

# Tutorial: Analyzing MPI Applications

**Intel® Trace Analyzer and Collector**

**Intel® VTune™ Amplifier XE**

---

# Contents

---

<b>Legal Information</b> .....	<b>3</b>
<b>1. Overview</b> .....	<b>4</b>
1.1. Prerequisites.....	5
1.1.1. Required Software .....	5
1.1.2. Setting Up the Environment Variables .....	5
1.1.3. Creating Trace Files .....	5
1.2. Starting Intel® Trace Analyzer .....	6
1.3. Starting Intel® VTune™ Amplifier XE.....	7
<b>2. Analyzing an MPI Application</b> .....	<b>9</b>
2.1. Optimizing MPI Communications.....	9
2.1.1. Prepare for Analysis .....	10
2.1.2. Ungroup MPI Functions.....	11
2.1.3. Detect Serialization in Function Profile and Message Profile .....	12
2.1.4. Compare Original Trace File With Idealized Trace File.....	13
2.1.5. Remove Serialization .....	15
2.1.6. Compare Two Trace Files .....	16
2.1.7. Analyze Optimized Communications .....	17
2.2. Improving Intra-process Performance.....	18
2.2.1. Run Basic Hotspots Analysis .....	18
2.2.2. Interpret Results .....	19
<b>3. Summary</b> .....	<b>22</b>
<b>4. Key Terms</b> .....	<b>23</b>

# Legal Information

---

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel, the Intel logo, and VTune are trademarks of Intel Corporation in the U.S. and/or other countries.

\* Other names and brands may be claimed as the property of others.

Copyright © 2015, Intel Corporation. All rights reserved.

Intel® Trace Analyzer ships libraries licensed under the GNU Lesser Public License (LGPL) or Runtime General Public License. Their source code can be downloaded from <ftp://ftp.i kn.intel.com/pub/opensource>.

# 1. Overview

---



Intel® Trace Analyzer and Collector enables you to understand MPI application behavior and quickly find bottlenecks to achieve high performance for parallel cluster applications. Use the Intel Trace Analyzer and Collector to evaluate profiling statistics and load balancing, identify communication hotspots, and increase application efficiency.



Intel® VTune™ Amplifier XE enables you to find serial and parallel code bottlenecks and speed execution. Use this tool to analyze the algorithm choices, understand where and how your application can benefit from available hardware resources, and identify code sections that may cause unnecessary power consumption.

Both products are installed as part of [Intel® Parallel Studio XE Cluster Edition](#).

To improve performance of some complex applications, it is necessary to analyze their cross-process behavior as well as their single process performance. Intel Trace Analyzer and Collector enables you to analyze communications between processes, while Intel VTune Amplifier helps you find single process performance issues.

<b>About This Tutorial</b>	<p>This tutorial demonstrates a workflow applied to a sample program. The source code is available at <code>&lt;install-dir&gt;/examples/poisson</code>, where <code>&lt;install-dir&gt;</code> is the Intel Trace Analyzer and Collector installation directory.</p> <p>You can ultimately apply the same workflow to your own applications:</p> <ul style="list-style-type: none"><li>• Find communication imbalance issues in your application using the Intel® Trace Analyzer charts</li><li>• Find hotspots on the intra-process level of your application using Intel® VTune™ Amplifier XE</li><li>• Review the application</li></ul>
<b>Estimated Duration</b>	20-25 minutes
<b>Learning Objectives</b>	<p>After you complete this tutorial, you should be able to:</p> <ul style="list-style-type: none"><li>• Conduct a complex application analysis using Intel Trace Analyzer and Collector and Intel VTune Amplifier XE</li><li>• Improve overall performance</li></ul>
<b>More Resources</b>	<p>Learn more about the Intel Trace Analyzer and Collector and Intel VTune Amplifier in the guides and tutorials:</p> <ul style="list-style-type: none"><li>• <a href="#">Intel® Trace Analyzer User and Reference Guide</a></li><li>• <a href="#">Intel® Trace Collector User and Reference Guide</a></li><li>• <i>Finding Hotspots</i> tutorial at the</li><li>• <i>Intel® Trace Analyzer User and Reference Guide</i></li><li>• <i>Intel® Trace Collector User and Reference Guide</i></li><li>• <i>Intel® VTune™ Amplifier XE User's Guide</i></li><li>• <i>Tutorial: Finding Hotspots</i></li></ul> <p>The guides and tutorials are available at:</p> <ul style="list-style-type: none"><li>• <a href="#">Intel® Trace Analyzer and Collector Product Page</a></li></ul>

	<ul style="list-style-type: none"> <li>• <a href="#">Intel® VTune™ Amplifier XE Product Page</a></li> <li>• <a href="#">Intel® Software Documentation Library</a></li> </ul>
<b>Submit Feedback</b>	You can submit your feedback on the documentation at <a href="http://www.intel.com/software/products/softwaredocs_feedback/">http://www.intel.com/software/products/softwaredocs_feedback/</a> .

