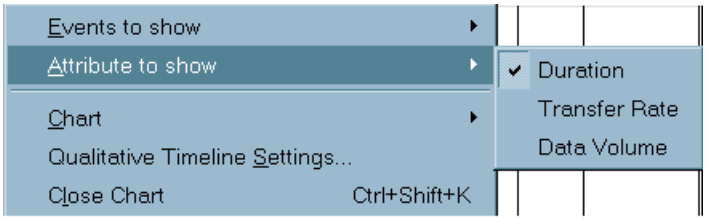


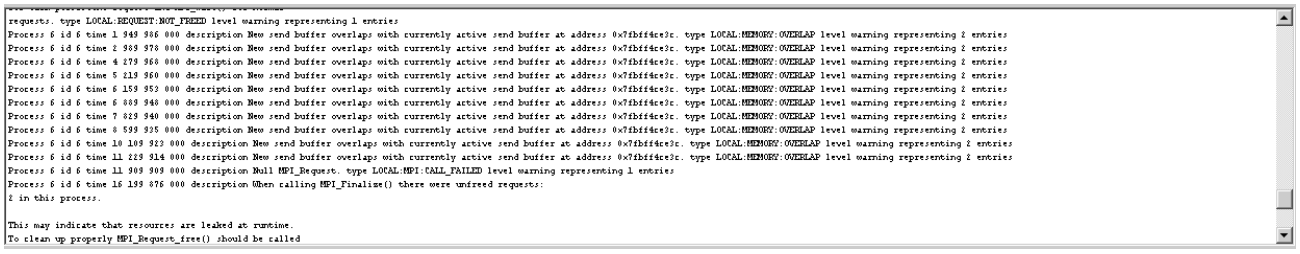
Figure 6.5 Submenu "Attribute to show" in Qualitative Timeline Context Menu



6.1.3 Debug EventAnalyzer

On the Debug EventAnalyzer chart CCRs are displayed in text mode – each field of a report is just printed.

Figure 6.6 CCRs in Debug EventAnalyzer Chart



6.1.4 Detailed Dialog

The detailed dialog displays the information about CCRs if they are present at the point where you have right-clicked. The information contains all fields of the Report Data structure that comes from STF. Each data item may contain five fields that are specifiers for the particular report. You can expand each report item in the Detailed Dialog to get the information from the specifiers.

Here are the descriptions of each field:

- **Process** – process where the issue occurred.
- **Show Source** – button. By pressing it you can get the exact line in the code at which the issue occurred.
- **Time** – moment of time (in seconds or ticks) at which the issue occurred.
- **Type** – string containing the type of the issue.
- **Level** – string containing the level of the issue. Can be **warning** or **error**.
- **Entry Time** – vector containing moments of time for every process involved into the issue.
- **Entry Process** – vector containing numbers of the processes involved into the issue.
- **Header** – vector of strings containing descriptions of the issue for the particular process.
- **Call** – vector of strings containing function calls involved into the issue.
- **Function** – vector of functions involved into the issue.

The values of fields 6-10 can be different; the sizes of these vectors are equal to each other.

Figure 6.7 Details on Report Dialog

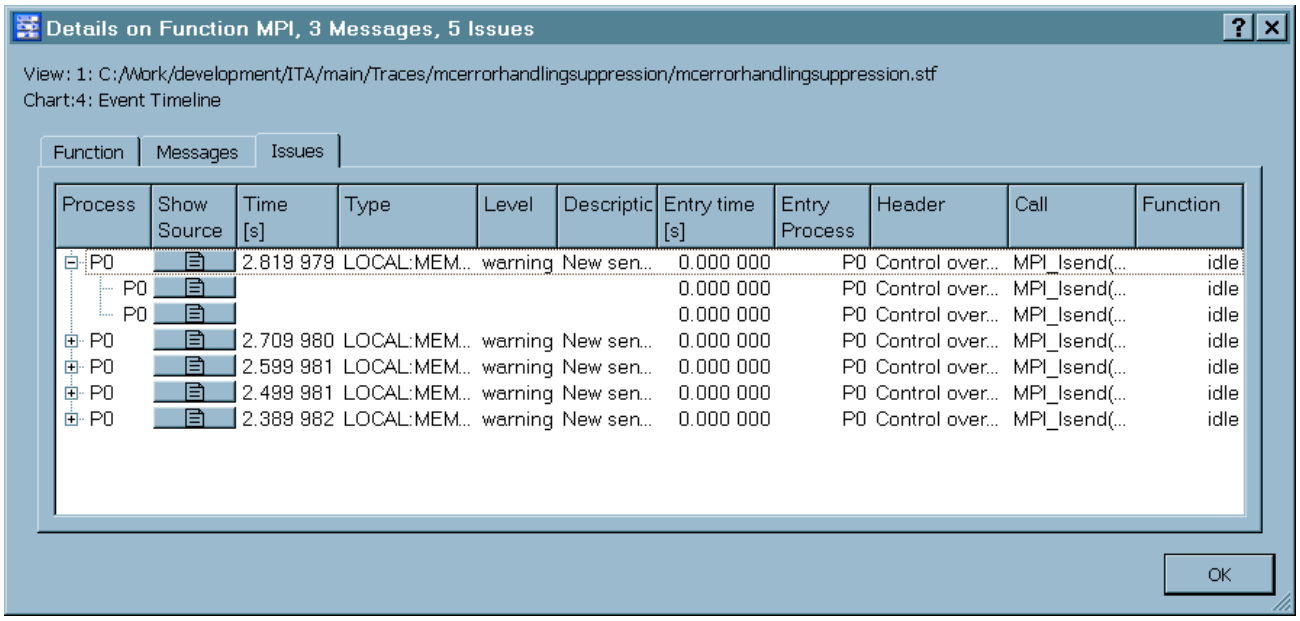
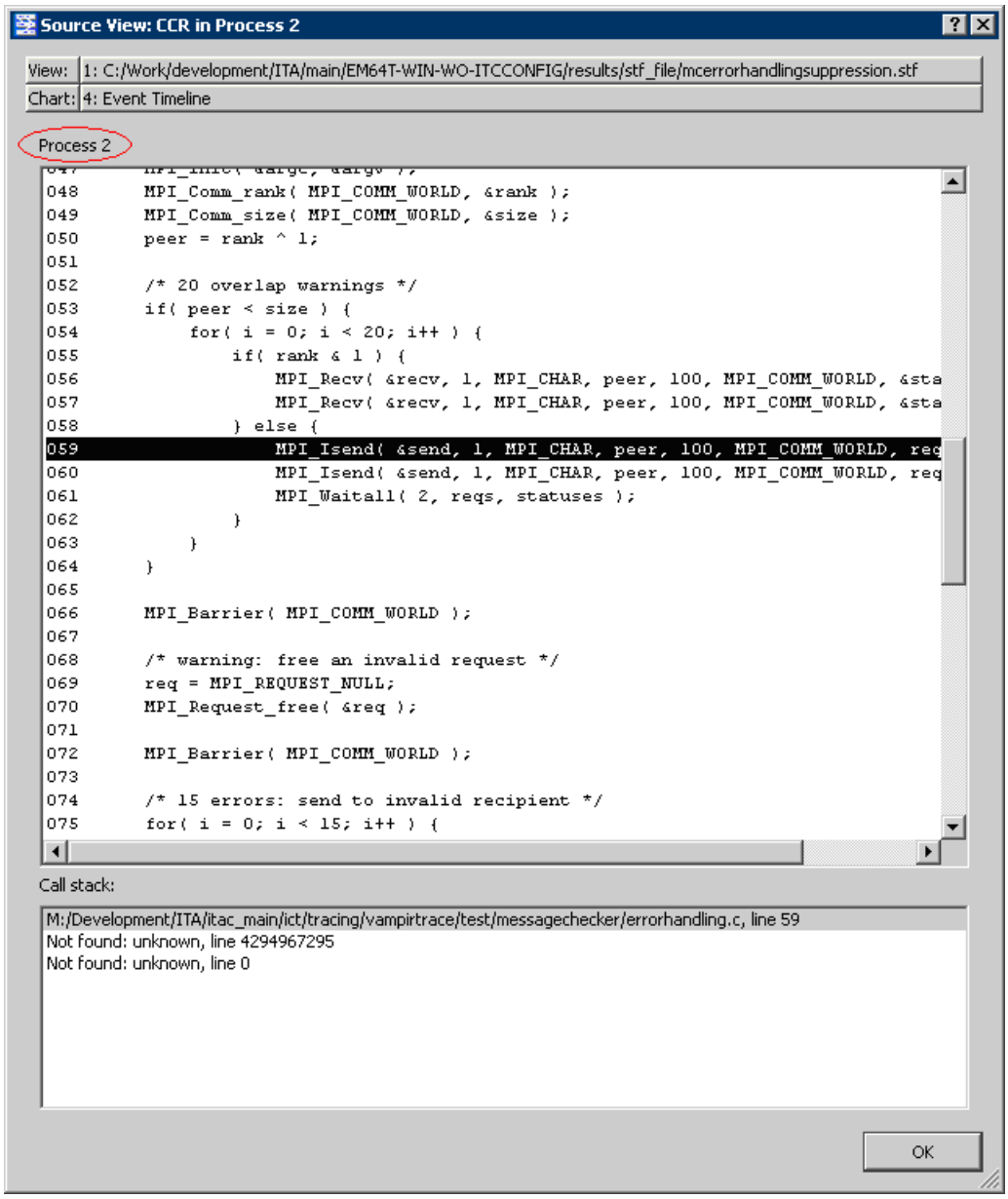


Figure 6.8 Source View Dialog



6.2 Caching

Each CCR is cached like other events (Function, Messages, Collective Operations, etc) when it is necessary. All CCRs are put into each level of the cache. The CCRs are not crowded now so each cache level contains the same information. It may lead to extra memory usage but this is unlikely since the total number of CCRs is not expected to be large.

