# Oracle® Solaris Studio 12.3: Performance Analyzer MPI Tutorial

Part No: E22002
January 2012

ORACLE

120118@25097

# Contents

# Preface

This document demonstrates Performance Analyzer features for programs that use MPI.

## Supported Platforms

This Oracle Solaris Studio release supports platforms that use the SPARC family of processor architectures running the Oracle Solaris operating system, as well as platforms that use the x86 family of processor architectures running Oracle Solaris or specific Linux systems.

This document uses the following terms to cite differences between x86 platforms:

- "x86" refers to the larger family of 64-bit and 32-bit x86 compatible products.
- "x64" points out specific 64-bit x86 compatible CPUs.
- "32-bit x86" points out specific 32-bit information about x86 based systems.

Information specific to Linux systems refers only to supported Linux x86 platforms, while information specific to Oracle Solaris systems refers only to supported Oracle Solaris platforms on SPARC and x86 systems.

For a complete list of supported hardware platforms and operating system releases, see the Oracle Solaris Studio 12.3 Release Notes.

## Oracle Solaris Studio Documentation

You can find complete documentation for Oracle Solaris Studio software as follows:

- Product documentation is located at the Oracle Solaris Studio documentation web site, including release notes, reference manuals, user guides, and tutorials.
- Online help for the Code Analyzer, the Performance Analyzer, the Thread Analyzer, dbxtool, DLight, and the IDE is available through the Help menu, as well as through the F1 key and Help buttons on many windows and dialog boxes, in these tools.
- Man pages for command-line tools describe a tool's command options.

# Resources for Developers

Visit the Oracle Technical Network web site to find these resources for developers using Oracle Solaris Studio:

- Articles on programming techniques and best practices
- Links to complete documentation for recent releases of the software
- Information on support levels
- User discussion forums.

# Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

# Typographic Conventions

The following table describes the typographic conventions that are used in this book.

**TABLE P–1** Typographic Conventions

| Typeface | Description | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories, and onscreen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`machine_name% you have mail.` |
| **`AaBbCc123`** | What you type, contrasted with onscreen computer output | `machine_name%` **`su`**<br>`Password:` |
| *aabbcc123* | Placeholder: replace with a real name or value | The command to remove a file is `rm` *filename*. |
| *AaBbCc123* | Book titles, new terms, and terms to be emphasized | Read Chapter 6 in the *User's Guide*.<br>A *cache* is a copy that is stored locally.<br>Do *not* save the file.<br>**Note:** Some emphasized items appear bold online. |

# Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

**TABLE P–2** Shell Prompts

| Shell | Prompt |
|---|---|
| Bash shell, Korn shell, and Bourne shell | `$` |
| Bash shell, Korn shell, and Bourne shell for superuser | `#` |
| C shell | `machine_name%` |
| C shell for superuser | `machine_name#` |

◆ ◆ ◆ CHAPTER 1

# Performance Analyzer MPI Tutorial

This tutorial is designed to be followed from beginning to end to show you how to use the Oracle Solaris Studio 12.3 Performance Analyzer's MPI features. The tutorial uses a sample program to demonstrate how to use the MPI Timeline and MPI Chart tabs.

## About MPI and Performance Analyzer

MPI is the Message Passing Interface, a standardized API used for parallel and distributed computing. This document assumes that you are experienced in developing applications that use MPI and run in a distributed system such as a cluster. The document does not explain how to set up a distributed computing environment or how to use MPI.

You can use the Performance Analyzer to examine MPI applications to answer the following questions:

- Would tuning the MPI code produce significant performance improvement?
- Is the MPI performance characterized by synchronization or data transfer?
- Does the program contain load imbalances?
- How long is one iteration of program execution?
- How long does it take for program performance to equilibrate?
- What are the message-passing patterns in program execution?
- Which are most important: long or short messages?
- Do processes that send messages synchronize with processes that receive messages?

While the preceding list is too broad to address in a single document, this tutorial guides you through some new features of the Performance Analyzer including the following:

- MPI Timeline. A graphical display of the MPI activity that occurred during an application's run.

- MPI Chart. A tool that generates scatter plots and histograms to visualize the performance data of MPI functions and MPI messages.
- MPI data-zooming and data-filtering. A set of controls that you can use to broaden or narrow your view of the data in the MPI Timeline and MPI Chart.

The MPI Timeline tab presents the data from a run of the test program as a timeline. Initially, your view of the timeline encompasses the run from beginning to end with all MPI functions and MPI messages represented graphically in a condensed form. You'll learn how to expand this presentation and move down from a complete view to a tightly focused view that can be as granular as a single function. The MPI Timeline tab offers many different ways to zoom, pan, and examine the data, together with MPI Chart tab. The MPI Chart tab enables you to plot statistical data about the functions and messages in graphical charts, to help you see what is happening in the run.

See the manual *Oracle Solaris Studio 12.3: Performance Analyzer* for detailed information about the Performance Analyzer.

# Setting Up for the Tutorial

The Performance Analyzer works with several implementations of the Message Passing Interface (MPI) standard, including the Oracle Message Passing Toolkit, a highly optimized implementation of Open MPI for Oracle Sun x86 and SPARC-based systems.

---

**Note –** The Oracle Message Passing Toolkit product was formerly named Sun HPC ClusterTools, and you might see both names in Oracle web pages and documentation. The last version of Sun HPC ClusterTools is version 8.2.1c. The next release of the product is Oracle Message Passing Toolkit 9, and is available for Oracle Solaris 11 in the software package repository.

---

You can see a list of MPI versions that work with the Performance Analyzer by running the command `collect` with no additional arguments.

This tutorial explains how to use the Performance Analyzer on a sample MPI application called `ring_c`.

You must already have a cluster configured and functioning for this tutorial.

## Obtaining MPI Software

Although this tutorial uses MPI software from Oracle, you can also use Open MPI, available at the Open MPI web site (`http://www.open-mpi.org`).

Information about the Oracle Message Passing Toolkit software is available at `http://www.oracle.com/us/products/tools/message-passing-toolkit-070499.html`. The site provides links for downloading the software, but please use the following detailed instructions instead.

## MPI Software for Oracle Solaris 10 and Linux

You can use Sun HPC ClusterTools 8 and its updates on Oracle Solaris 10 or Linux for this tutorial. See the instructions below for information about downloading Sun HPC ClusterTools 8.2.1c.

To download Sun HPC ClusterTools for Oracle Solaris 10 and Linux:

1. Log in to My Oracle Support (`http://support.oracle.com`). You must have a support contract for Oracle Solaris or Oracle Solaris Studio in order to download the software.
2. Type **`ClusterTools`** in the search box and click the search button.
3. Click Patches in the Refine Search area on the left side of the page.
4. Click `HPC-CT-8.2.1C-G-F - HPC ClusterTools 8.2.1c.`
5. Click the Download button, and follow instructions in the dialog box to download a compressed file containing the ClusterTools software.
6. After the file is downloaded, extract it and navigate to the appropriate platform directory for system. If necessary extract more files in this directory.
7. Install the software as described in the *Sun HPC ClusterTools 8.2.1c Software Installation Guide*, which is available as a PDF in the extracted directory. Note that the names of the top level directories described in the manual might not match the paths used in the extracted directories, but the subdirectories and program names are the same.
8. Add the directories */Studio-installation*`/bin` and */ClusterTools-installation*`/bin` directory to your path.

## MPI Software for Oracle Solaris 11

If you are running Oracle Solaris 11, the Oracle Message Passing Toolkit is made available as part of the Oracle Solaris 11 release under the package name `openmpi-15`. See the article *How to Compile and Run MPI Programs on Oracle Solaris 11* for information about installing and using the Oracle Message Passing Toolkit.

See the manual *Adding and Updating Oracle Solaris 11 Software Packages* in the Oracle Solaris 11 documentation library for general information about installing software in Oracle Solaris 11.

**Note –** The sample code used in this tutorial is not provided in the Oracle Solaris 11 package `openmpi-15`. If you install this version, you must obtain the sample code separately, as described in "Sample Code for Oracle Message Passing Toolkit in Oracle Solaris 11" on page 12. You can also download Open MPI to get the sample code.

# Prepare the Sample Source Code

See the section below that is appropriate for your MPI software for information about preparing the sample source code.

## Sample Code for ClusterTools and Open MPI

If you have either Sun HPC ClusterTools or Open MPI, you must make a writable copy of the `examples` directory in a location that is accessible from all the cluster nodes.

For example:

- If you have Sun HPC ClusterTools 8.2.1c, copy the directory `/opt/SUNWhpc/HPC8.2.1c/sun/examples` into a shared directory to which you have write access.
- If you have Open MPI 1.4.4, copy the directory `openmpi-1.4.4/examples` into a shared directory to which you have write access.

## Sample Code for Oracle Message Passing Toolkit in Oracle Solaris 11

The sample code is not included in the Oracle Solaris 11 `openmpi-15` package. The source code for `ring_c.c` and the `Makefile` for building the program is shown in Appendix B, "Sample Code for the Tutorial." You can copy the text of the files and create the `ring_c.c` and `Makefile` yourself.

To create the files:

1. Make a directory called `examples` in a location that is accessible from all the cluster nodes.
2. Using the mouse or keyboard, copy the source code for `ring_c.c` and the `Makefile` from Appendix B, "Sample Code for the Tutorial," and paste the text in a text editor.
3. Save the files in your `examples` directory.