## 1.1. Prerequisites

This section describes the steps you need to do before you start using the Intel® Trace Analyzer and Collector and Intel® VTune™ Amplifier XE.

### 1.1.1. Required Software

To perform all the steps described in this tutorial, you will need the following software installed on your system:

- Intel® compilers
- Intel® MPI Library
- Intel® Trace Analyzer and Collector
- Intel® VTune™ Amplifier XE

All of these products are installed as part of [Intel® Parallel Studio XE Cluster Edition](#).

### 1.1.2. Setting Up the Environment Variables

#### Linux\* OS:

Set the required environment variables by sourcing the `psxevars.c[sh]` script available at `<install-dir>/parallel_studio_xe_<version>.x.xxx/bin`, where `<install-dir>` is the Intel® Parallel Studio XE Cluster Edition installation directory. For example:

```
$ source psxevars.sh
```

#### Windows\* OS:

Open the command prompt from **Start > Intel Parallel Studio XE version > Compiler and Performance Libraries > Intel 64 Visual Studio version environment**. This will set all required environment variables and you will be ready to trace your applications.

### 1.1.3. Creating Trace Files

To trace the `poisson` application, go to `<install-dir>/examples/poisson` and copy its contents into your working directory, then trace the application:

#### Linux\* OS:

1. Compile the application running the `make` command. Adjust the `Makefile`, if necessary.
2. Run the application with the `-trace` option of `mpirun`:

```
$ mpirun -n 4 -trace ./poisson
```

#### Windows\* OS:

1. Compile all components of the `poisson` application from the compiler command prompt with the `-trace` option. Use the `-zi` option to compile the application in debug mode. The basic command line for Fortran programs is:

```
> mpiifort -trace -zi myApp.f90
```

2. Run the application to generate a tracefile:

```
> mpiexec -n 4 myApp.exe
```

## Analyzing MPI Applications

For your convenience, this tutorial comes with a set of trace files available at <install-dir>/examples/traces.

### OS X\*:

On OS X\* Intel® Trace Collector is unavailable, therefore you cannot trace applications on this OS. Generate a tracefile on a Linux\* or Windows\* machine, or use the generated tracefiles from <install-dir>/examples/traces.

Once you have the trace data available for the `poisson` application, you are ready to start the analysis.

## 1.2. Starting Intel® Trace Analyzer

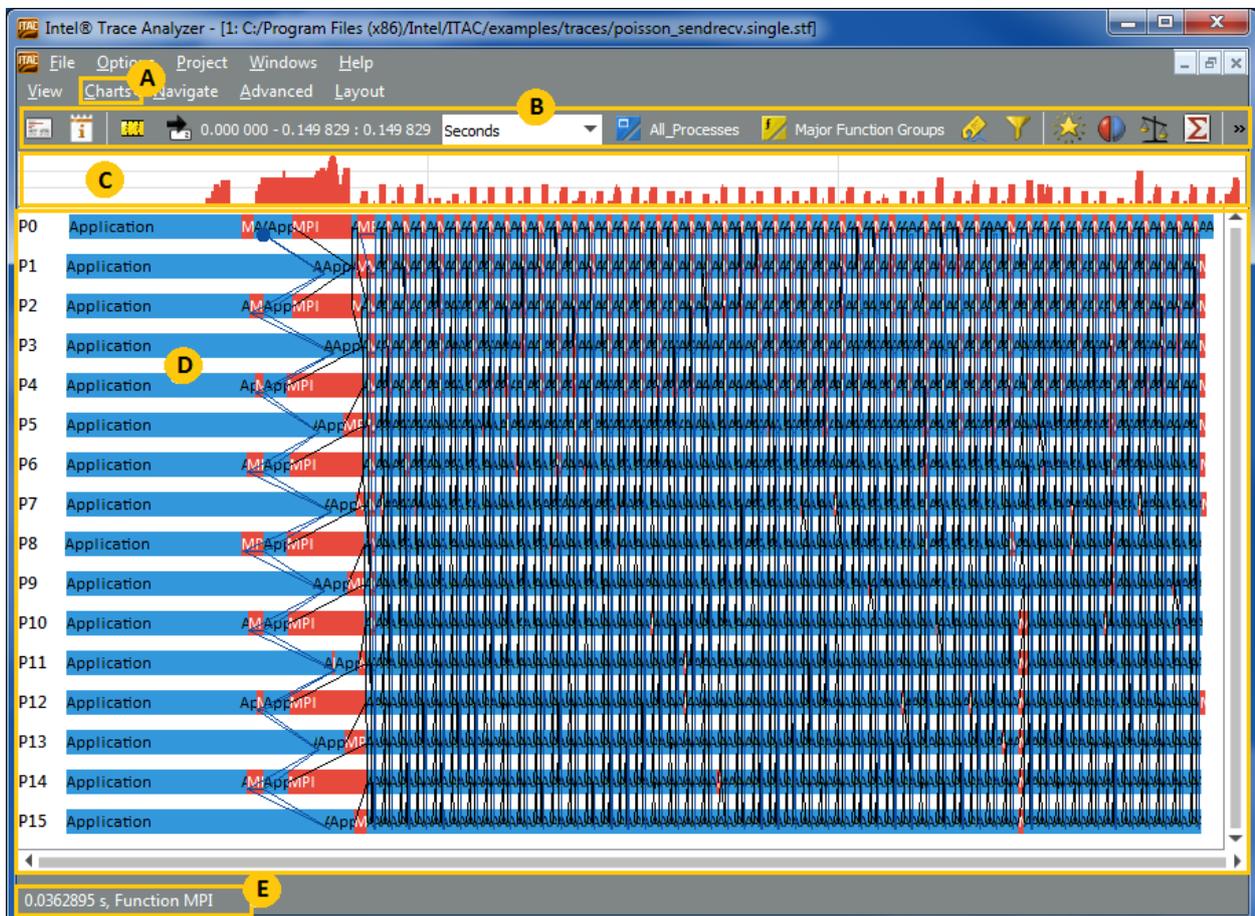
Invoke the Intel® Trace Analyzer GUI.