7 Comparison of two Trace Files

The simplest way to compare two trace files or two time intervals from the same trace file is to open two Views and to look at them next to each other. While this provides a rough overview, a Comparison View allows calculating the exact differences and speedups between two runs or between two ranges of the same run. To open a Comparison View for two files, open the files and choose **View Menu > View > Compare** in one of the Views.

Choose the other View from the dialog that appears. The dialog provides the opportunity to open another file.

Figure 7.1 Comparison View

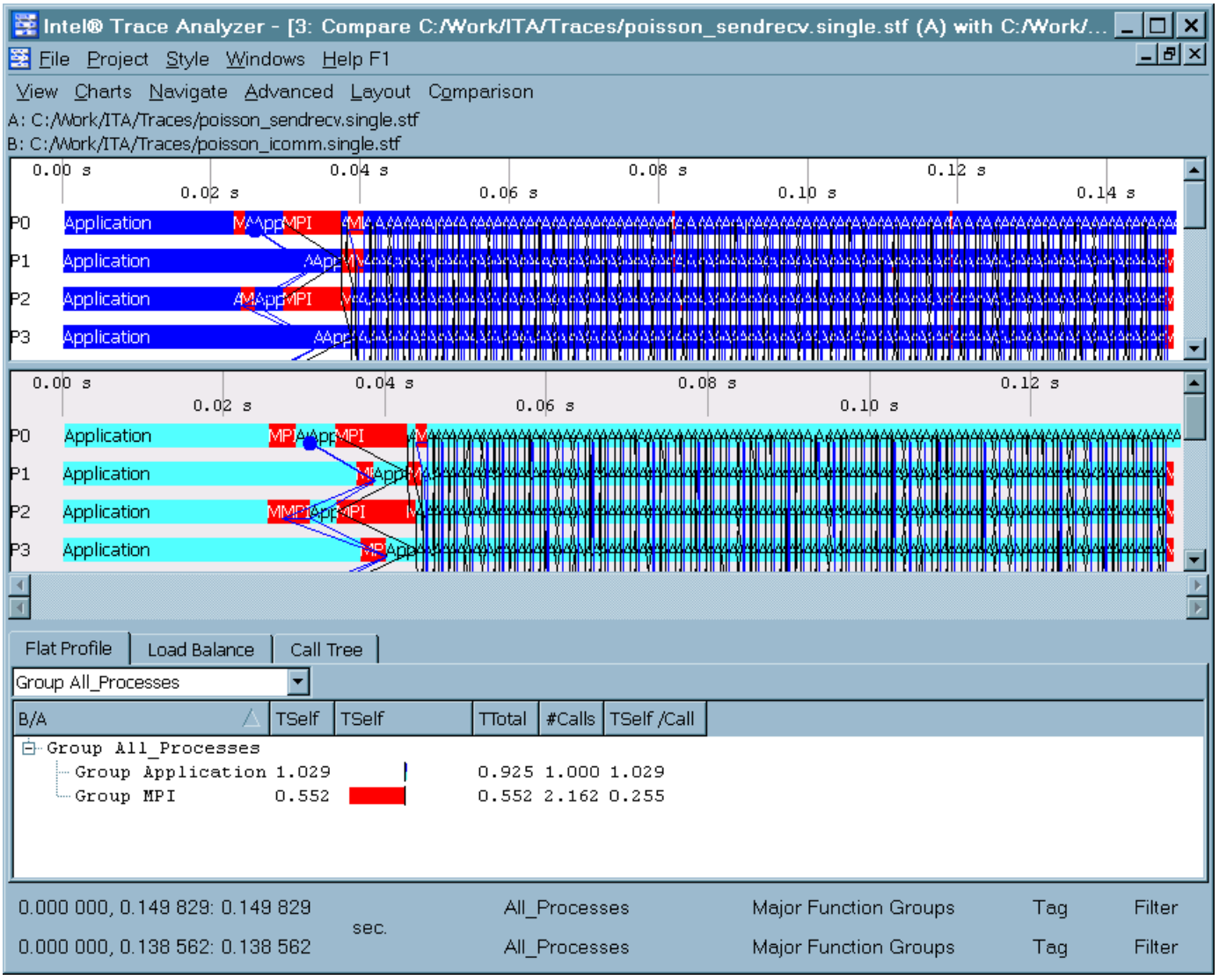


Figure 7.1 shows a comparison View. The two labels right below the View's menu bar indicate which trace files are shown in the Comparison View. These files are denoted as file A and B in the entire Comparison View.

If you move the mouse pointer into the View's status bar you will notice that it now consists of two lines: one for file A and another for file B. The labels and controls for file B are shaded as are the charts that refer to file B only.

NOTE: The comparison View inherits the time interval, aggregation and filter settings from the normal Views that are chosen to create the comparison View from.

It is perfectly valid to create a comparison View from two regular Views showing the same file. This allows you to compare either different time intervals or different subsets of processes.

When creating a comparison View, three Charts open by default: an Event Timeline each for A and B and a Comparison Function Profile.

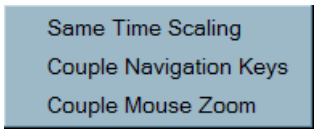
The rules that a View imposes on its Charts, namely that it enforces the same time interval, aggregation and filters on the Charts are extended for Comparison Views. The Comparison View holds two sets of time interval, aggregation and filters, one for each file.

All Charts that were described in Charts so far can only show data from a single trace file. This will stay the same and they will be tied exactly to one set of the constraints. If you choose **View Menu > Charts > Qualitative Timeline** in a comparison View then actually two timelines will appear, one for each file.

Until now a Comparison View does not provide a really striking advantage over just having two regular Views side by side. Additional benefit comes in when you open one of the now available Comparison Charts. The **Charts** menu of a Comparison Chart contains new comparing variants of the profiles that will calculate differences and speedups between the two runs. These new Chart variants are explained in Comparison Charts. In a nutshell they provide the same displays as the usual profiles but they can calculate values for A-B, B-A, A/B and B/A.

An additional menu entry **View Menu > Comparison** provides some control over the Comparison View behavior.

Figure 7.2 Comparison Menu



View Menu > Comparison > Same Time Scaling is switched off by default. Selecting this option causes all timelines to use the same scaling. For example, when you look at 2 seconds of file A and 4 seconds of file B the timelines for A are shortened so that they occupy half the width of the timelines for B to allow for easier visual comparison.

If however the time intervals for A and B differ by more than a factor of hundred, this setting is ignored and the timelines are aligned as usually to avoid numeric exceptions and distorted diagrams.

View Menu > Comparison > Couple Navigation Keys is switched off by default. This switch controls the behavior of the navigation keys in comparison mode. When switched off, the result of pressing a navigation key such as the right arrow depends on the Chart that currently has the focus. If the Chart belongs for example to file A then the time interval for A will change. If this switch is on then all navigation keys will have an effect on both of the zoom stacks (refer to Zoom Stack).

View Menu > Comparison > Couple Mouse Zoom is switched off by default. If switched on zooming with the mouse in a timeline that belongs to file A will zoom to a corresponding time interval in file B and vice versa.

7.1 Mappings in Comparison Views

The proverbial error in doing any comparison is to compare apples and oranges. When comparing two program runs this could be to compare the time that a process A.P0 spent in a function in run A to the time of another process B.P0 in run B with or without caring for the fact that B.P0 did only half of the work because run B used twice as much processes.

It is quite easy to see that depending on the domain decomposition or load balancing that is done in the application the meaningful mapping between the processes of two runs can not be determined automatically. There might be even no such mapping: imagine comparing a run that did a domain decomposition of a cube into 8 processes 2x2x2 with a run that used a 3x3x3 decomposition.

But functions and function groups can be mapped between the runs just by their fully qualified name. This works as long as the structure of modules, namespaces and classes is not changed dramatically.

It is next to impossible to even enumerate all combinations of parameters that might have changed between two runs. To foresee all these cases in terms of automatically adapting GUI does not look promising.

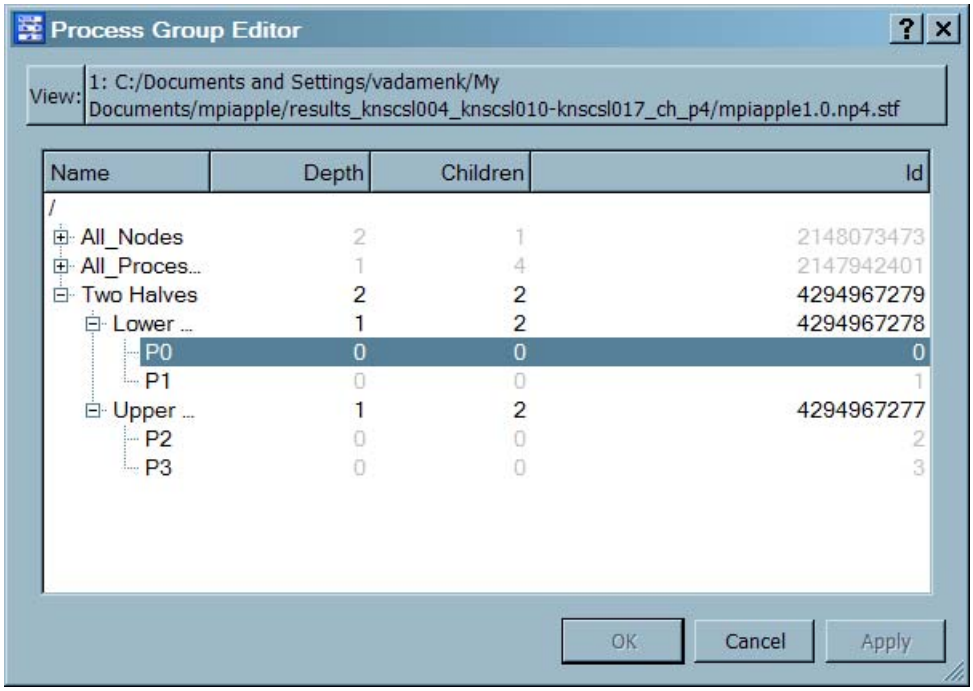
Based on these considerations the mappings of processes, functions, communicators and message tags between the runs are handled differently:

Communicators are mapped by their Ids. Message tags are mapped literally by their value.