# Compile and Run the Sample Program

To compile and run the sample program:

1. Change directory to your new `examples` directory.
2. Build the `ring_c` example.

   ```
   % make ring_c
   mpicc -g -o ring_c ring_c.c
   ```

   The program is compiled with the `-g` option, which allows the Performance Analyzer data collector to map MPI events to source code.
3. Run the `ring_c` example with `mpirun` to make sure it works correctly. The `ring_c` program simply passes a message from process to process in a ring, then terminates.

   This example shows how to run the program on a two-node cluster. The node names are specified in a host file, along with the number of slots that are to be used on each node. The tutorial uses 25 processes, and specifies one slot on each host. You should specify a number of processes and slots that is appropriate for your system. See the `mpirun`(1) man page for more information about specifying hosts and slots. You can also run this command on a standalone host that isn't part of a cluster, but the results might be less educational.

   The host file for this example is called `clusterhosts` and contains the following content:

   ```
   hostA slots=1
   hostB slots=1
   ```

   You must have permission to use a remote shell (`ssh` or `rsh`) to each host without logging into the hosts. By default, `mpirun` uses `ssh`.

   ```
   % mpirun -np 25 --hostfile clusterhosts ring_c
   Process 0 sending 10 to 1, tag 201 (25 processes in ring)
   Process 0 sent to 1
   Process 0 decremented value: 9
   Process 0 decremented value: 8
   Process 0 decremented value: 7

   Process 0 decremented value: 6
   Process 0 decremented value: 5
   Process 0 decremented value: 4
   Process 0 decremented value: 3
   Process 0 decremented value: 2
   Process 0 decremented value: 1

   Process 0 decremented value: 0
   Process 0 exiting
   Process 1 exiting
   Process 2 exiting
   .
   .
   .
   Process 24 exiting
   ```

   Run this command and if you get similar output, you are ready to collect data on an example application as shown in the next section.

If you have problems with mpirun using ssh, try using the option --mca plm_rsh_agent rsh with the mpirun command to connect using the rsh command:

```
% mpirun -np 25 --hostfile clusterhosts --mca plm_rsh_agent rsh -- ring_c
```

# Collecting Data on the ring_c Example

1. Change to the directory where your example binaries and source code are located.
2. Run the following command:

   ```
   % collect -M OMPT mpirun -np 25 --hostfile clusterhosts  -- ring_c
   ```

   The command might take a few moments to run and the output should be the same as the test run through the mpirun command.

   The -M OMPT option indicates the MPI version is the Oracle Message Passing Toolkit. See the collect(1) man page for more information about MPI versions supported.

   The -np 25 option specifies 25 processes on the cluster, and --hostfile clusterhosts indicates that the node names and the number of slots that are to be used on each node are specified in a file called clusterhosts.

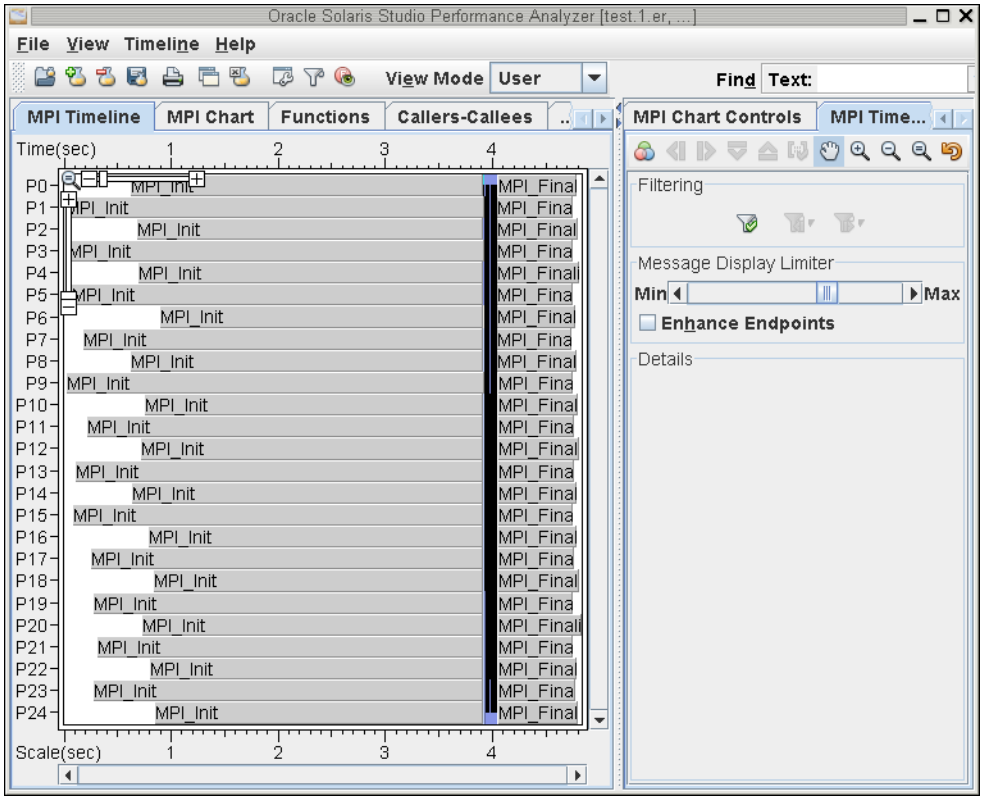
   This command specifies to use 25 processes on two hosts, and specifies one slot on each host. You should specify a number of processes and slots that is appropriate for your system.
3. List the contents of the newly created test.1.er directory and make sure the date on the files reflects the latest execution. This means you ran the command successfully and are ready to run the Performance Analyzer on ring_c. The integer in test.1.er increments for each collect command you run so the rest of this tutorial refers to this name generically as test.*.er.

# Opening the Experiment

1. Change to the directory that contains the ring_c.c source file, the ring_c executable, and the test.*.er directory.
2. Start the Performance Analyzer from the command line:

   ```
   %  analyzer
   ```

   The Performance Analyzer opens a file browser for you to find and open an experiment. If not, choose File > Open Experiment.
3. Find the test.*.er experiment that you just created and open it. The Performance Analyzer window should look similar to that below.



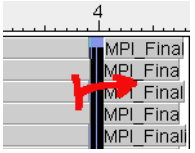

The experiment opens on the MPI Timeline tab. The MPI Chart tab is next to it. In the right panel you can see the MPI Chart Controls and MPI Timeline Controls tabs.

The MPI Timeline shows a view of the data over time as the program was run through the collector. The horizontal axis shows elapsed time. At the bottom, the horizontal axis shows *relative time* with the origin at the left edge of the display. At the top, the horizontal axis shows *absolute time* where the origin is the start of the data. The vertical axis shows MPI process rank. For each MPI process you can look horizontally to see what the process is doing as a function of elapsed time.

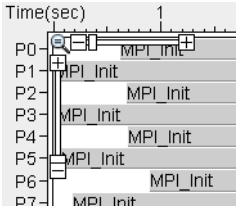
This initial view of the timeline answers the question: What is the time scale of program execution? The screen capture shows the time scale is approximately 5 seconds. However, the actual run time spans 3.90 to 4.05 seconds, the steady state of the application program. The `collect` tool uses `MPI_Init` and `MPI_Finalize` to set up and terminate data collection.

# Navigating the MPI Timeline

1. Click the MPI Timeline tab if it is not already selected.
2. Zoom in on the data by clicking and dragging from the left to the right on any process row as shown by the directional arrow in the graphic below. When you release the mouse button, the area inside the box automatically expands to reveal a zoomed in view.




An alternative to clicking and dragging is to use the zooming slider controls in the top left of the timeline.




Use the horizontal slider to change the time scale. You'll see progressively smaller chunks of time, while still showing all the processes, as you zoom.

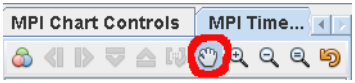
Use the vertical slider to zoom in on the MPI processes.

Click the Revert to Previous Zoom button in the MPI Timeline Controls tab shown below to go back to the previous level of zooming.


Click the Revert to Previous Zoom button a second time to return to the first zoom.

3. Pan across the data by sliding the scroll bars located at the bottom and the right of the timeline.

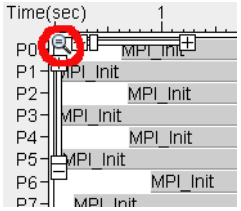
Alternatively you can toggle between a pointer that zooms and a pointer that pans by clicking the hand icon in the MPI Timeline Controls tab.