On Linux\* OS and OS X\*, enter the command:

```
$ traceanalyzer
```

On Windows\* OS, navigate to an `.stf` file and double-click to open it in the Intel Trace Analyzer.

### Intel® Trace Analyzer GUI



<b>A</b>	Use the <b>Charts</b> menu to open and navigate the various Intel Trace Analyzer charts within the current tracefile and use them to analyze the application trace data.
<b>B</b>	Use the Toolbar buttons to control the display of the currently open trace file.
<b>C</b>	The Trace Map displays the MPI function activity for the application over time. MPI function activity is displayed in red. Drag your mouse on a section in the Trace Map to zoom into the relevant subsets of tracefile

	charts. This map appears for all the charts.
<b>D</b>	The currently open chart is the Event Timeline. This chart displays individual process activities over time. Horizontal bars represent the processes with the functions called in these processes. The bars consist of colored rectangles labeled with the function names. Black lines indicate messages sent between processes. These lines connect sending and receiving processes. Blue lines represent collective operations, such as broadcast or reduce operations. To change the displayed chart, go to <b>Charts</b> .
<b>E</b>	The Status Bar displays the exact time point and function type when you hover the mouse over the processes shown in the Event Timeline.

## 1.3. Starting Intel® VTune™ Amplifier XE

Invoke the Intel® VTune™ Amplifier XE GUI and continue the analysis of the `poisson` application.

On Linux\* OS and OS X\*, run the command:

```
$ ampxe-gui
```

On Windows\* OS:

- From the Windows\* Start menu, choose **All Programs > Intel® Parallel Studio XE version > Analyzers > Intel VTune Amplifier XE version**.
- On Windows 8, find the Intel VTune Amplifier XE shortcut on the Start screen.

### Intel® VTune™ Amplifier XE GUI

The screenshot displays the Intel VTune Amplifier XE GUI. The main window shows the 'Basic Hotspots' view, which is a table of CPU time usage for various functions. The table is titled 'Hotspots by CPU Usage viewpoint' and includes columns for 'Function / Call Stack', 'Effective Time by Utilization', 'Spin Time', 'Ove... Time', and 'Module'. The 'Effective Time by Utilization' column shows a bar chart for each function, with colors indicating utilization levels: Idle (white), Poor (red), Ok (orange), Ideal (green), and Over (blue). The 'Function / Call Stack' column lists functions such as 'initialize\_2D\_buff', 'grid\_intersect', 'sphere\_intersect', 'GdipDrawImageF', 'grid\_bounds\_int', 'KiUserCallbackDi', 'tri\_intersect', 'pos2grid', and 'libm\_sse2\_sart'. The 'Effective Time by Utilization' column shows values like 9.748s, 2.916s, 2.388s, 0.421s, 0.421s, 0.094s, 0.078s, 0.063s, and 0.063s. The 'Spin Time' and 'Ove... Time' columns show values of 0s. The 'Module' column lists modules like 'find\_hotspots.exe', 'gdipplus.dll', 'ntdll.dll', and 'MSVCR110.dll'. The 'Data Of Interest (CPU Metrics)' panel on the right shows '100.0% (9.748s of 9.748s)'. The 'Thread' panel at the bottom shows 'thread\_video (TID: 7840)', 'WinMainCRTStartup (TI', and 'func@0x25cf784b (TID:'. The 'CPU Usage' panel at the bottom shows 'CPU Usage'.

- A** The **Project Navigator** shows projects and analysis results in hierarchy view. Click the **Project Navigator** button  on the toolbar to enable/disable the **Project Navigator**.