The mapping of processes and process groups is controlled by choosing the process aggregations for both files as outlined in Mapping of Processes.

The mapping between functions and function groups is handled automatically as outlined in Mapping of Functions.

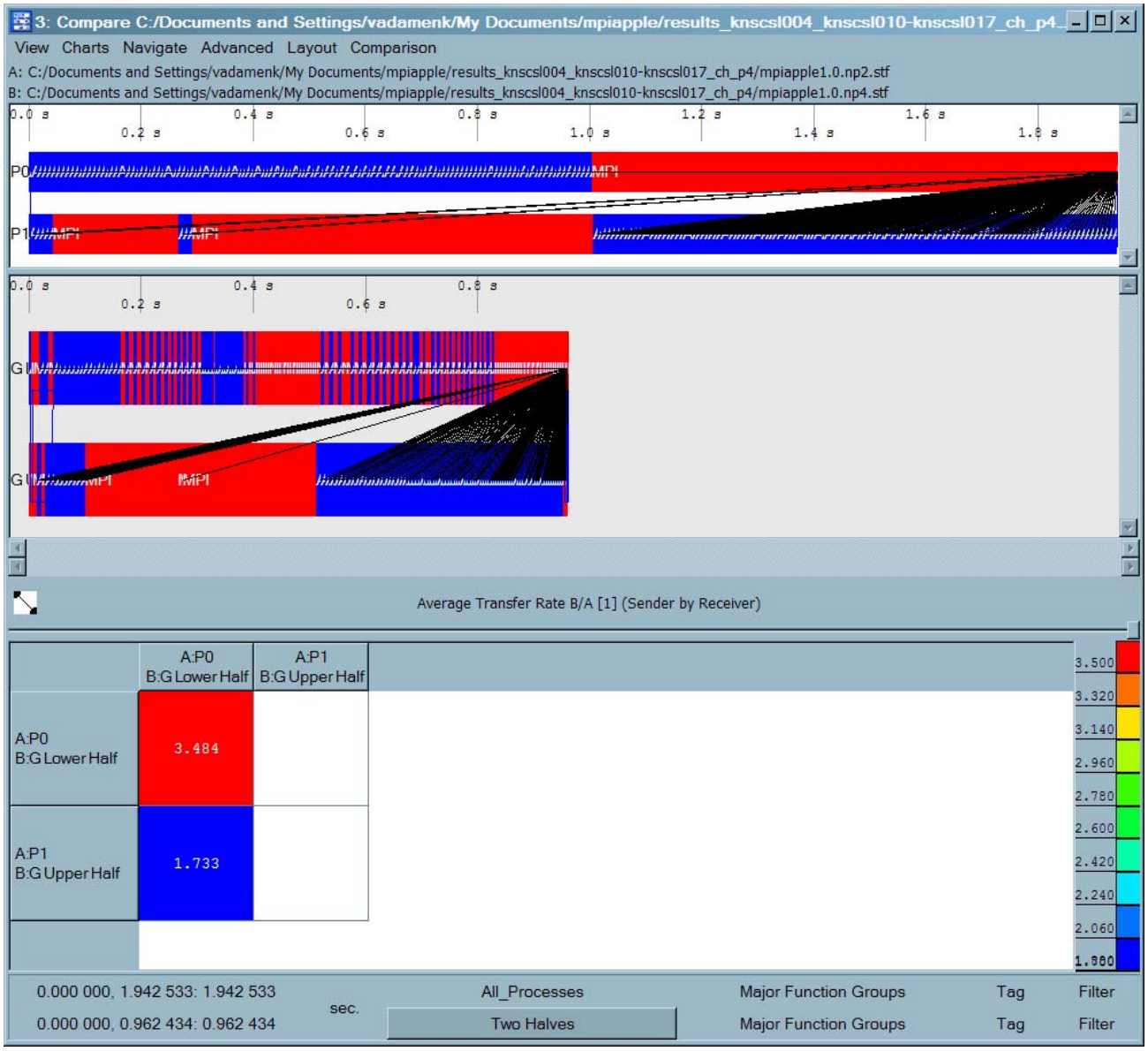
Figure 7.3 Creating a Suitable Process Group for the Comparison between a 2 and a 4 Processor Run in the Process Group Editor



7.1.1 Mapping of Processes

Assume that run A had A.P0, A.P1 and run B had B.P0, B.P1, B.P2, B.P3 and assume that A.P0 did the same work as B.P0 and B.P1 and A.P1 did the same work as B.P2 and B.P3. To get a Comparison Message Profile that is meaningful under these assumptions choose the aggregation as shown in *Figure* 7.3 and *Figure* 7.4. Here for the run B process aggregation into two halves was chosen.

Figure 7.4 Comparing Run A with 2 Processes to Run B with 4 Processes



The message profile shows the quotient B/A of the average transfer rate. The rule that the Comparison Message Profile (and in fact the whole Comparison View) uses to map the senders and receivers of the two runs onto each other is quite simple: child number i of run A's process aggregation is always compared with (mapped to) child number i of run B's process aggregation.

7.1.2 Mapping of Functions

Functions of the two files A and B are mapped onto each other by their fully qualified name. This name contains not only the mere function name, but a hierarchical name that is constructed by the Intel® Trace Collector using any information about modules, name spaces and classes that was available at trace time.

For example the fully qualified name of MPI_Allreduce will be MPI:MPI_Allreduce because Intel Trace Collector puts all MPI functions into the group MPI. Function groups that were defined by the user in Intel Trace Analyzer have no influence on these full function names. The function group editor described in Function Group Editor shows the fully qualified name of a function in a small tooltip window when the mouse hovers long enough over an entry.

The mapping of function groups is a little more subtle. For function groups that are within the hierarchy of the automatically created function group **Major Function Groups** in file A it is tried to find a matching group in B with the same name and nesting level in the corresponding hierarchy in B.

For automatically generated groups this works quite good. For example MPI is always mapped to MPI even if the groups differ because the two program runs did use a different subset of MPI calls. The same is true for groups that were created by instrumentation using the API provided by Intel® Trace Collector.

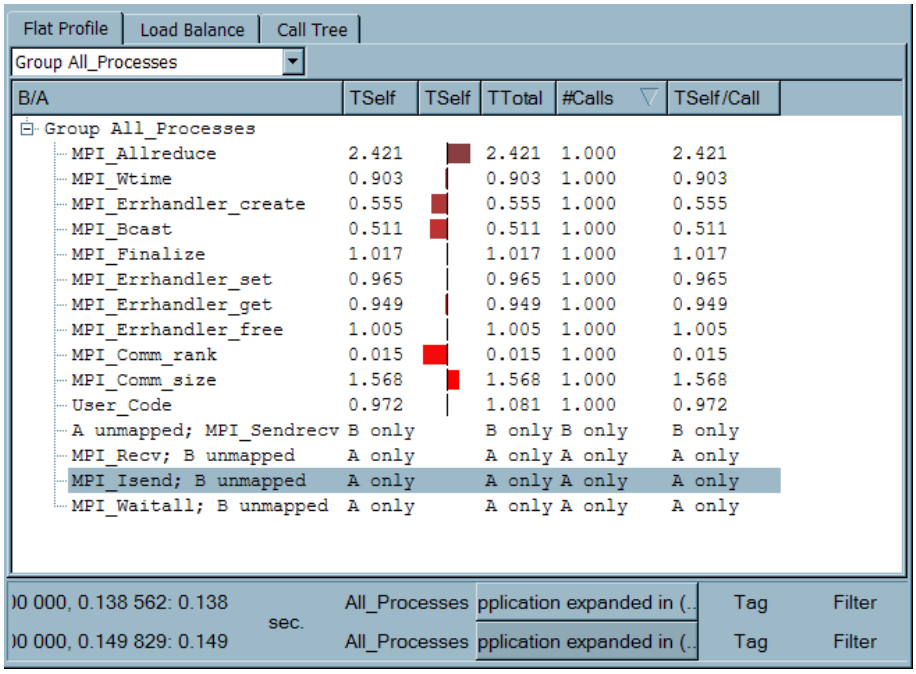
When you create new function groups either by using the Function Group Editor or the ubiquitous context menu entries to ungroup existing function groups for one file then there will be created matching groups for the other file. You can find these read only groups under the header **Generated Groups** in the Function Group Editor.

7.2 Comparison Charts

7.2.1 Comparison Function Profile

The Comparison Function Profile is very similar to the regular Function Profile. It does not have the pie diagrams in the **Load Balance** tab and no Call Graph tab at all.

Figure 7.5 Comparison Function Profile Chart



The column headers are the same as in the regular Function Profile with the exception of the first column. In the Comparison Function Profile this column header is used to display the currently selected comparison operation. This operation can be selected from the context menu.