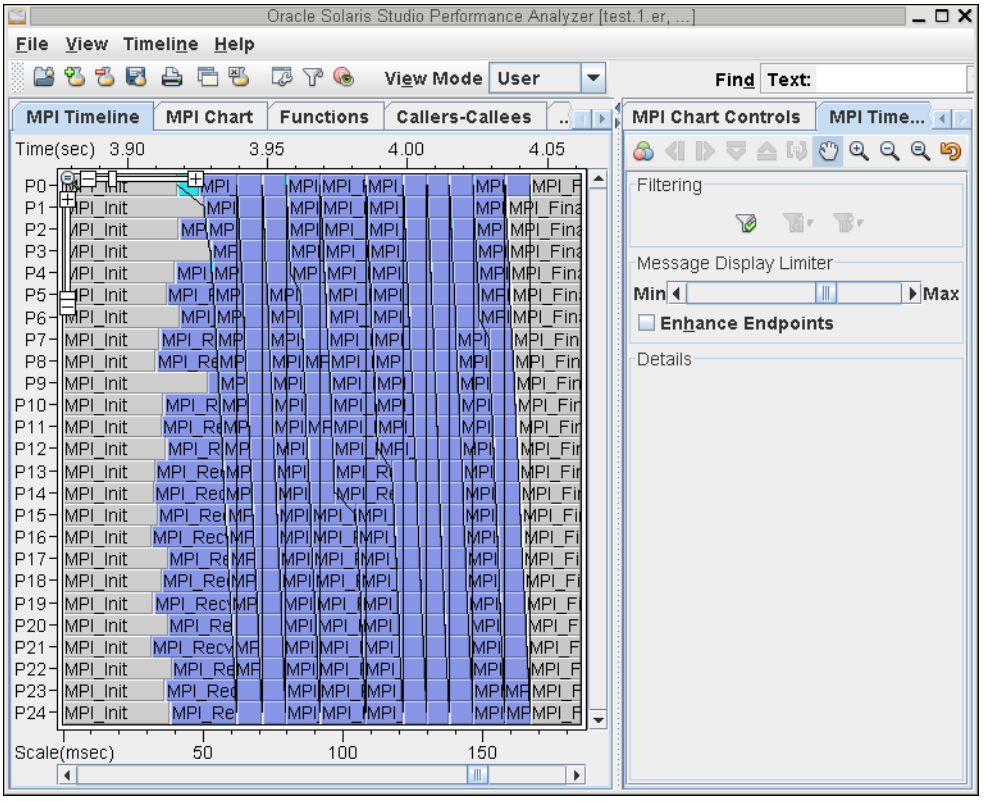

When the pointer is a hand, you can drag across the MPI Timeline to pan horizontally.

# Viewing Message Details

1. Reset the view to the original, maximum, zoomed-out setting by clicking the Zoom Reset button, which is located to the top left of the zoom sliders.




2. Zoom in on the activity area by dragging on the area horizontally with the mouse so it looks similar to what you see here.



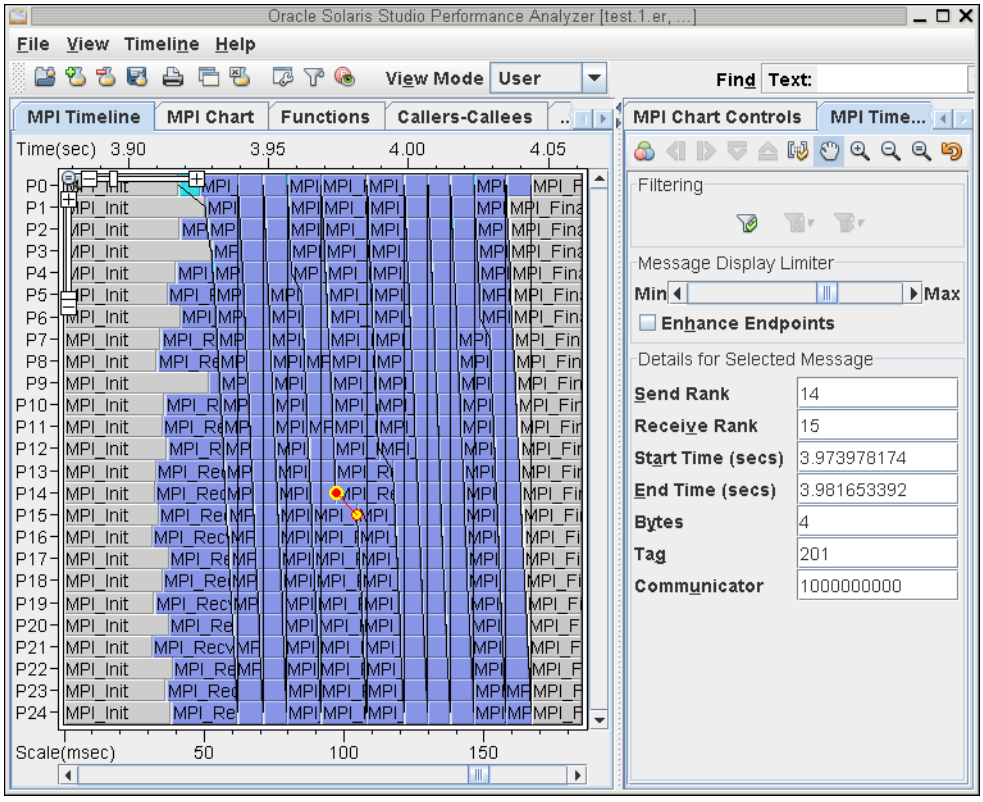

In the zoomed in timeline, now you can see that the steady state portion of the program execution appears to be from 3.93 seconds to 4.03 seconds.

You can also see that MPI functions are color coded. The black lines drawn between events represent point-to-point messages exchanged by the MPI processes.

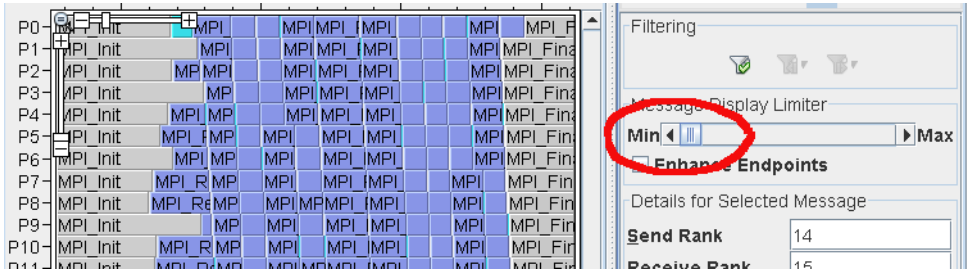
With this view of the timeline, you can answer the question: How long is one iteration before the pattern repeats? The answer is roughly 10 milliseconds. Look at the relative time scale at the bottom to see how often the loop seems to repeat.

3. Click one of the black message lines.

   The line turns red and details about the message are displayed in the right-hand panel MPI Timeline Controls tab.




4. In the MPI Timeline Controls tab, find the Message Display Limiter slider, then click and drag it to Min as shown.




The Message Display Limiter slider controls the number of message lines displayed on the screen. At Min, only functions are displayed in the MPI Timeline tab.

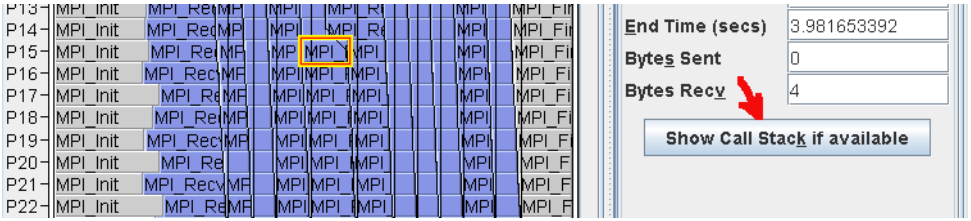
In this simple example, all of the messages can be displayed. However, displaying all messages in complex applications can overwhelm the tool and make the screen too cluttered to be usable. Select a lower limit to reduce the number of messages shown in the timeline. If fewer than 100% of the messages are shown, the messages used are those messages that are most costly in terms of the total time used in the message's send and receive functions.

5. Set the Message Display Limiter slider back to Max.

# Viewing Function Details and Application Source Code

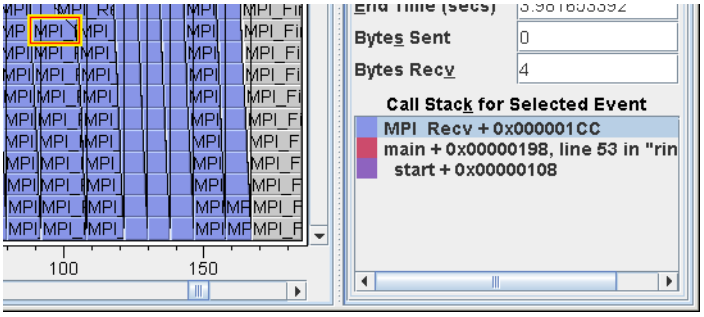
1. Click on one of the `MPI_Recv` function events in the MPI Timeline tab.

   The function is highlighted in yellow, and details about the function are displayed in the MPI Timeline Controls tab on the right.




2. In the MPI Timeline Controls tab, you should see the call stack. If you see a button labeled Show Call Stack if available, click the button.

   After a few moments, the call stack for the highlighted state should be shown in the MPI Timeline Controls tab:




3. When the Call Stack for Selected Event is displayed in the MPI Timeline Controls tab, click on `main + 0x00000198, line 53 in "ring_c.c`.
4. Click the Source tab in the main Performance Analyzer panel.