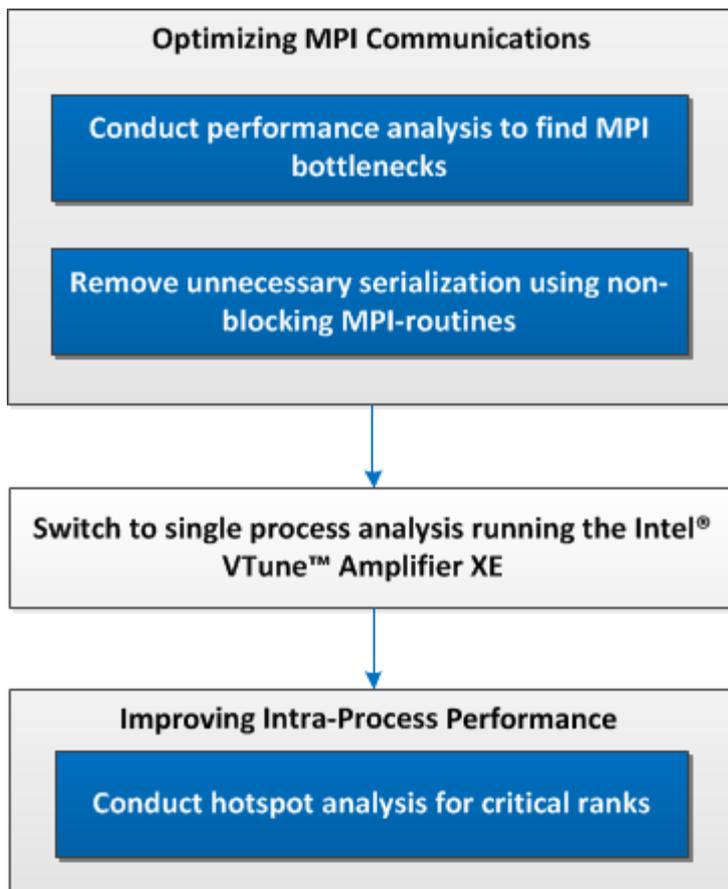
## Analyzing MPI Applications

<b>B</b>	The viewpoint header indicates the preset configuration of windows/panes for an analysis result. Click the <b>(change)</b> link to change the viewpoint. For each analysis type, you can switch among several viewpoints to focus on particular performance metrics. Click the question mark icon  to read the viewpoint description.
<b>C</b>	Switch between window tabs to explore the analysis type configuration options and collected data provided by the selected viewpoint.
<b>D</b>	Use the <b>Grouping</b> drop-down menu to choose a granularity level for grouping data in the grid.
<b>E</b>	Use the filter toolbar to filter out the result data according to the selected categories.

## 2. Analyzing an MPI Application

Use Intel® Trace Analyzer to conduct a performance analysis and tune MPI communications between processes. Then switch to the Intel® VTune™ Amplifier XE to carry out a new round of analysis and identify hotspots on specific processes.

This tutorial uses the `poisson` sample code as well as the sample trace files `poisson_sendrecv.single.stf` and `poisson_icommm.single.stf` to demonstrate the interoperability of the Intel Trace Analyzer and Intel VTune Amplifier XE that enables you to analyze and further tune the application.



<p> <b>Step 1: Optimize MPI communications</b></p>	<ul style="list-style-type: none"> <li>• Detect serialization in Function Profile and Message Profile.</li> <li>• Find imbalance on the inter-process level by comparing the original application with the idealized one.</li> <li>• Remove serialization.</li> </ul>
<p> <b>Step 2: Improve intra-process performance</b></p>	<ul style="list-style-type: none"> <li>• Identify hotspots on the intra-process level.</li> <li>• Interpret the collected data to find MPI processes in the workload and find possible ways to resolve the issue.</li> </ul>

### 2.1. Optimizing MPI Communications



Use the Intel® Trace Analyzer to analyze an MPI application's behavior to improve performance at the inter-process level.

## Analyzing MPI Applications

This part of the tutorial uses the sample trace files `poisson_sendrecv.single.stf` and `poisson_icomms.single.stf` to demonstrate how to detect and remove serialization in your application.

<b>Step 1: Prepare for analysis</b>	Use the Intel Trace Analyzer Event Timeline chart to zoom in to a single iteration of your application.
<b>Step 2: Detect serialization</b>	<ul style="list-style-type: none"><li>• Ungroup MPI functions to analyze MPI process activity in your application.</li><li>• Analyze your application with the Function Profile and Message Profile charts.</li><li>• Compare the original tracefile with the idealized trace to identify problematic interactions.</li></ul>
<b>Step 3: Remove serialization</b>	Improve your application performance by replacing the problem-causing function.
<b>Step 4: Check the result</b>	Use the Intel Trace Analyzer Comparison chart to compare the serialized application with the revised one.
<b>Step 5: Analyze optimized communications</b>	Analyze the revised application in the Event Timeline to see if the revised code needs further optimization.