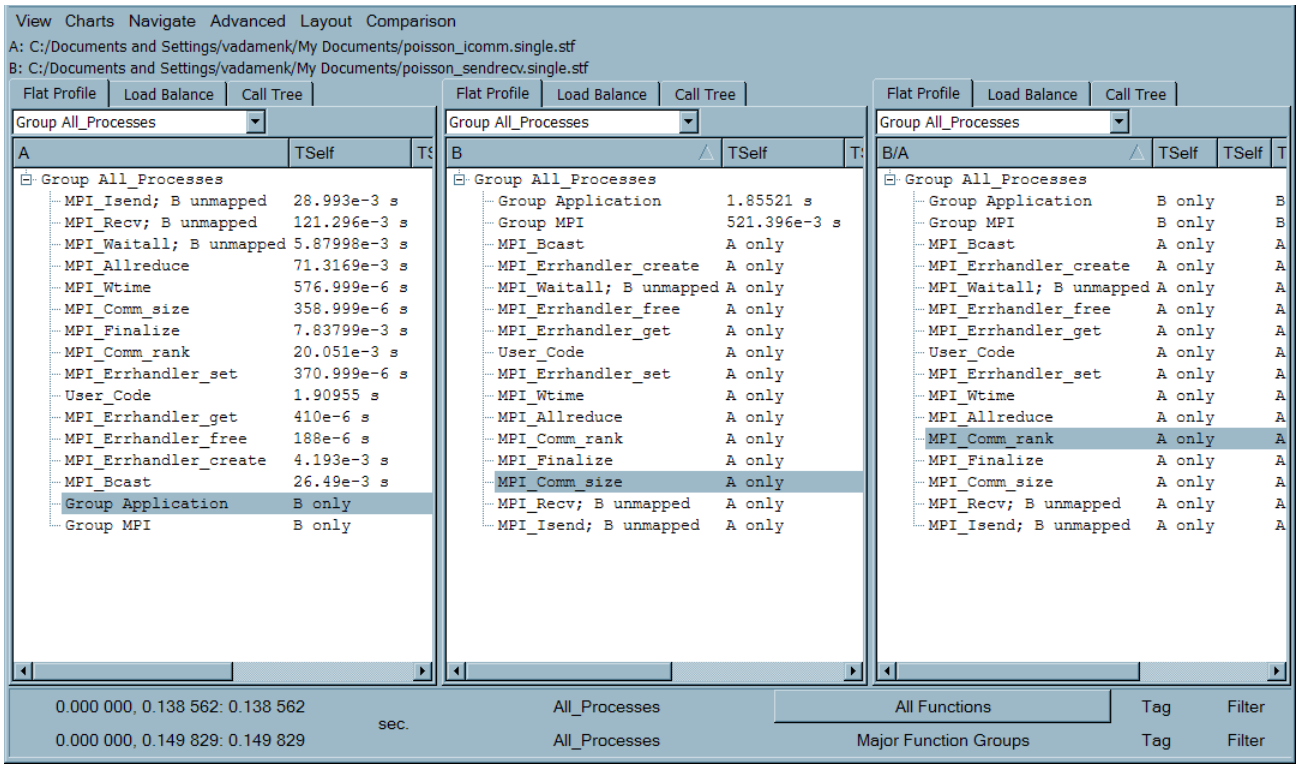
Wherever the regular Function Profile shows the name of a function, function group, process, or process group the Comparison Function Profile shows either a pair of mapped names formatted as “NameA; NameB” if the names are different, “A unmapped; NameB” or vice versa if there is no mapping for one file or just as “Name” if the names are equal for both files.

Figure 7.5 shows a case where MPI_Sendrecv in A was replaced by `MPI_Isend`, `MPI_Recv` and `MPI_Waitall` in B. The first column of the profile flags which functions are present in which trace. The

other columns indicate for which of the traces there was valid data. When there is only valid data from one file, no meaningful comparison can be done.

Even for a function **F** that is present in both traces the currently selected time intervals (or aggregations, or filters) can have the effect that there is no data from one of the traces. In this case the first column would carry the label **F** since the mapping is ok but the other columns would have **A only** or **B only** respectively.

Figure 7.6 Undefined Fields in the Profile due to the Chosen Aggregations



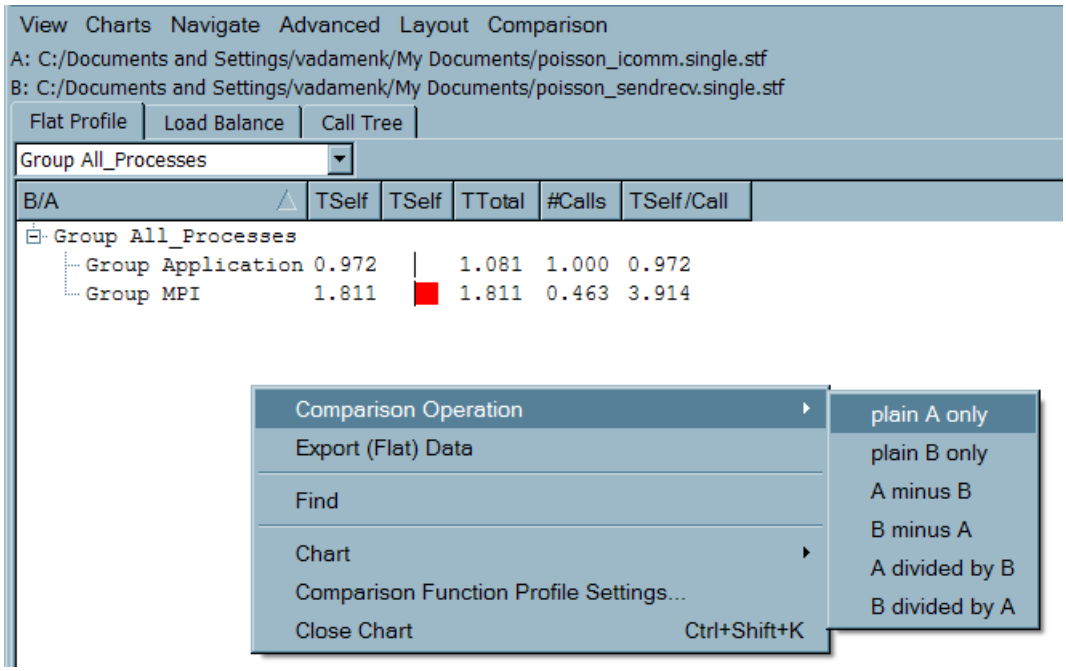
If you choose **Major Function Groups** and **All Functions** for the function aggregation in both runs then all fields will show remarks indicating that no comparison could be made because of lacking data – at least as long as the current comparison operation needs data from A and B. If you switch to a comparison operation of A or B then the respective fields will show values even when the function aggregations do not fit. *Figure* 7.6 shows such a case: there is data for run A and B but in the rightmost profile there is nothing to calculate the comparison operation B/A on.

The horizontal bars in the graphical columns work similar to the regular Function Profile except when the comparison operation is set to A/B or B/A. In this case the length of the bars is proportional to the logarithm of the given value. A value of for example 1/5 results in a bar from the middle to the left, while a value of 5 results in a bar of the same length but from the middle to the right. So switching between A/B and B/A will mirror the bars around the middle. Please note that the full column width shows at least the range from 1/2 to 2 so that factors near to 1 do not generate big bars that would invite to misinterpret them as very big changes between the two runs.

The context menu of the Comparison Function Profile is pretty much the same as the one found in the regular Function Profile (refer to Context Menu) with the addition of a submenu to choose the comparison operation. See below.

The **Settings** dialog of the Comparison Function Profile is the same as the one for the regular Function Profile (refer to Function Profile Settings) with an additional tab that allows to switch the comparison operation.

Figure 7.7 Comparison Function Profile Context Menu with the Available Comparison Operations



7.2.2 Comparison Message Profile

The Comparison Message Profile is very similar to the regular Message Profile. The values shown in the cells are calculated using the currently selected comparison operation shown in the title of the Chart. The comparison operation can be switched from the context menu or from the settings dialog. If the comparison operation makes use of A and B and only one of the values is present for a given cell, then this cell is labeled with **A** or **B** indicating for which trace data is present. If there is no data at all, the cell is left empty.

Some groupings are dependent on the mapping rules in which they either use processes, process groups or communicators as row or column labels. If differing labels are mapped onto the same column or row then the label is made up of two lines with the first line holding the label for file A and the second for file B. This can happen for the groupings **Sender**, **Receiver**, **Sender/Receiver** and **Communicator**. In *Figure* 7.8 the process aggregation is **All_Processes** for one file and **All_Nodes** for the other which results in double line labels in the profile.

Figure 7.8 shows a Comparison Message Profile with `poisson_sendrecv.single.stf` as run A and `poisson_icomm.single.stf` as run B.

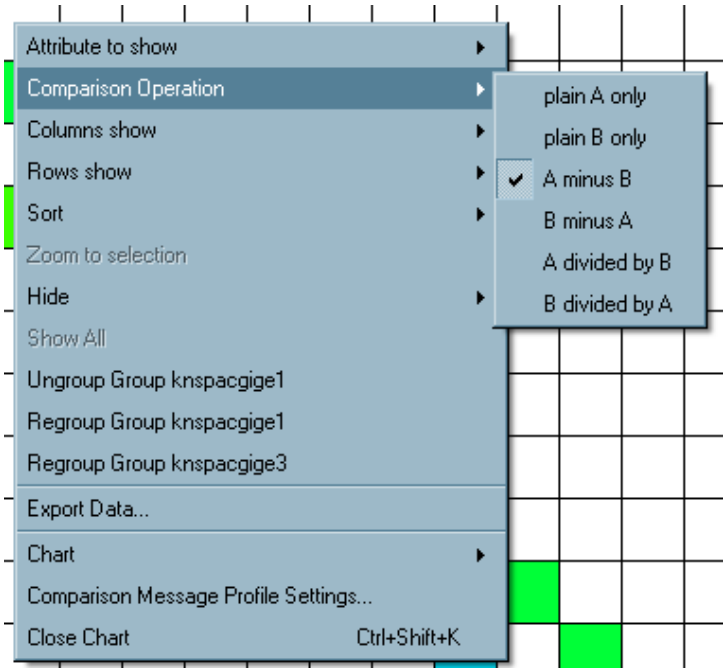
Figure 7.8 Comparison Message Profile

There can be cases where a whole row or column does not apply to one of the traces. In these cases the double line label holds **A missing** or **B missing**. This can happen for the groupings **Sender**, **Receiver** and **Sender/Receiver** that may have unmappable TGroups in the labels or for groupings like **Tag**, **Communicator** or **Volume** that directly depend on the data that is present in the trace. The **Settings** dialog box of the Comparison Message Profile has means to suppress such rows and columns, but they are shown by default.