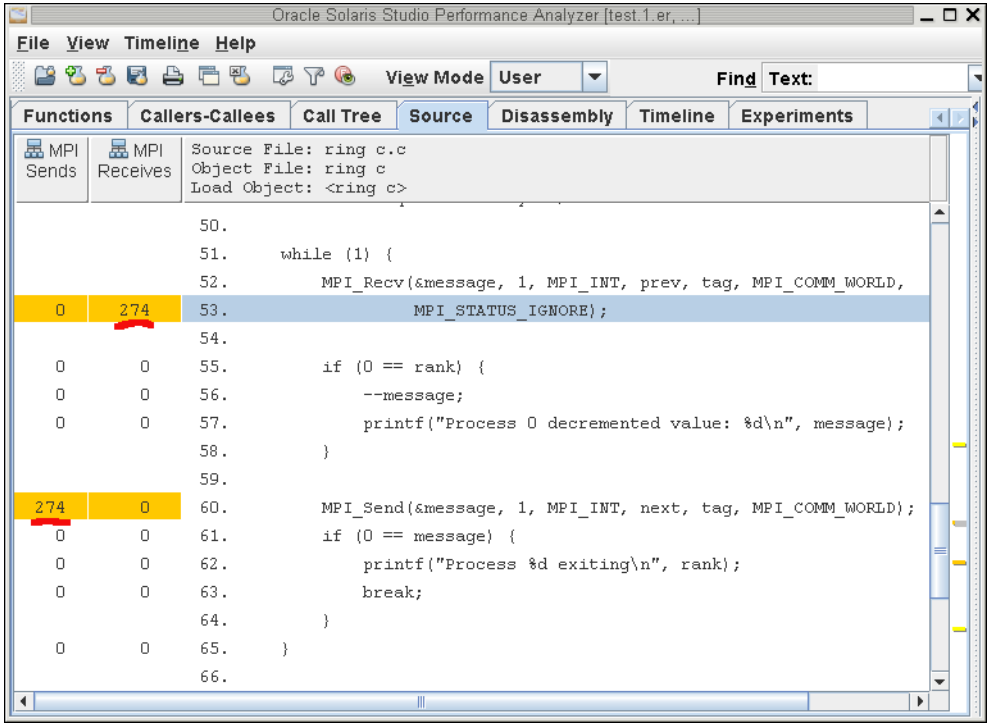
If you get a message such as `Object file (unknown) not readable`, make sure you selected the stack frame `main + 0x00000198, line 53 in "ring_c.c`.

---

**Note –** Source is only visible when the source is in the same location it was in when the program was run through the collector, or when it can be found in the `$expts` path as set in `.er.rc` or in View > Set Data Presentation > Search Path. Source also needs to be compiled with `-g`. If the source code is not visible, you may not have started the Analyzer from the directory containing the `ring_c` binary and source code. If this is the case, quit the Performance Analyzer and restart after you change to the directory containing `ring_c`.

---

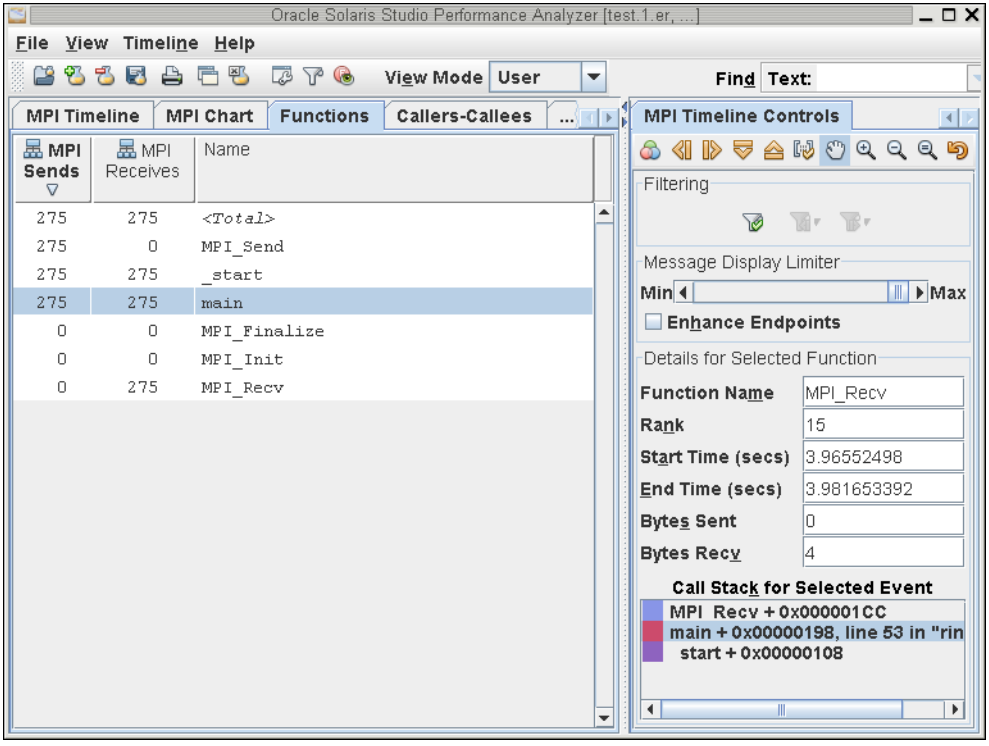
When the source is visible, it should show where `main()` calls the `MPI_Recv()` function. As shown below, `MPI_Recv()` is called from line 53 in the source. The metrics with high values are highlighted: 274 receives are associated with line 53, and 274 sends are associated with `MPI_Send()` on line 60.



**Tip –** The screen capture shows a reduced number of metrics. You can select more metrics to display by choosing View > Set Data Presentation > Metrics.

5. Click the Functions tab in the main Performance Analyzer panel.

   The Functions tab shows the same MPI Send and MPI Receive metrics in columns on the left side of a table. You can sort the table by clicking in the column headers.



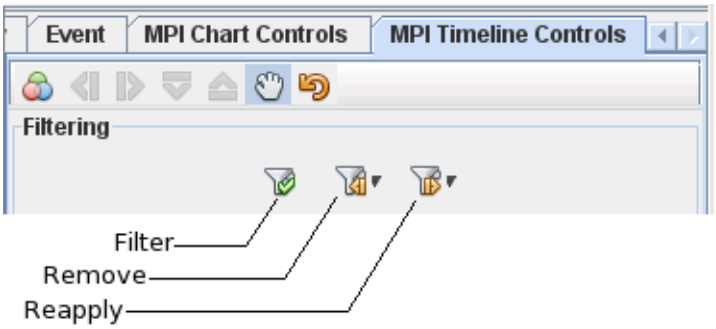

**Tip –** The screen capture shows a reduced number of metrics in the Functions tab. You can select metrics to display by choosing View > Set Data Presentation > Metrics.

6. Click the MPI Timeline tab to return to the MPI timeline.

   Do not click the regular Timeline tab because it does not apply to MPI programs.

# Filtering Data in the MPI Tabs

The filtering facility lets you select different views of the collected messaging data. You can remove and reapply the filters using the filtering controls in either the MPI Chart Controls tab or the MPI Timeline Controls tab.




The first control filters the data by removing everything that is not currently in view.

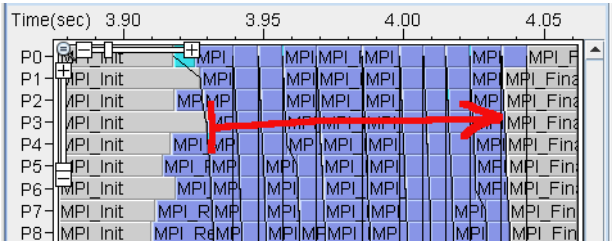
The second control is the Remove button which provides an associated drop down list for removing filters. Clicking this button removes the last filter applied. Clicking the down arrow presents a list of the filters applied, in the order they were applied, with the most recent at the top of the list. When you select a filter in this list, the selected filter and all filters above it on the list are removed.

The third control is the Reapply button, and it also has an associated drop down list for reapplying filters. Clicking the button reapplies the last filter that you removed. Clicking the down arrow opens a list of all the filters that have been removed, in the order in which they were removed. When you select a filter in this list, the selected filter and all filters above it on the list are reapplied.

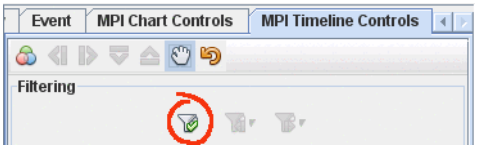
You can remove and reapply the filters by using the arrows, similar to going backward and forward in a web browser. You can even remove and reapply more than one filter in one click by using the down arrows next to the filter buttons.

The following steps explain how to use a filter to focus on the steady state portion of the program by filtering out the `MPI_Init` and `MPI_Finalize` functions.

1. On the timeline, drag the mouse horizontally to select a region such that `MPI_Init` and `MPI_Finalize` are no longer visible. In the example below, drag from t=3.93 to 4.03.



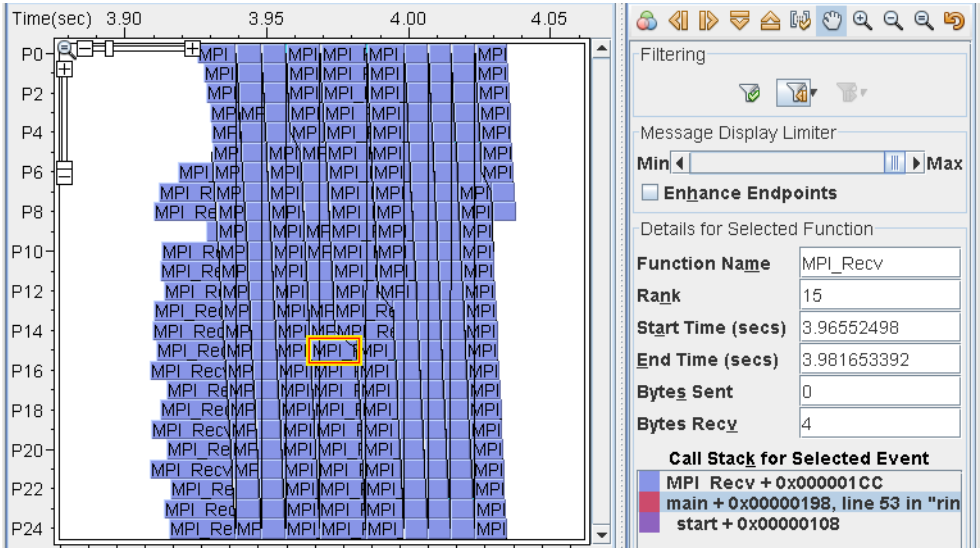

2. Click the Filter button in the MPI Timeline Controls tab.




It may look line nothing happened because the filtering is not evident until you change your view by zooming out or by looking at a chart.