## Key Terms

Idealized Tracefile

Serialization

### 2.1.1. Prepare for Analysis



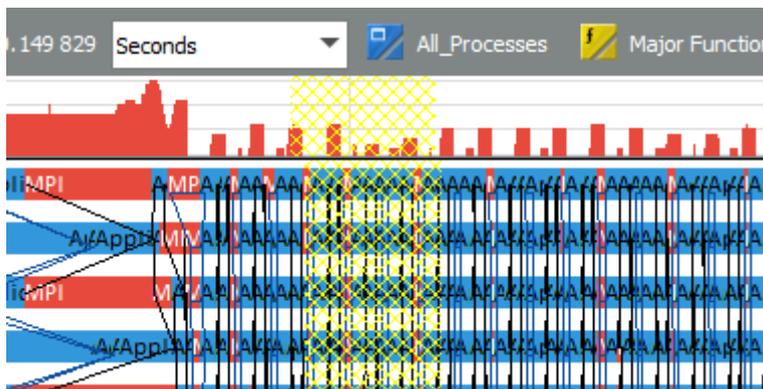
To analyze the application, start with the following steps:

1. Open the `poisson_sendrecv.single.stf` sample trace file.
2. Go to **Charts > Event Timeline** to open the Event Timeline.

#### NOTE

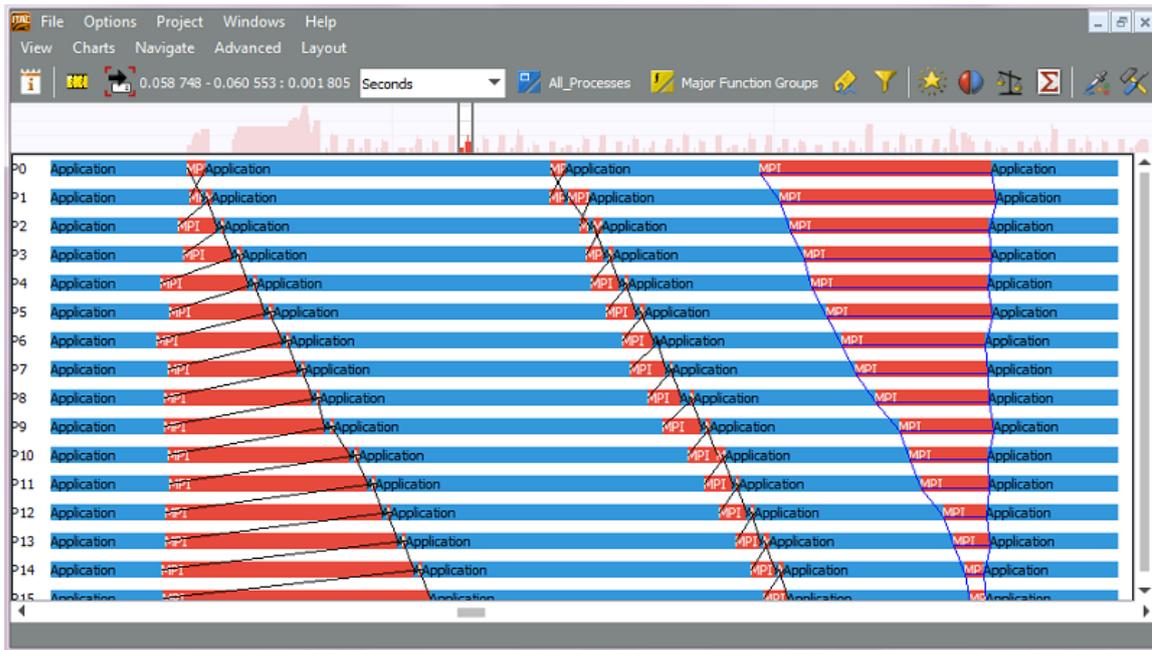
When you open a new trace file, Function Profile and Performance Assistant charts open by default. You can change the default chart in the Preferences dialog box (**Options > Preferences > Tracefile preferences**).

3. In the Event Timeline, click and drag your mouse over a specific time interval to zoom into it.



4. You should start noticing the iterative nature of the application. Zoom deeper into the trace by selecting a single iteration.

This is the view of the zoom. The Trace Map shows the section within the trace that is displayed. The Event Timeline chart shows the events that were active during the selected time.

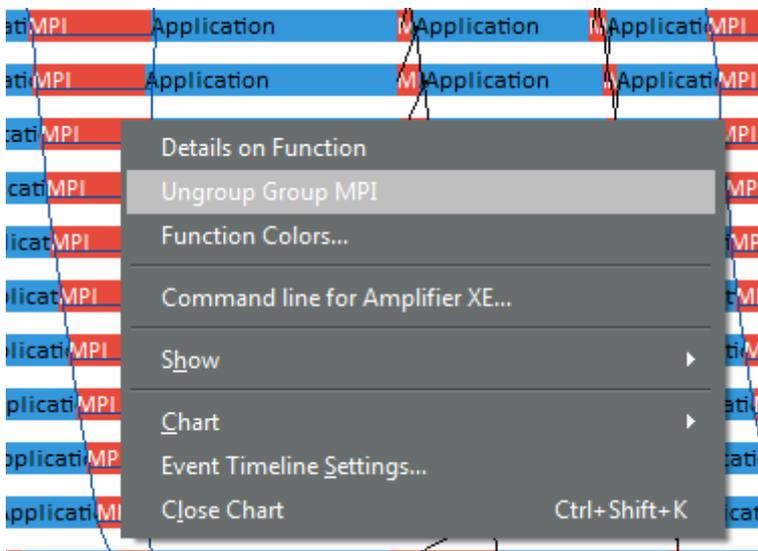


## 2.1.2. Ungroup MPI Functions



Analyze MPI process activity in your application.