The Context Menu of the Comparison Message Profile is pretty much the same as in the normal Message Profile (refer to Context Menu), with the difference that it has an additional entry, **Comparison Operation**, where the type of comparison can be selected from a submenu.

The **Settings** dialog box of the Comparison Message Profile is pretty much the same as for the normal Message Profile (refer to Message Profile Settings). In the **Preferences** tab there are two additional check boxes to suppress rows or columns that only apply to one trace and the **Data** tab has an additional combo box to choose the comparison operation.

Figure 7.9 Available Comparison Operations

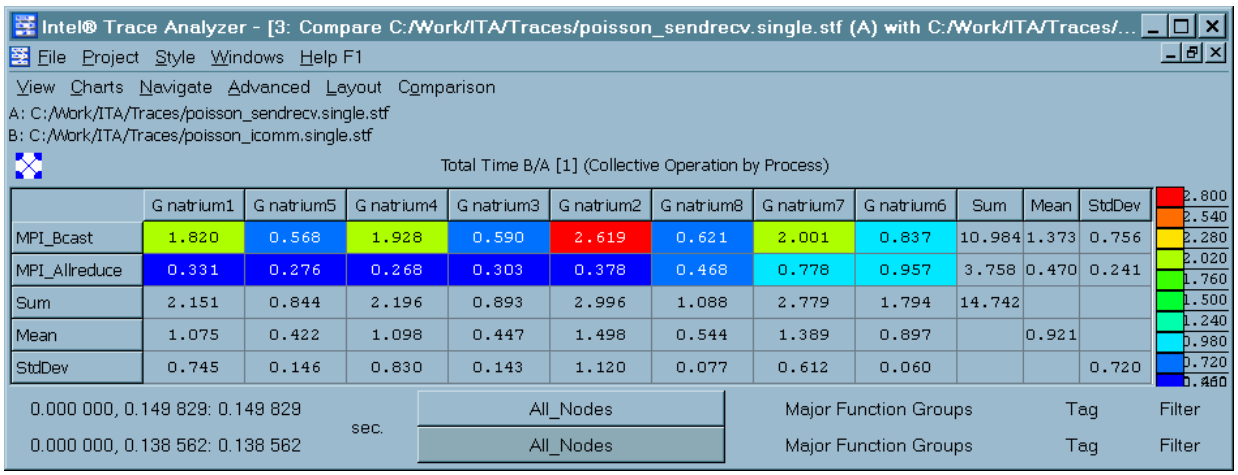


7.2.3 Comparison Collective Operations Profile

The Comparison Collective Operations Profile is very similar to the regular Collective Operations Profile. The values shown in the cells are calculated using the currently selected comparison operation shown in the title of the Chart. The comparison operation can be switched from the context menu or from the settings dialog.

Regarding missing values and unmappable column or row labels the Comparison Collective Operations Profile behaves identically to the Comparison Message Profile (refer to Comparison Message Profile).

Figure 7.10 Comparison Collective Operations Profile



The Context Menu of the Comparison Collective Operations Profile is pretty much the same as in the normal Collective Operations Profile (see Context Menu), with the difference that it has an additional entry, **Comparison Operation**, where the type of comparison can be selected from a submenu.

The **Settings** dialog box of the Comparison Collective Operations Profile is pretty much the same as for the normal Collective Operations Profile (refer to Collective Operations Profile Settings). In the **Preferences** tab there are two additional check boxes to suppress rows or columns that only apply to one trace and the **Data** tab has an additional combo box to choose the comparison operation.

8 Command Line Interface (CLI)

The Command Line Interface to Intel® Trace Analyzer allows you to process trace files without a GUI. It can be used for automated computation of profiling data or to generate pre-computed trace caches for trace files.

To use the CLI, you must use `--cli` as the first argument to switch off the graphical user interface followed by a trace file name and any other CLI options.

For example, to create the cache for `trace.stf` with default resolution, enter the following command line:

```
traceanalyzer --cli trace.stf -c0 -w
```

A batch file to pre-compute caches might look like this:

```
traceanalyzer --cli poisson_icomm.single.stf -c0 -w
traceanalyzer --cli poisson_sendrecv.single.stf -c0 -w
traceanalyzer --cli vtcounterscopec.single.stf -c0 -w
```

NOTE: The CLI is for expert use and can be changed with any version without notice.

The command line interface provides the following options:

Table 8.1 CLI Options

Option Name	Action
`--messageprofile`	Perform message profile analysis.
`--collopprofile`	Perform collective operation profile analysis.
`--functionprofile`	Perform function profile analysis.
`--starttime=TICKS` or `-sTICKS`	Starting time of the analysis.
`--endtime=TICKS` or `-eTICKS`	Ending time of the analysis.
`--tgroup=ID` or `-tID`	Use this thread aggregation.
`--fgroup=ID` or `-fID`	Use this function aggregation.
`--dump=FILE` or `-oFILE`	The file where to store the analysis results.
`--funcformat`	A string that contains format switchers specifying how the information about functions are printed; the default value is `TFNEIS.` Possible format options: • `f` or `F` - prints the name of the function group • `t` or `T` - prints the name of the thread/process group • `g` or `G` - prints the number of processes/threads in the group • `E` - prints self time in ticks • `e` - prints self time in seconds • `I` - prints total time in ticks • `i` - prints total time in seconds • `n` or `N` - prints the number of calls • `s` or `S` - prints the source code location (if possible)

Option Name	Action
--messageformat	A string that contains format switchers specifying how the information about point-to-point messages is printed; the default value is 12DdIiXxAauUn. Possible format options: 1 - prints if the first member of the message is sender and/or receiver 2 - prints if the second member of the message is sender and/or receiver D - prints the summary duration in ticks d - prints the summary duration in seconds v or V - prints the summary amount of bytes sent k or K - prints the minimum amount of bytes sent l or L - prints the maximum amount of bytes sent U - prints the minimum duration in ticks u - prints the minimum duration in seconds X - prints the maximum duration in ticks x - prints the maximum duration in seconds I - prints the minimum rate in Bytes/tick i - prints the minimum rate in Bytes/second A - prints the maximum rate in Bytes/tick a - prints the maximum rate in Bytes/second n or N - prints the number of messages
--collopformat	A string that contains format options specifying how the information about collective operations is printed; the default value is 12DdIiXxAauUnvwyzlk. Possible format options: 1 - prints the name of the process group 2 - prints the name of the operation D - prints the summary duration in ticks d - prints the summary duration in seconds U - prints the minimum duration in ticks u - prints the minimum duration in seconds X - prints the maximum duration in ticks x - prints the maximum duration in seconds I - prints the minimum rate in Bytes/tick i - prints the minimum rate in Bytes/second A - prints the maximum rate in Bytes/tick a - prints the maximum rate in Bytes/second v or V - prints the summary amount of bytes sent k or K - prints the minimum amount of bytes sent l or L - prints the maximum amount of bytes sent w or W - prints the summary amount of bytes received y or Y - prints the minimum amount of bytes received z or Z - prints the maximum amount of bytes received n or N - prints the number of collective operations
--readstats or -S	Request statistics, if available, instead of trace data.
--readcache[=FILE] or -r[FILE]	Read the trace cache from the specified (if provided) or default file.

Option Name	Action
--writecache[=FILE] or -w[FILE]	If a trace cache has been built, write it to the specified (if provided) or default file.
--buildcache=RESOLUTION or -cRESOLUTION	Build a trace cache with the specified resolution. The resolution is given in clock ticks. Higher values result in smaller (coarser) cache files, 0 (zero) is used as the default resolution.
--filter=FILTER or -FFILTER	The filter to use for the analysis, specified as a filter grammar string.
--messagefirst=GROUPING	The first grouping in the message profile analysis result (first dimension of matrix).
--messagesecond=GROUPING	The second grouping in the message profile analysis result (second dimension of matrix).
--collopfirst=GROUPING	The first grouping in the collective operation profile analysis result (first dimension of matrix).
--collopsecond=GROUPING	The second grouping in the collective operation profile analysis result (second dimension of matrix).
--summary	Generate the application summary sheet with the format that is described below.
--icpf [options] <tracefile> --simulator <simulator libraray>	Process a trace file using the specified simulator at runtime. Use the traceanalyzer –icpf option to process your trace files using specific simulator library. In --icpf [options] <tracefile> --simulator <simulator libraray>, the [options] can be: -s *<NUM>* - processes the trace starting at the time (NUM measured in ticks). -e *<NUM>* - processes the trace to the end time (NUM measured in ticks). -w *<NUM>* - processes the trace based on NUM, 0 for STF, 1 for ASCII, else devnull. -o *<new_name>* - trace output file name. -u - single file mode. The output file is a single STF. -h - prints this message and exits.
--ideal [options] <tracefile>	Produce an ideal trace. Use the traceanalyzer --ideal option to idealize a trace by Ideal Interconnect Simulator. In --ideal [options] <tracefile>, the [options] can be: -s *<NUM>* - processes the trace starting at the time (NUM measured in ticks; the default value is 0). -e *<NUM>* - processes the trace to the end time (NUM measured in ticks; the default value is the end time of the trace). -w *<NUM>* - processes the trace based on NUM, 0 for STF, 1 for ASCII, else devnull (the default value is 0). -o *<new_name>* - trace output file name. -u - single file mode. The output file is a single STF. -sp – shows percent progress indicator. -q - quiet mode; turns off all output. -h - prints this message and exits.
--breakdowns <real_trace_name> <ideal_trace_name>	Create intermediate *.bdi files that contain all needed information for the Application Imbalance Diagram.