3. Click the Revert to Previous Zoom button to go back out to the previous zoom.


The display now shows white areas in place of the `MPI_Init` and `MPI_Finalize` functions. White areas indicate that there is no MPI data collected for that area or the data has been filtered out.

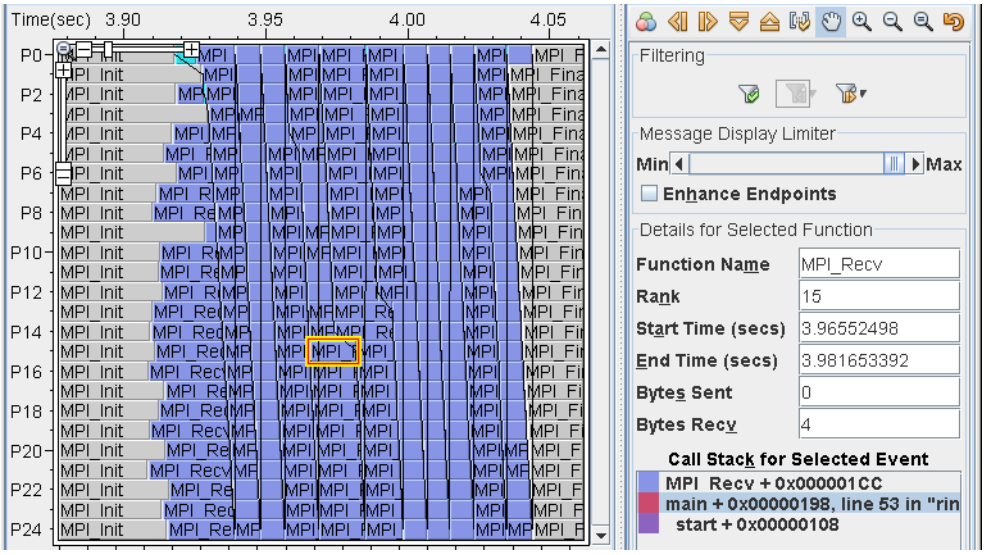



## Using the Filter Stack

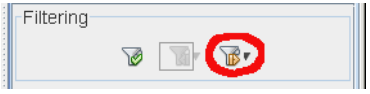
1. Click the Remove drop down arrow to reveal a list of applied filters.


This list lets you choose which filters to remove. It works like a stack: if you select `No filters applied`, everything on top of it will be taken off, which means there will be no filters applied.

2. Select `No filters applied` from the Remove list. The timeline should now look similar to the following.




3. Restore the previous filter using Reapply.

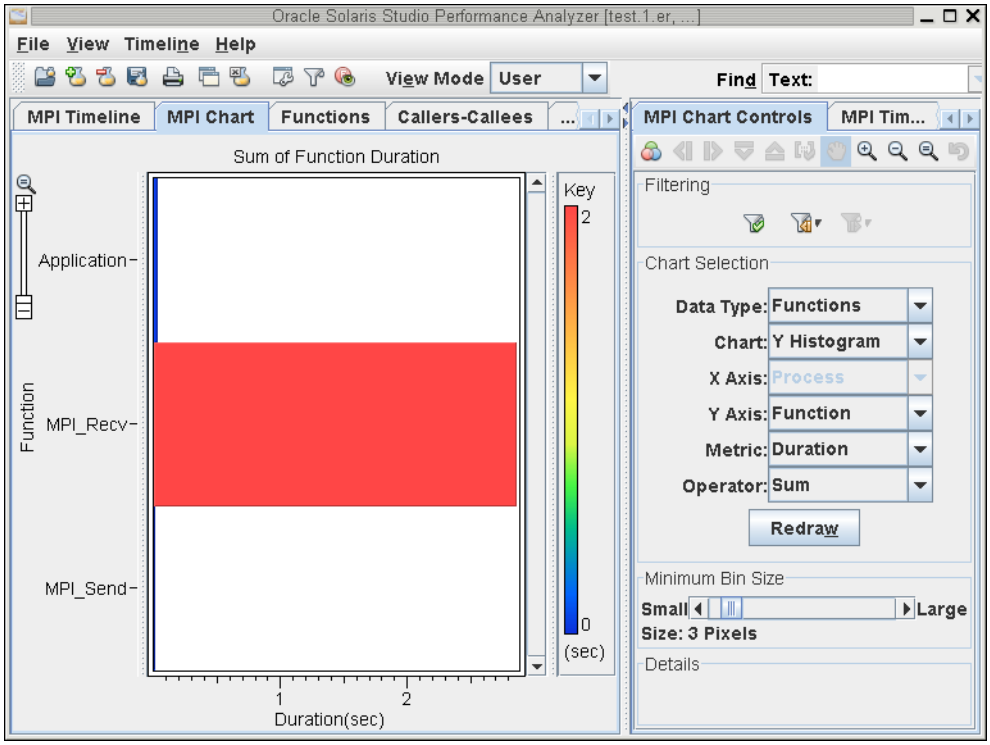



4. Before moving on to the next section, verify that `MPI_Init` and `MPI_Finalize` are filtered out and white areas are displayed as before.

# Using the MPI Chart Tab

With `MPI_Init` and `MPI_Finalize` already filtered out, explore the MPI Chart features. The initial chart shows which functions took the most time.

1. Click the MPI Chart tab to see a chart similar to the following.




The MPI Chart tab opens with a chart that shows the sum of the durations of the functions as they ran in all the processes. The vertical rainbow scale to the right of the chart shows the mapping between colors and values. The `MPI_Send` and `Application` functions take almost no time, whereas in the example, the cumulative time in `MPI_Recv` was nearly three seconds.

2. Click on the bar for the `MPI_Recv` function.

   The exact value of the bar is displayed in the MPI Chart Controls tab.

In this particular application, every process waits until the token has passed to every other process. As a result, significant time is spent in `MPI_Recv`, and little time is spent in `Application`, a state that represents time between MPI functions. As you will see later, a delay in message delivery to one rank prevents all other ranks from making progress.

## Using the MPI Chart Controls

This section shows how to use the MPI Chart Controls tab in different ways to visualize the data. Depending on the program under analysis, some chart options are more useful than others. See Appendix A, "MPI Chart Control Settings," for descriptions of all the MPI Chart controls.

## Make a Chart to Show Where Messages are Being Sent

1. Create a chart to look at messages by making the following selections in the MPI Chart Controls tab:

   Data Type: Messages

   Chart: 2–D Chart

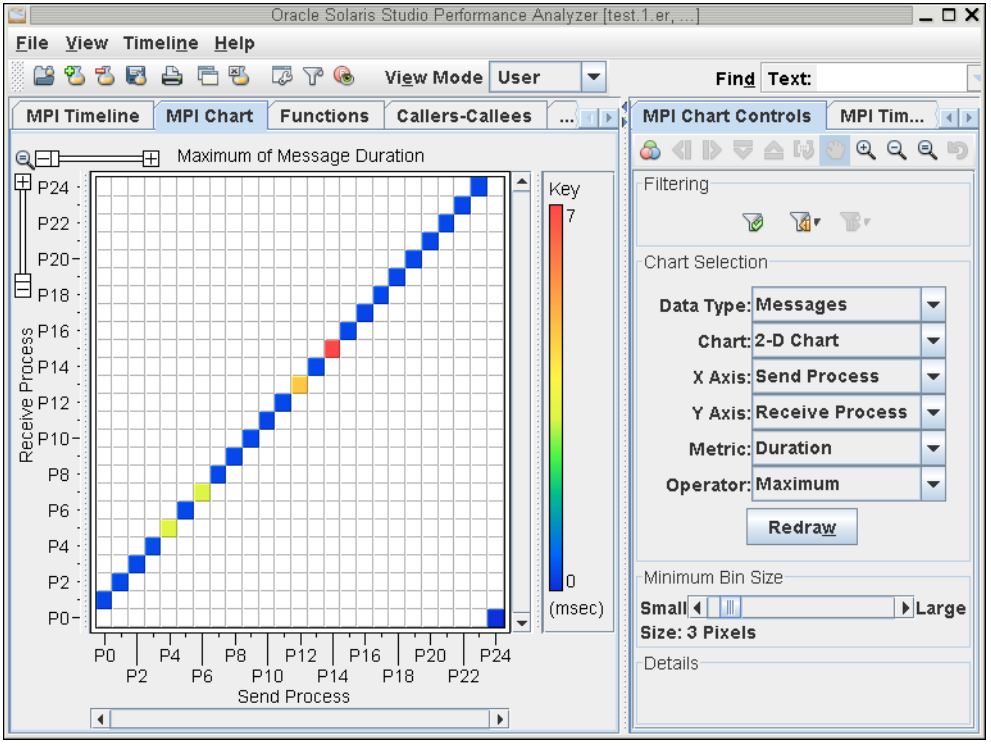
   X Axis: Send Process

   Y Axis: Receive Process

   Metric: Duration

   Operator: Maximum

2. Click Redraw, and you should see a chart similar to the following:




This chart shows that Process 0 sends only to Process 1. Process 1 only sends to Process 2, and so on. The color of each box is set by the metric selected (Duration) and the operator

(Maximum). Since this graph's Data Type is Messages, this will be the sum of duration of the messages, or the length of message lines in the time dimension.