To see the particular MPI functions called in the application, right-click on MPI (marked with a red rectangle) in the Event Timeline and select **Ungroup Group MPI**. This operation exposes the individual MPI calls.



After ungrouping the MPI functions, you see that the processes communicate with their direct neighbors using MPI\_Sendrecv at the start of the iteration.

## Analyzing MPI Applications



This data exchange has a disadvantage: process  $i$  does not exchange data with its neighbor  $i+1$  until the exchange between  $i-1$  and  $i$  is complete. This delay appears as a staircase pattern resulting with the processes waiting for each other.

The `MPI_Allreduce` at the end of the iteration resynchronizes all processes; that is why this block has the reverse staircase appearance.

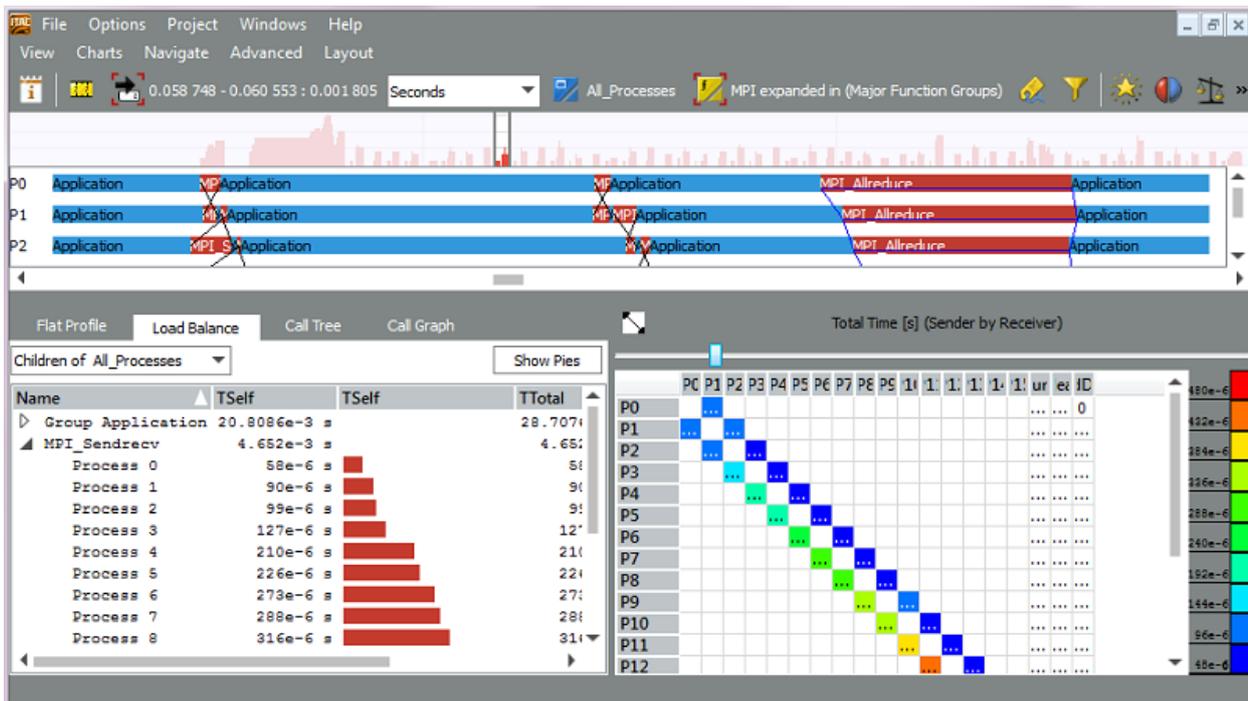
### 2.1.3. Detect Serialization in Function Profile and Message Profile



Continue the analysis of your application with information provided by other charts.

In the Function Profile chart, open the **Load Balance** tab.

Go to the **Charts** menu to open a Message Profile.



In the Load Balance tab, expand `MPI_Sendrecv` and `MPI_Allreduce`. The Load Balancing indicates that the time spent in `MPI_Sendrecv` increases with the process number, while the time for `MPI_Allreduce` decreases.

Examine the Message Profile Chart down to the lower right corner. The color coding of the blocks indicates that messages travelling from a higher rank to a lower rank need proportionally more time while the messages travelling from a lower rank to a higher rank reveal a weak even-odd kind of pattern.