The application summary sheet consists of a three-line header:

```
<# processes>:<# processes per node>
<application time>:<MPI time>:<IIS time>
<first message size of middle bucket (2)>: \
<first message size of highest bucket (3)>
```

The header is followed by these sets of lines, for each of the top ten functions, sorted by descending total time:

```
<Name of MPI_group>:<# involved processes>
<total time in above func for bucket 1>:<for bucket 2>:<for bucket 3>
<total IIS time in above func for bucket 1>:<for bucket 2>:<for bucket 3>
<count in above func for bucket 1>:<for bucket 2>:<for bucket 3>
<total # bytes in above func for bucket 1>:<for bucket 2>:<for bucket 3>
```

In the application summary sheet, IIS stands for Ideal Interconnect Simulator, which predicts MPI behavior on an ideal interconnect.

You can import the application summary sheet to spreadsheet applications such as Microsoft* Office Excel*. Fields are separated by colons. Unknown values are indicated by **N/A**.

9 Custom Plug-in Framework

The Custom Plug-in Framework (CPF) enables you to write your own simulators using the provided API. For example, Ideal Interconnect Simulator (IIS) is implemented using this framework. Custom developed simulators can interact with trace files from Intel® Trace Analyzer and can model any condition you can think of.

CPF implementation works with traces that were obtained from applications that pass correctly message checking. To use CPF, install Intel® Professional Edition Compilers on the following systems:

- the system in which you are building your simulator
- the system in which you are running CPF

9.1 Usage Instructions

Use the `traceanalyzer –icpf` option to process your trace files through a specific simulator library. You can use the following command line:

```
traceanalyzer --cli --icpf [options] <tracefile> --simulator <simulator library>
```

where

- `<tracefile>` is the name of your trace file
- `<simulator library>` is the name of your simulator.

9.2 Developing Simulators

The Trace Analyzer installation includes an examples section with an example simulator. You can create your own copy of the example and customize it to create your own simulator.

Follow these basic steps to create your own simulator library:

1. Set up the development environment
2. Writing your simulator

9.2.1 Setting up the Development Environment

An example of source files for developing simulators is in:
<trace_analyzer_install_dir>/examples/icpf

In this directory, you can only modify the following files:

- `h_devsim.cpp`
- `Makefile` (or `Makefile_win` for windows)

To set up the development environment, do the following steps

1. Rename the `devsim.cpp` file to a meaningful name for your simulator, with the suffix `sim.cpp`. The renamed file is used for next step.

NOTE: Rename all devsim.cpp files.

2. Modify the Makefile in the simulator development directory. Open Makefile with any text editor. Edit the following lines:

```
Makefile:

SIMCOMP = icc

SIMFLAGS = -shared -g -fPIC -I ./include

LIBSUFFIX = .so

RM = rm -f

devsim: h_devsim.cpp

        $(SIMCOMP) $(SIMFLAGS) -o devsim$(LIBSUFFIX) h_devsim.cpp

clean:

        $(RM) devsim$(LIBSUFFIX)

Makefile_win:

SIMCOMP = icl

SIMFLAGS = /LD /I ./include /D _WINDLL

LIBSUFFIX = .dll

RM = del

devsim: h_devsim.cpp

        $(SIMCOMP) $(SIMFLAGS) -o devsim$(LIBSUFFIX) h_devsim.cpp

clean:

        $(RM) devsim.*, *.obj
```

   Change every occurrence of the .so shared library name in the makefile from devsim.so (or devsim.dll for Windows OS*) to your simulator name + .so (or .dll).

   Close the Makefile.

3. Edit the simulator file. Open your renamed simulator .cpp file in any text editor. Locate the following line near the top (line 24):

```
class DevSim : public CustomPluginFramework
```

   1. Change DevSim to be the name of your simulator and your class.

   2. Locate the extern C function, near the bottom of the file. Edit the following line (line 74):

```
return new DevSim;
```

   Change DevSim to the name you choose for your class/simulator.

After setting up the development environment, you can write your simulator.

9.2.2 Writing your Simulator

Before beginning simulator development, you need to understand the virtual class that is inherited by a simulator.

The CustomPluginFramework.h header file connects between the Intel Trace Analyzer source and other tools such as simulators. This file contains the following information:

- data structures that are passed to the API
- public virtual functions that are called by CPF
- private functions that can be called by any simulator help/aid with simulator development
- private members that provide a simulator with access to certain aspects about a particular trace

NOTE: If you change the file, you will not be able to plug a simulator into the API.

You can create your simulator filling virtual functions in h_devsim.cpp (or your .cpp file if you have renamed it).

Examples

1. In this example, durations of all non-blocking sending MPI events in trace file become zero time.

```
         virtual void ProcessNonBlockingSend( FuncEventInfo * _event_, P2PInfo *
_message_ ) {

        _event_->_endTime = _event_->_startTime;

  }
```

2. In this example, durations of all collective operations are doubled.

```
  virtual void ProcessCollOp( CollOpInfo * _collop_ ) {

        //Traverse vector of CollOpInfo

        for( U4 k = 0; k < _collop_->_threads->count(); k++) {

            (*(_collop_->_threads))[k]->_endTime += ( (*(_collop_-
>_threads))[k]->_endTime - (*(_collop_->_threads))[k]->_startTime );

        }

    }
```

If you need to add more inline functions for your simulator, add them only to h_devsim.cpp or your .cpp file. Do not modify the h_custompluginframework.h header file.

NOTE: When modifying end time, ensure that the end time you change is never less than the start time.

NOTE: All values of time are stored in a simulator in ticks instead of in seconds. To convert ticks in seconds, use _clockResolution coefficient.

NOTE: When you have trouble with trace, it is suggested that you run the trace through Intel Trace Analyzer and Collector message checker before filling a bug report.

9.2.3 CPF Simulator API

CPF interacts with the simulators through the custom plug-in framework API. The API provides pre-defined functions that the simulators have to implement.

9.2.3.1 Framework Variables

Table 9.1 Framework Variables

Framework Variables	
U4	unsigned 4 byte integer
U8	unsigned 8 byte integer
I4	signed 4 byte integer
I8	signed 8 byte integer
F4	4 byte floating point
F8	8 byte floating point

9.2.3.2 Data Structures

FuncEventData

The `FuncEventData` structure represents any function events that occur in a trace on a specific thread and do not span multiple threads. CPF passes a `FuncEventData` structure to a simulator whenever it encounters any application event, MPI single rank event, or a non-tracing event (such as idle time on a process). The `FuncEventData` structure also represents a non-blocking sending because a non-blocking sending occurs on only a specific thread.

The only member of `FuncEventData` that you can modify using a simulator is the end time of the function event. The other parameters are all constant because modifying any of these values would drastically change the fundamental parts of the trace. The constant parameters can be the thread id, function type, size of message (if it is a non-blocking sending), or start time.

NOTE: The end time of this event should never be less than the start time. CPF checks this rule when returned from the simulator.

Table 9.2 Variables of FuncEventData

Variables	Explanation
`_threadID`	The thread id where this function event occurred.
`_func`	The function id of this event. You can look up the id using mapping provided. See Private Functions for details.
`_sizeSent`	The size of the message sent by the thread. This variable is available when the function event is used as part of the collective operation.
`_sizeRecv`	The size of the message recieved by the thread. This variable is available when the function event is used as part of the collective operation.
`_startTime`	The start time of this function event.
`_endTime`	The end time of this function event. You can modify this variable, but make sure that it is not less then `_startTime`.

P2PInfo

The `P2PInfo` data structure represents a point-to-point (P2P) message that occurs in a trace. It is also passed to non-blocking API function to give the simulator access to different aspects of the point-to-point message. The values are slightly different depending on the type of point-to-point message that this data structure represents. See Private Functions for details.

The following table lists the P2Pinfo variables. All variables, except the end times of the send and receive messages, are constant because modifying any of these values would drastically change the fundamental parts of the trace.

- the thread id of both the sending and receiving threads
- start times of the sending and receiving threads
- function ids of the sending and receiving threads
- i-receive start time (if applicable)
- communicator
- tag
- message size

Table 9.3 Variables of P2PInfo

Variables	Explanation
_sendStartTime	This variable contains the send start time or the time when the message is sent.
_sendEndTime	This variable is only valid on • sendrecv's • non-blocking sends (only in processnonblockingsend()) • sync sends This variable can be modified, but should never be less then _sendStartTime.
_recvStartTime	This variable is only valid on sendrecv's, sync sends, waitall/waitsomes.
_recvEndTime	This variable is valid on all cases, can be modified, but should never be less then _recvStartTime.
_iRecvStartTime	This variable is only valid when P2P message involves a waitall/waitsome. This variable represents the start time of i-receive that occurs before the waitall/waitsome for this thread.
_backwards	This variable indicates that the message send start time > receive end time.
_sender	The thread id of the sending thread.
_receiver	The thread id of the receiving thread.
_communicator	The MPI communicator. The P2P message communication is over; there is a communicator to text mapping provided for the simulator in the API. See Private Functions for details.
_tag	The MPI tag associated with the message.
_messageSize	The size of the message sent.
_sendFID	The identifier of the sending function. You can get more information on the function using the mapping provided. See Private Functions for details.
_recvFID	The identifier of the receiving function. You can get more information on the function using mapping provided. See Private Functions for details

CollOpInfo

The CollOpInfo data structure represents a collective operation that occurs in the trace. In a trace, there is a collective operation that connects the same event on each thread in the trace to one another. This CollOpInfo data structure provides this grouping to the simulator.