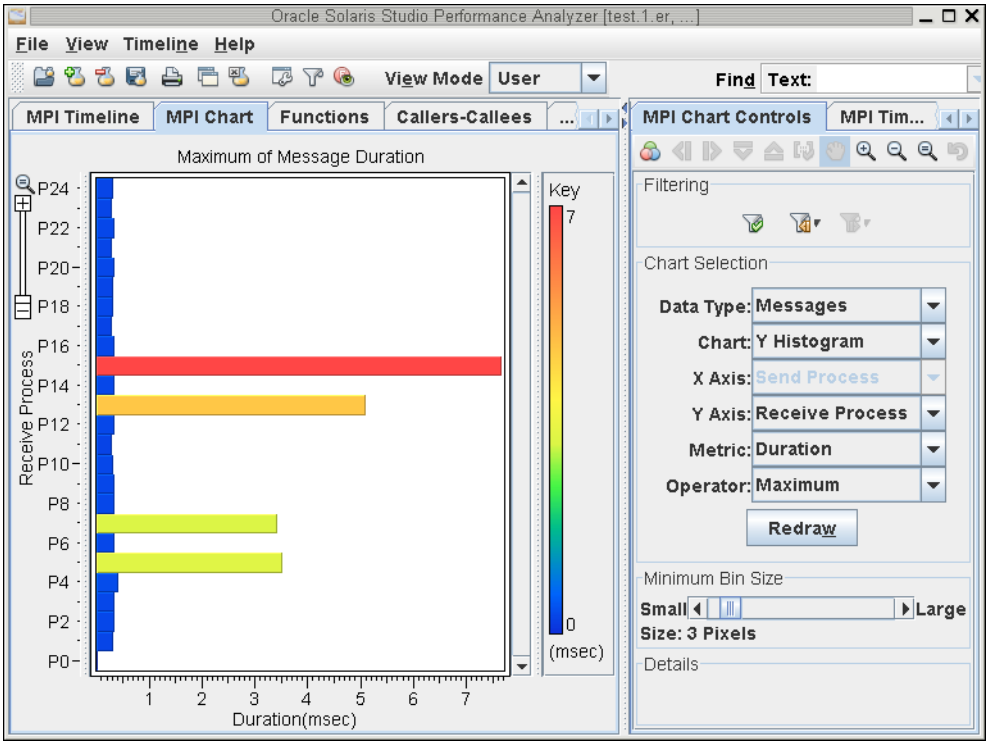
The chart colors indicate maximum message durations between ranks. In this example, the message that took the longest to arrive was sent from P14 to P15.

## Make a Chart to Show Which Ranks Waited Longest to Receive a Message

1. Make the following selections in the MPI Chart Controls tab:

   | | |
   |---|---|
   | Data Type: | Messages |
   | Chart: | Y Histogram |
   | X Axis: | *N/A* |
   | Y Axis: | Receive Process |
   | Metric: | Duration |
   | Operator: | Maximum |

2. Click Redraw to draw a new chart




In this example, the chart shows that the P15 rank waited the longest to receive a message.

3. To show a histogram for when these long duration messages occurred, make the following selections for the controls:

   Data Type: Messages

   Chart: X Histogram

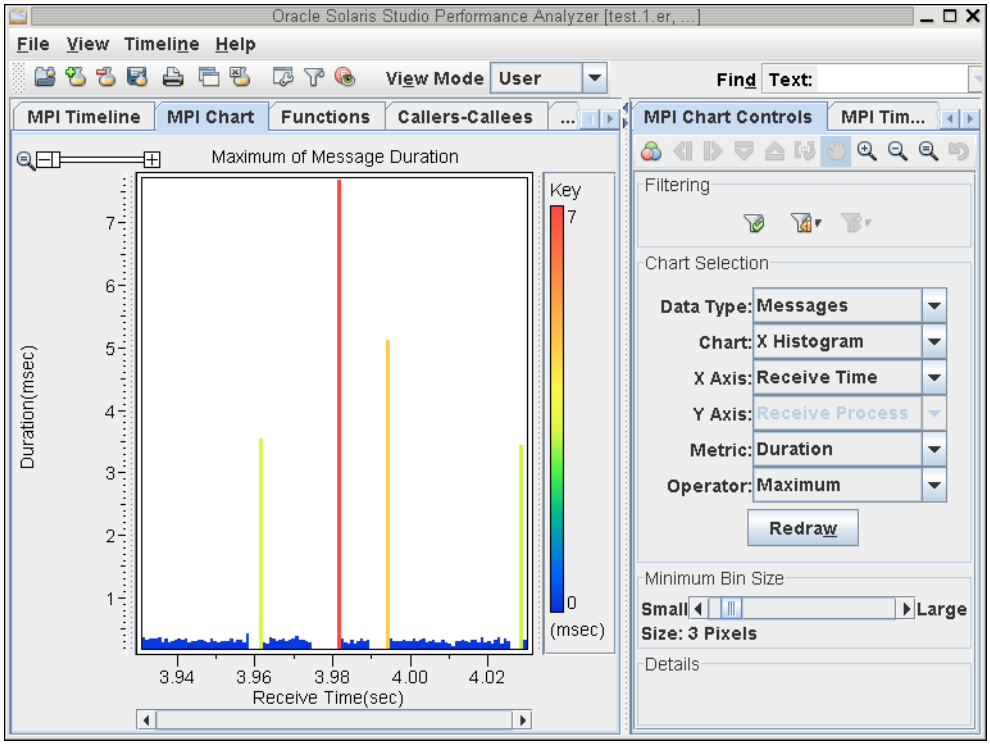
   X Axis: Receive Time

   Y Axis: *N/A*

   Metric: Duration

   Operator: Maximum

4. Click Redraw to see a chart similar to the following:




In this example, the slowest message was received at 3.981 seconds.

## Look for Slow Message Effects on Time Spent in MPI Functions

To see the effect of the long duration messages, create a graph that shows duration of functions versus time.