## Key Terms

Serialization

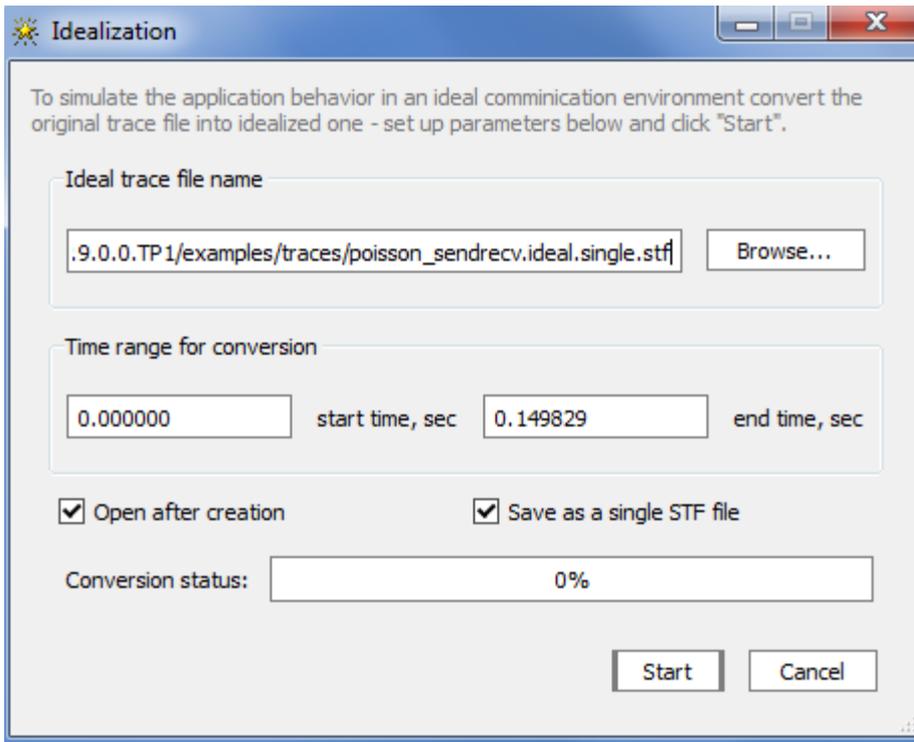
### 2.1.4. Compare Original Trace File With Idealized Trace File



See your application under the ideal circumstances and compare the original trace file with the idealized one to isolate problematic interactions.

Create the idealized trace:

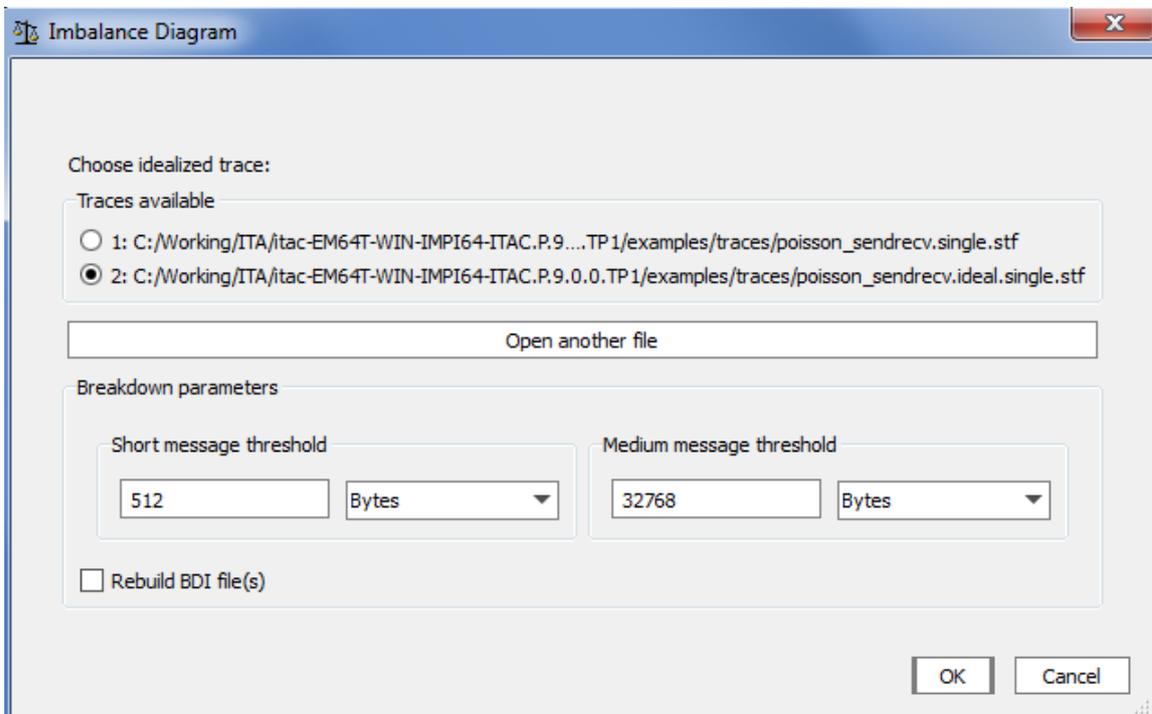
1. In the `poisson_sendrecv.single.stf` view, select **Advanced > Idealization**, or use the  toolbar button.
2. In the Idealization dialog box, check the idealization parameters. By default, Intel® Trace Analyzer stores the idealized trace in the local folder where the original trace file was opened. The default name of the new ideal trace file is the input trace file name with the suffix `ideal` added before the `.stf` extension.
3. Click **Start** to idealize the trace `poisson_sendrecv.single.stf`.