All members of the CollOpInfo data structure are constant and cannot be modified except for the FuncEventData events that represent each thread that is involved in the collective operation. To modify end times of the collective operation on each thread, modify the FuncEventData for each thread in the vector in this collective operation. The same rules apply CollOpInfo for FuncEventData.

Table 9.4 Variables of CollOpInfo

Variables	Explanation
_firstTime	The earliest time that a thread enters a collective operation.
_communicator	Communicator on which collective operation is executed.
_collOpId	The function id of the collective operation. You can look up what it is using mapping provided. See Private Functions for details.
_root	The thread id of the root, valid only if the collective operation is a One-To-All or All-To-One.
_rootIndex	The root's index into the thread vector.
_threads	The thread vector that represents the individual function events per thread that make up this collective operation.

9.2.3.3 Public Functions

The following topics list the public API functions and general public functions that CPF calls. When CPF is interacting with the API, it first creates a data structure of type described in Data Structures; then passes a pointer to the simulator's API function. Any modification to this data structure in the API function automatically reflects changes in CPF. These functions do not return any values.

void setup()

Use this function to pass important information/characteristics about the trace from CPF to the simulator. This function takes care of initialization of the private member variables in Private Members. Do not modify this function.

Syntax

```
Void setup (const Vector< U4 >, const Vector< U4 >, const Vector< U4 >, const HashMap< U4, const char* >, const char*, const U8, const U8, const U4)
```

virtual void Initialize( ) = 0;

Your simulator needs to implement this API function. The function is called before the start of processing a trace file, not to be used as a constructor. This function enables simulator extensions to pre-trace file operations before the executable starts processing the trace file.

virtual void Finalize( ) = 0;

Your simulator needs to implement this API function. The function is called at the very end of processing a trace file, not to be used as a destructor. This function allows simulator extensions to have a way of doing post-trace file operations before the executable exits.

virtual void ProcessFunctionEvent(FuncEventInfo * _event_) = 0;

Your simulator needs to implement this API function. This function processes application events that occur in the trace file. The event object is a regular application event in the trace that is processed from the queues.

virtual void ProcessNonTracingEvent(FuncEventInfo * _event_) = 0;

Your simulator needs to implement this API function. This function processes non-tracing events that occur in the trace file. This event object is a non tracing event in the trace that is to be processed from the queues. A non-tracing event is an empty white space at beginning of trace before each thread, or parts of a trace file which have tracing turned off.

virtual void ProcessMPIFunctionEvent(FuncEventInfo * _event_) = 0;

Your simulator needs to implement this API function. This function processes a single rank MPI event in trace file. This event object is a single rank MPI event in the trace that is to be processed from the queues. A single rank MPI event is an MPI event that only involves one thread.

virtual void ProcessCollOp(CollOpInfo * _collop_) = 0;

Your simulator needs to implement this API function. This function processes a collective operation MPI event in trace file. This event object is a collective operation MPI event in the trace that is ready to be processed from the queues. This one object represents all threads involved in collective operation. The simulator does not get separate events for each thread involved in the collective operation.

virtual void ProcessP2P(P2PInfo * _message_) = 0;

Your simulator needs to implement this API function. This function processes a point-to-point MPI event in trace file. This event object is a P2P MPI event in the trace that is to be processed from the queues. This function is called for sendrecv, synchronous, or normal send to receive MPI operations.

virtual void ProcessNonBlockingSend(FuncEventInfo * _event_, P2PInfo * _message_) =0;

Your simulator needs to implement this API function. This function processes a non-blocking send MPI event in a trace file. This function gets an event object that represents the send operation and the corresponding P2P message that goes with that send. This function is only called for non-synchronous sends, or sends that do not rely on the receive thread. The start time of the receive thread may not be known.

virtual void ProcessP2PWaitAllSome(Vector<P2PInfo*, ObjectPointerAllocator<P2PInfo*> > * _messages_) = 0;

Your simulator needs to implement this API function. This function processes a `waitall` and `waitsome` MPI event in trace file. This function gets a vector of all P2P messages involved with the `waitall/waitsome`. There is no individual function events corresponding to the send or receive thread involved in this P2P operation.

9.2.3.4 Private Members

Members	Explanation
`Vector< char* > _commandLineParams;`	This vector represents the command-line parameters passed to the simulator by CPF. It is a vector of char pointers of parameters that do not match any keywords for CPF. These parameters may contain invalid values, and need to be checked by simulator.
`char * _stfPath;`	Pathway to the directory of the original STF file loaded by CPF at runtime.
`U8 _stfStart;`	Start time of original STF file.
`U8 _stfEnd;`	End time of original STF file

`U4 _numThreads;`	Number of threads in trace
`F8 _clockResolution;`	Resolution length of clock tick in seconds.
`Vector< U4 > _singleRankFuncs;`	Vector of single rank function id's. This vector is initialized in an `init()` call with the id values of every rank function in the trace file. Ranks that are not in the trace file are initialized with the value `-1`.
`Vector< U4 > _collOpFuncs;`	Vector of collective operation id's. This vector is initialized in an `init()` call with the id values of every collective function in the trace file. Given a function id number, the corresponding collective operation id is returned.
`Vector< U4 > _P2PFuncs;`	Vector of P2P identifiers. This vector is initialized in an `init()` call with the id values of every P2P function in the trace file. This vector returns the related the corresponding collective operation id for every function id number.
`HashMap< U4, const char* > _communicatorsMapped;`	`HashMap` of all communicator names. Lookup is by id of communicator. Data value is strings.

9.2.3.5 Private Functions

Functions	Explanation
`bool isSingleRankFunc( U4 _id_ )`	Checks to see whether the function id is a single rank ID. Returns `TRUE` if `_id_` is a single rank function; `FALSE` otherwise.
`bool isCollectiveOperation( U4 _id_ )`	Checks to see whether the function id matches a collective operation ID. Returns `TRUE` if `_id_` is a collective operation; `FALSE` otherwise.
`bool isP2P( U4 _id_ )`	This function checks to see whether the function id is a P2P ID. Returns `TRUE` if `_id_` is a P2P function; `FALSE` otherwise.
`bool isAllToAll( U4 _id_ )`	Returns `TRUE` if it is an ALL-TO-ALL collective operation id number; `FALSE` otherwise.
`bool isOneToAll( U4 _id_ )`	Returns `TRUE` if it is a ONE-TO-ALL collective operation id number; `FALSE` otherwise.
`bool isAllToOne( U4 _id_ )`	Returns `TRUE` if it is an ALL-TO-ONE collective operation id number; `FALSE` otherwise.
`bool isAll( U4 _id_ )`	Returns `TRUE` if it is an ALL collective operation id number; `FALSE` otherwise.
`bool isSendP2P( U4 _id_ )`	Returns `TRUE` when the FuncEventData is a P2P send operation; `FALSE` otherwise.
`bool isSyncSendP2P( U4 _id_ )`	Returns `TRUE` when the `FuncEventData` is a P2P sync send operation; `FALSE` otherwise.
`bool isSendRecv( U4 _id_ )`	Returns `TRUE` when the `FuncEventData` is a P2P send- receive operation; `FALSE` otherwise.
`bool isRecv( U4 _id_ )`	Returns `TRUE` when the `FuncEventData` is a P2P receive operation; `FALSE` otherwise.

9.2.3.6 Typedef Functions

These functions are essential to providing the plug-in module for CPF. They are already included in DevSim and if you modified DevSim, then you should have properly modified these functions to handle your simulator. The creat_t function creates an instance of your simulator class and passes a reference back to CPF. When CPF has completed processing a trace, it destroys the instance of the simulator through the destroy_t function.

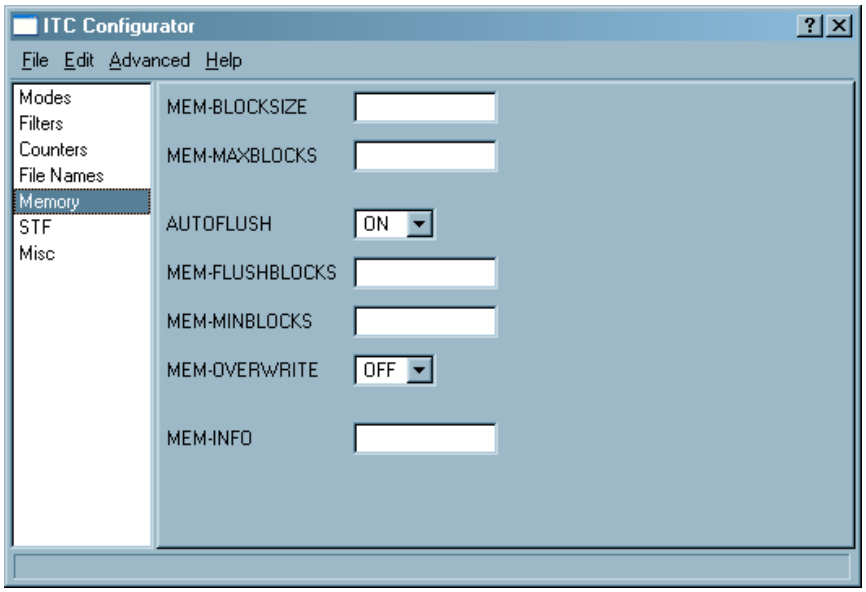
```
typedef CustomPluginFramework* create_t();
typedef void destroy_t(CustomPluginFramework*);
```

10 Intel® Trace Collector Configuration Assistant

10.1 General Description

The Intel® Trace Collector configuration assistant provides a simple GUI interface to view and edit Intel Trace Collector configuration files. It can also compute an average size of the trace file taking into account the modified parameters.