1. Create a chart to look at messages by making the following selections in the MPI Chart Controls tab:

   Data Type: Functions

   Chart: 2–D Chart

   X Axis: Exit Time

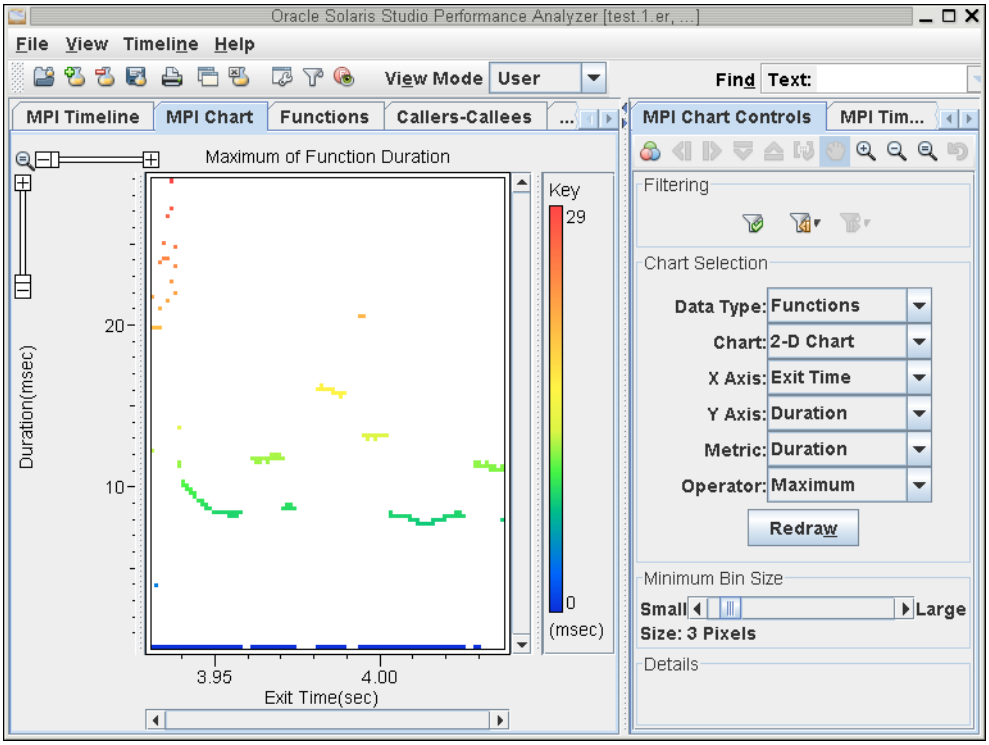
   Y Axis: Duration

   Metric: Duration

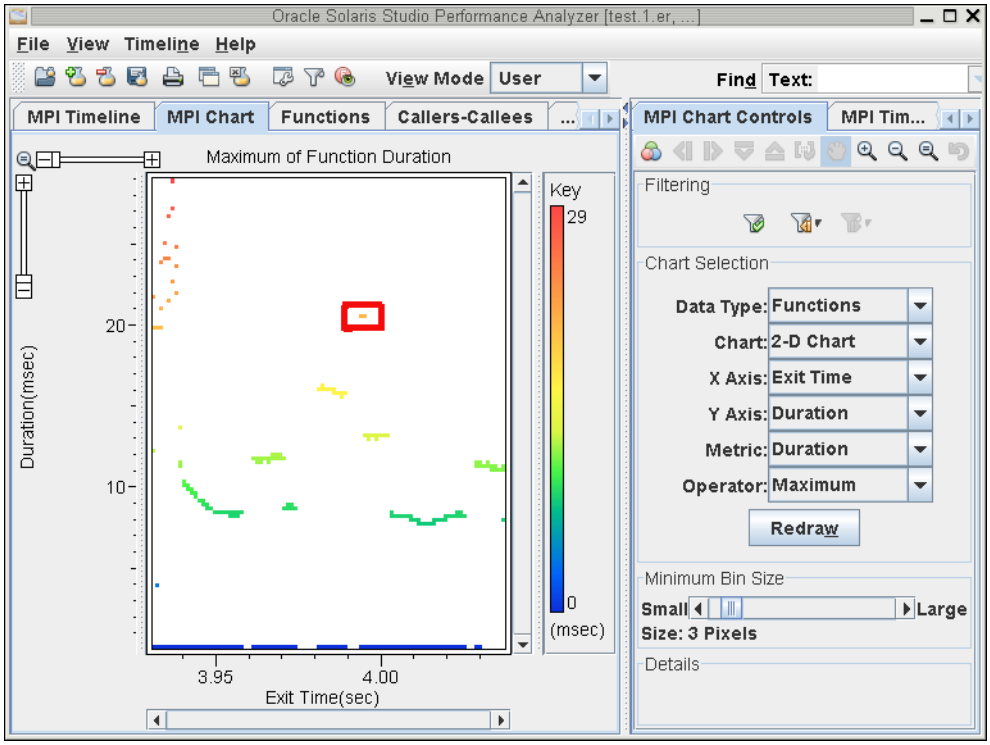
   Operator: Maximum

2. Click Redraw.

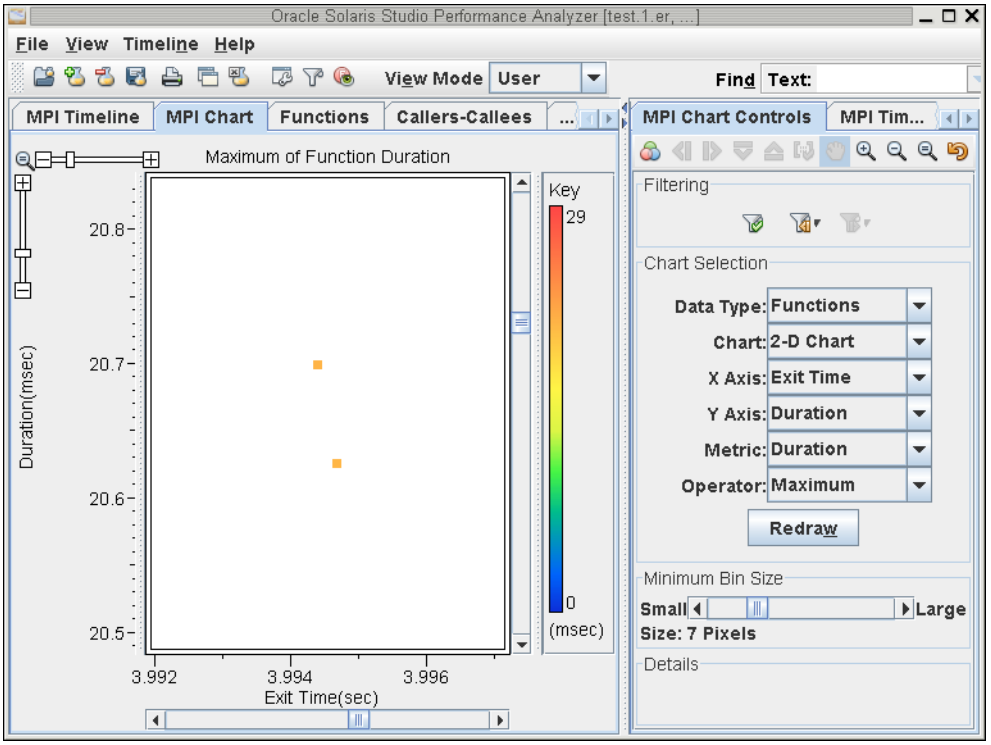
In this example, the graph shows several time regions that have long-duration functions. Note that at t=3.99, there are function durations longer than 20 seconds. This time corresponds to the long message flight-times viewed in the previous chart.




3. Isolate these long duration functions by dragging a box around them to zoom:

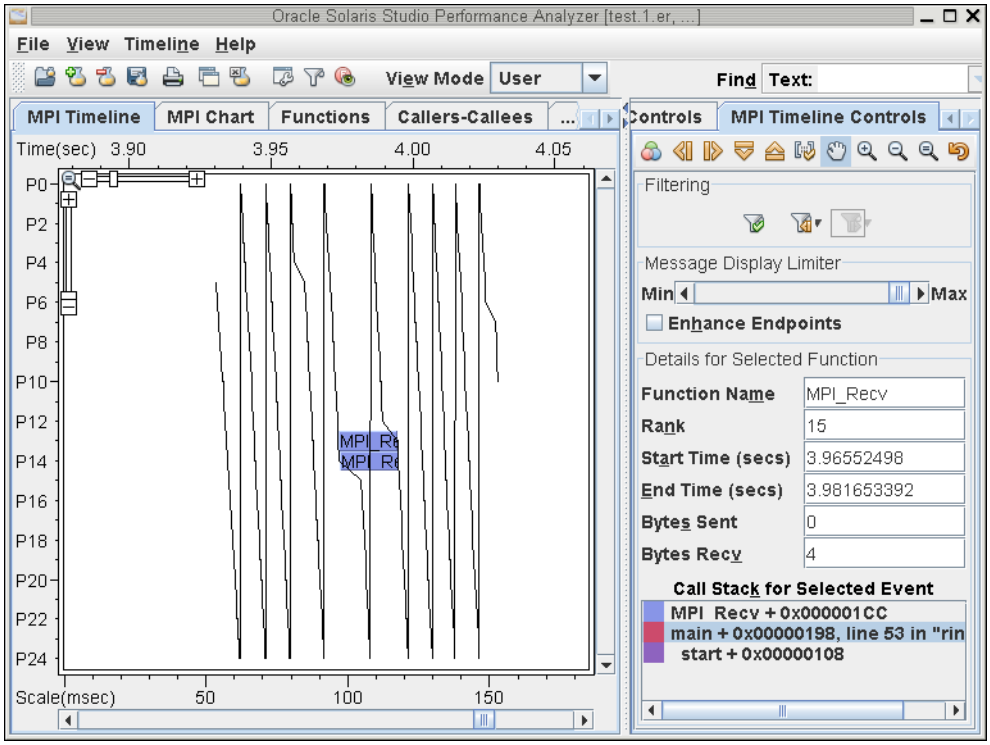



4. Click the Filter button to examine the only these function calls.




5. Click the MPI Timeline tab.

   You can now identify the high duration functions on the MPI Timeline. They are the result of messages with slow delivery times.




6. Click the Remove and Reapply buttons to toggle the context around the long duration functions.

For more information about the MPI Chart Controls, see Appendix A, "MPI Chart Control Settings"

# Conclusion

Oracle Solaris Studio Performance Analyzer enables you to observe performance and pinpoint problem areas in complex multithreaded applications. The simple example described in this tutorial illustrates the basics for examining relationships between MPI functions and messages. Using the MPI timeline, MPI charts, and zooming and filtering capabilities, you can gather and process performance data, view metrics at the program, function, source line, and instruction level, and identify potential performance problems before they become deployment issues.

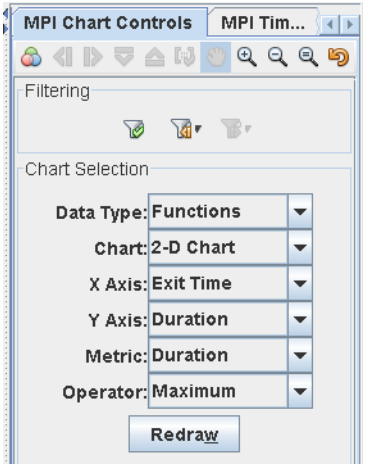
◆ ◆ ◆ APPENDIX A

# MPI Chart Control Settings

This appendix describes all the settings of the MPI Chart Controls tab in the Oracle Solaris Studio Performance Analyzer.

## Chart Attributes

This section describes the attributes you can set in the MPI Chart Controls tab to create charts of the MPI experiment data. The MPI Chart Controls tab is shown in the following screen capture.