Figure 10.1 Intel® Trace Collector Configuration Assistant



As an example, to launch the Intel® Trace Collector configuration assistant for `trace.stf` you can type the following in the command line:

```
itcconfig trace.stf [configfile.conf]
```

The optional parameter `configfile.conf` is a valid Intel Trace Collector configuration file. If it is not presented in the command line then the default settings will be used.

The configuration file data created by the assistant can be imported into the Intel Trace Collector when running an application linked with it.

10.2 Detailed Description

The assistant GUI has three panels:

- The left one contains the variable groups
- The middle one contains the variables for each group
- The right one contains the documentation for each group of variables

The status bar has two sections:

- The left one represents the estimated trace file size for the currently set values of all the variables
- The right one shows the configuration file name currently open or default if the assistant is working with the default configuration

Here is the list of the supported Intel® Trace Collector variables:

```
ACTIVITY MEM-MINBLOCKS

AUTOFLUSH MEM-OVERWRITE

COUNTER OS-COUNTER-DELAY

CURRENT-DIR PCTRACE

DEMANGLE PCTRACE-CACHE

DETAILED-STATES PCTRACE-FAST

ENTER-USERCODE PROCESS

ENVIRONMENT PROGNAME

FLUSH-PID PROTOFILE-NAME

FLUSH-PREFIX STATE

INTERNAL-MPI STATISTICS

LOGFILE-FORMAT STF-CHUNKSIZE

LOGFILE-NAME STF-PROCS-PER-FILE

LOGFILE-PREFIX STF-USE-HW-STRUCTURE

LOGFILE-RANK SYMBOL

MEM-BLOCKSIZE SYNCED-CLUSTER

MEM-FLUSHBLOCKS SYNCED-HOST

MEM-INFO SYNC-MAX-DURATION

MEM-MAXBLOCKS VERBOSE
```

11 Concepts

This chapter contains explanations of some abstract concepts of Intel® Trace Analyzer.

11.1 Level of Detail

Tracing all available events over time can generate billions of events even for a moderate program runtime of a few minutes and a handful of CPUs. The sheer amount of data is a challenge for any analysis tool that has to cope with this data. This is even worse as in most cases the analysis tool cannot make use of the same system resources as the parallel computer on which the trace was generated.

An aspect of this problem arises when generating graphical diagrams of the event data. Obviously, it is next to impossible to graphically display all the data. Firstly, it would take ages to do that. Secondly, it would depend on round-off errors in the scaling and on the order of the data traversal which events would actually make it to the screen without being erased by others. So it is clear that only representatives of the actual events are shown.

A valid choice would be to paint only every 100th or 1000th event and to hope that the resulting diagram gives a valid impression of the data. But this approach has its problems, because the pattern selects the representatives can interfere with the patterns in the underlying data.

Intel® Trace Analyzer uses a Level of Detail concept to solve this problem. The Event Timeline Chart (as the other timelines) calculates a hint for the analysis that describes a time span that can reasonably be painted and selected with the mouse. This hint is called Resolution. The resolution requested by the timeline takes into account the currently available screen space and the length of the current time interval. Hence a higher screen resolution or a wider timeline results in more data being displayed for the same time interval.

Intel® Trace Analyzer then tries to find a near match for the requested resolution. The exact resolution depends on internals, which will not be discussed here.

Intel Trace Analyzer divides the requested time interval into slots of length resolution. After that, representatives for the function events, the messages and the collectives in these slots are chosen in a deterministic way. If a functions spans more than the given resolution it results in a larger slot.

The representatives for function events are chosen as follows: for each slot and each process (or thread group respectively) there is only a single function event representing the function where the thread or group spent most of its time.

The representatives for messages are chosen as follows: for each tuple (sender, receiver, sender slot, receiver slot) only one message is generated that carries averaged attributes. These attributes are averaged over all messages matching the tuple.

The representatives for collective operations are chosen as follows: for each tuple (communicator, first slot) one collective operation is generated. So it can happen that an operation of type `MPI_Gather` is merged with an operation of type `MPI_Bcast` resulting in a merged operation with no particular type at all (mixed).

To prevent misconceptions, emphasis is given to the fact that the merging of events only applies to the timelines and not to the profiles. The profiles always show sums, minima, maxima or averages over the

complete set of events. The calculation of these results does obviously not depend on the screen resolution.

11.2 Aggregation

Aggregation reduces the amount of data by aggregating events into thread groups and into function groups.

11.2.1 Thread Aggregation

A striking example for the benefit of thread groups is a parallel code that runs on a cluster of SMP systems. In fact this scenario was the inspiration to introduce this concept. To analyze the behavior of such an application, the data transfer rate is verified to check if the reached rate is plausible with respect to the data rates that are expected (maybe a fraction of the data rates advertised). Of course the effective and expected data transfer rates differ for messages that travel inside an SMP node (intra-node) and between two SMP nodes (inter-node).

In Intel® Trace Analyzer selecting Aggregation into the predefined process group **All_Nodes** is enough to make the distinction between intra-node and inter-node messages very easy: in the Message Profile the values for the intra-node messages appear on the diagonal of the matrix.

NOTE: Selecting a process group generally results in displaying the information for the group children (with the notable exception of the function profile). That is the reason why single, unthreaded processes or single threads cannot be selected for aggregation.

Until now, there was only a hierarchy with two levels, but more complicated hierarchies are useful too: threads living on the same core (due to Hyper Threading), threads living on different cores in the same CPU, threads living on the same FSB in different CPUs, threads living in the same SMP box on different FSBs, threads living in different boxes connected by a faster interconnect, threads living in SMP boxes connected by a not so fast interconnect and so on. These considerations suggest allowing for deeply nested thread groups.

If the thread group representing a single node is selected to concentrate on intra-node effects then the analysis might appear to be not faster but slower than using the thread group **All_Processes** alone. Why is that? The reason is twofold. The first is that Intel Trace Analyzer does not have to do any aggregation for the thread group **All_Processes** since it is flat (assuming no threads are used). The second is, despite the fact that only a single SMP node is chosen, all other threads go through the analysis and are thrown into the artificially created thread group **Other**. Click on
Views Menu > Advanced > Show Process Group 'Other' to make this group visible. To speed things up, choose a filter that only lets the threads of the selected SMP node pass.

NOTE: Filtering and Aggregation are orthogonal mechanisms in Intel Trace Analyzer.

11.2.2 Function Aggregation

Aggregation into function groups allows to decide on what level of detail to look at the threads or thread groups activity. In many cases it might be enough to see that a code spends some percent of its time in MPI without knowing in which particular function. (In some cases optimizing the serial parts of the program might seem more rewarding than optimizing the communication structure).

However, if the fraction spent in MPI exceeds the expectation, then it is interesting to know in which particular MPI call the time was spent. Function grouping allows exactly this shift in perspective by ungrouping the function group MPI.

While the argumentation given in Thread Aggregation for having nested thread groups may not be that compelling, the reason for having nested function groups comes quite clear as soon as there occur nested modules, classes and/or name spaces. This gets clearer if the binary instrumentation feature of Intel® Trace Collector is used as the result is thousands of functions instrumented.

Provided that there are adequate function groups, it is also much easier to categorize code by library or by author. In this way, it is possible to concentrate on precisely the code that is considered tunable while code that is controlled by third parties is aggregated into coarse categories.

NOTE: Selecting a function group generally results in displaying the information for the groups children. That is the reason why single functions cannot be selected for aggregation.

In the case of timelines, certain events may not be visible at all times. This does not necessarily mean that they are not there. The reason behind this is that the finer your grouping is, the less time is spent in each individual function/group. On aggregating over processes in a time interval where some processes are idle, nothing is displayed because of the idle state of the processes. Zooming in helps to see these events better. For a better overview, check one of the corresponding profiles. If the timeline and the profile seem to contradict, then the information from the profile is more precise.

11.3 Tagging and Filtering

Both concepts use the same filter grammar in their filter expressions. See Filtering Dialog and Tagging Dialog for usage hints.

Conceptually there is a different filter for each class of events: function events, messages and collective operations. During analysis the right filter expression is evaluated for each event and the event is tagged or passed on, if the expression is true while it is left untagged or suppressed, if the expression is false.

The behavior of a filter is determined at the time of its creation. A filter will continue filtering the same way until it is changed, even if the thread groups or function groups that it references are changed. Events will be treated as belonging to these groups based on the state of the groups at the time the filter was first created.

11.3.1 Tagging

If several events are merged as described in Level of Detail then the merged event is tagged if at least one of the singular events is tagged.

Therefore tagging in particular together with the Qualitative Timeline Chart (see Qualitative Timeline) is a very powerful tool to find events matching a certain criterion. This is because only if a single event out of billions matches the criteria of the tag filter, then it is guaranteed that there will be a highlighted peak in the Qualitative Timeline Chart that indicates where to zoom into the trace.

11.3.2 Filtering

If an event is suppressed by filtering then the effect is as if it were never written to the trace file. This is relatively easy to understand for messages and collective operations.

However, if a filter is designed that lets all functions except MPI pass, then there won't be holes in the Event Timeline, instead it will look as if MPI was not called. This means that it will look as if the thread was in the calling function instead of being in MPI. The same is true for the call tree and the call graph.

What happens if there are functions A, B and C in the trace where A calls B and B calls C and then suppress B by filtering it out? It would appear as if A had called C directly! This is quite different from

choosing a function aggregation that does not cover B, because that will show the function group other wherever B was shown before. Again: Filtering and Aggregation are orthogonal.