- Data Type – Controls the type of data that is displayed in the chart, and can be set to one of the following values:

    - Functions - Plot data about the MPI functions used by the program. See "Functions Chart Attributes" on page 38 for attributes you can set for Functions charts.
    - Messages - Plot data about the MPI messages sent between process ranks. See "Messages Chart Attributes" on page 39 for attributes you can set for Messages charts.
- Chart – Controls the type of chart that is created, and can be set to one of the following values:
    - Y Histogram - One dimensional chart with the data plotted on the vertical axis as a function of another metric on the horizontal axis. You must select the type of data to plot on the Y axis and the metric for the X axis.
    - X Histogram - One dimensional chart with the data plotted on the horizontal axis as a function of time on the vertical axis. You must select the type of data to plot on the X axis, and the metric for the Y axis.
    - 2-D chart - Two dimensional chart with data plotted on both X and Y axis, making a 2-D matrix or scatter plot. You must specify what to plot on the X and Y axis, and the metric.
- X Axis - Select the type of data to plot on the horizontal axis, for X Histogram or 2-D Chart. The types available depend on whether you have selected Functions or Messages from the Data Type list. The possible values are described in "Functions Chart Attributes" on page 38 and "Messages Chart Attributes" on page 39.
- Y Axis - Select the type of data to plot on the vertical axis, for Y Histogram or 2-D Chart. The types available depend on whether you have selected Functions or Messages from the Data Type list. The possible values are described in "Functions Chart Attributes" on page 38 and "Messages Chart Attributes" on page 39
- Metric - Select what data is shown as a function of X or Y. The metric value is indicated through color in the charts. The types available depend on whether you have selected Functions or Messages from the Data Type list. The possible values are described in "Functions Chart Attributes" on page 38 and "Messages Chart Attributes" on page 39.
- Operator – Select the method used to combine metrics in the chart. The possible values are described in "Operator Settings" on page 40.

# Functions Chart Attributes

The following are the chart attributes you can set when plotting data for Functions. These attributes can be selected for the X Axis, Y Axis, and Metric.

- Time (range) - The range of times from entry to exit of a function
- Entry Time - The time when a function is called
- Exit Time - The time when a function returns to the caller
- Duration - The time difference between function entry and function exit
- Process - The MPI global ranks in numerical order. Each function call has a unique process rank associated with it.

- Function - The MPI function called.
- Send Bytes - Number of bytes sent in an MPI function call
- Receive Bytes - Number of bytes received in an MPI function call
- 1 (only for Metric) - Specifying 1 as the metric simply specifies an attribute whose value is always 1. This can be used to count data records or signal the presence or absence of data. For example, to count the number of function calls for each function, set Y Axis: Function, Metric: 1, Operation: Sum. To detect whether any function calls were made, set Operation: Maximum.

# Messages Chart Attributes

The following are the chart attributes you can set when plotting data for Messages. These attributes can be selected for the X Axis, Y Axis, and Metric.

- Time (range) - The range of time from send to receive of a message
- Send Time - The time that a message was sent
- Receive Time - The time that a message was received
- Duration - The time difference between send and receive of a message
- Send Process - The process that sent a message
- Receive Process - The process that received a message
- Communicator - An arbitrarily defined ID that uniquely labels the communicator (set of processes) used to send and receive the message
- Tag - The MPI tag used to identify the message
- Send Function - The function that sent the message
- Receive Function - The function that received the message
- Bytes - Number of bytes in the message
- 1 (only for Metric) - Specifying 1 as the Metric simply specifies an attribute whose value is always 1. This can be used to count data records or signal the presence or absence of data. For example, to count the number of function calls for functions that send a message, set Y Axis to Send Function, set Metric to 1, and set Operator to Sum. To detect whether any function calls were made for each function that sends a message, set Operator to Maximum.

# Operator Settings

The Operator control determines how multiple metric values are combined in the chart, and can be set to one of the following values:

- Sum - calculates the sum of the selected Metric, which must be one of: Time, Duration, Send/Receive Bytes, or "1"
- Maximum - calculates the maximum value of the selected Metric, which must be one of: Time, Duration, Send/Receive Bytes, or "1"
- Minimum - calculates the minimum value of the selected Metric, which must be one of: Time, Duration, Send/Receive Bytes, or "1"
- Average - calculates the average value of the selected Metric, which must be one of: Time, Duration, Send/Receive Bytes, or “1”
- Fair - The Fair operator operates on any type of metric. When many metric values are all assigned to the same chart bin, the Fair operator picks a single one of those values "fairly". For example, suppose that 90% of the MPI messages use the communicator `MPI_COMM_WORLD`, but 10% of them use a user-defined communicator `mycomm`. If you create a message chart using "Communicator" as the metric and "Fair" as the operator, the chart would report `MPI_COMM_WORLD` 90% of the time but `mycomm` 10% of the time.

◆ ◆ ◆ APPENDIX B

# Sample Code for the Tutorial

This appendix shows the sample code and Makefile used in the tutorial.

**EXAMPLE B–1** Sample Code for `ring_c.c`

```
/*
 * Copyright (c) 2004-2006 The Trustees of Indiana University and Indiana
 *                         University Research and Technology
 *                         Corporation.  All rights reserved.
 * Copyright (c) 2006      Cisco Systems, Inc.  All rights reserved.
 *
 * Simple ring test program
 */

#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int rank, size, next, prev, message, tag = 201;

    /* Start up MPI */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Calculate the rank of the next process in the ring.  Use the
       modulus operator so that the last process "wraps around" to
       rank zero. */

    next = (rank + 1) % size;
    prev = (rank + size - 1) % size;

    /* If we are the "master" process (i.e., MPI_COMM_WORLD rank 0),
       put the number of times to go around the ring in the
       message. */

    if (0 == rank) {
        message = 10;
```

**EXAMPLE B–1** Sample Code for `ring_c.c` *(Continued)*

```
        printf("Process 0 sending %d to %d, tag %d (%d processes in ring)\n",
               message, next, tag, size);
        MPI_Send(&message, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
        printf("Process 0 sent to %d\n", next);
    }

    /* Pass the message around the ring.  The exit mechanism works as
       follows: the message (a positive integer) is passed around the
       ring.  Each time it passes rank 0, it is decremented.  When
       each processes receives a message containing a 0 value, it
       passes the message on to the next process and then quits.  By
       passing the 0 message first, every process gets the 0 message
       and can quit normally. */

    while (1) {
        MPI_Recv(&message, 1, MPI_INT, prev, tag, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);

        if (0 == rank) {
            --message;
            printf("Process 0 decremented value: %d\n", message);
        }

        MPI_Send(&message, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
        if (0 == message) {
            printf("Process %d exiting\n", rank);
            break;
        }
    }

    /* The last process does one extra send to process 0, which needs
       to be received before the program can exit */

    if (0 == rank) {
        MPI_Recv(&message, 1, MPI_INT, prev, tag, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
    }

    /* All done */

    MPI_Finalize();
    return 0;
}
```

The following code shows the contents of the `Makefile` that is included with the sample code and can be used to build `ring_c.c`.

**EXAMPLE B–2** Makefile for building the ring program

```
#
# Copyright (c) 2004-2005 The Trustees of Indiana University and Indiana
#                         University Research and Technology
#                         Corporation.  All rights reserved.
# Copyright (c) 2004-2005 The University of Tennessee and The University
```

**EXAMPLE B–2** Makefile for building the ring program *(Continued)*

```
#                         of Tennessee Research Foundation.  All rights
#                         reserved.
# Copyright (c) 2004-2005 High Performance Computing Center Stuttgart,
#                         University of Stuttgart.  All rights reserved.
# Copyright (c) 2004-2005 The Regents of the University of California.
#                         All rights reserved.
# Copyright (c) 2006-2007 Sun Microsystems, Inc.  All rights reserved.
# $COPYRIGHT$
#
# Additional copyrights may follow
#
# $HEADER$
#

# Use the Open MPI-provided wrapper compilers.  Note that gmake
# requires the CXX macro, while other versions of make (such as Sun's
# make) require the CCC macro.

CC = mpicc
CXX = mpic++
CCC = mpic++
F77 = mpif77
FC = mpif90

# Using -g is not necessary, but it is helpful for example programs,
# especially if users want to examine them with debuggers.  Note that
# gmake requires the CXXFLAGS macro, while other versions of make
# (such as Sun's make) require the CCFLAGS macro.

CFLAGS = -g
CXXFLAGS = -g
CCFLAGS = -g
F77FLAGS = -g
FCFLAGS = -g

# Example programs to build

EXAMPLES = hello_c hello_cxx hello_f77 hello_f90 \
           ring_c ring_cxx ring_f77 ring_f90 connectivity_c

# Default target.  Always build the C example.  Only build the others
# if Open MPI was build with the relevant language bindings.

all: hello_c ring_c connectivity_c
    @ if test "`ompi_info --parsable | grep bindings:cxx:yes`" != ""; then \
        $(MAKE) hello_cxx ring_cxx; \
    fi
    @ if test "`ompi_info --parsable | grep bindings:f77:yes`" != ""; then \
        $(MAKE) hello_f77 ring_f77; \
    fi
    @ if test "`ompi_info --parsable | grep bindings:f90:yes`" != ""; then \
        $(MAKE) hello_f90 ring_f90; \
    fi


# The usual "clean" target
```

**EXAMPLE B–2** Makefile for building the ring program *(Continued)*

```
clean:
    rm -f $(EXAMPLES) *~ *.o

# Don't rely on default rules for the fortran examples

hello_f77: hello_f77.f
    $(F77) $(F77FLAGS) $^ -o $@
ring_f77: ring_f77.f
    $(F77) $(F77FLAGS) $^ -o $@

hello_f90: hello_f90.f90
    $(FC) $(FCFLAGS) $^ -o $@
ring_f90: ring_f90.f90
    $(FC) $(FCFLAGS) $^ -o $@
```