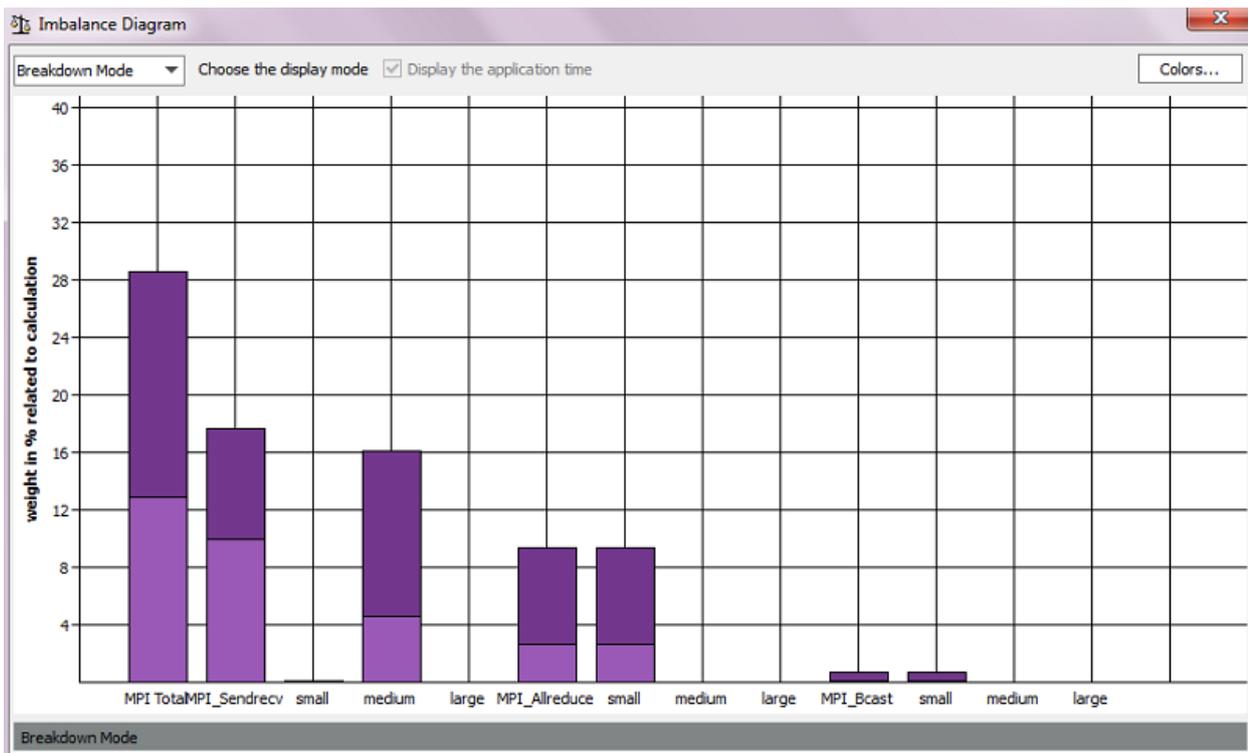
To get more information on idealization, refer to the *Idealization Dialog Box* section of the *Intel® Trace Analyzer User and Reference Guide*.

Compare the original trace file with the idealized trace:

1. In the `poisson_sendrecv.single.stf` view, select **Advanced > Imbalance Diagram** or press the  toolbar button.
2. In the Imbalance Diagram dialog box, press the **Open Another File** button, navigate to the idealized trace `poisson_sendrecv.ideal.stf` and select it.



3. Click **OK**.
4. In the Imbalance Diagram window, click the **Total Mode** button and select **Breakdown Mode**.



This chart shows you a breakdown of the interconnect vs. imbalance overhead in your application, as well as which MPI routines are at fault. You can see that `MPI_Sendrecv` is the most time-consuming function. The imbalance weight is displayed in the light purple color and comprises about 10% for the `MPI_Sendrecv` function. This is the time the processes spend waiting for each other.

## Key Terms

[Idealized trace file](#)

## 2.1.5. Remove Serialization



You can improve the performance of the `poisson` sample program by replacing the blocking `MPI_Sendrecv` with non-blocking communications via `MPI_Isend`. The modified source file `pardat.f90_icom` is available in the source folder. The trace file of the modified program is also available: `<install-dir>/examples/traces/poisson_icom.single.stf`.

Once corrected, the single iteration of the revised program will look similar to this:

## Analyzing MPI Applications



Since `poisson_sendrecv.single.stf` is a striking example of serialization, almost all of the Intel® Trace Analyzer charts show this interesting pattern. But in the real-world cases, it may be necessary to formulate a hypothesis regarding how the program should behave and to check this hypothesis using the most suitable chart.

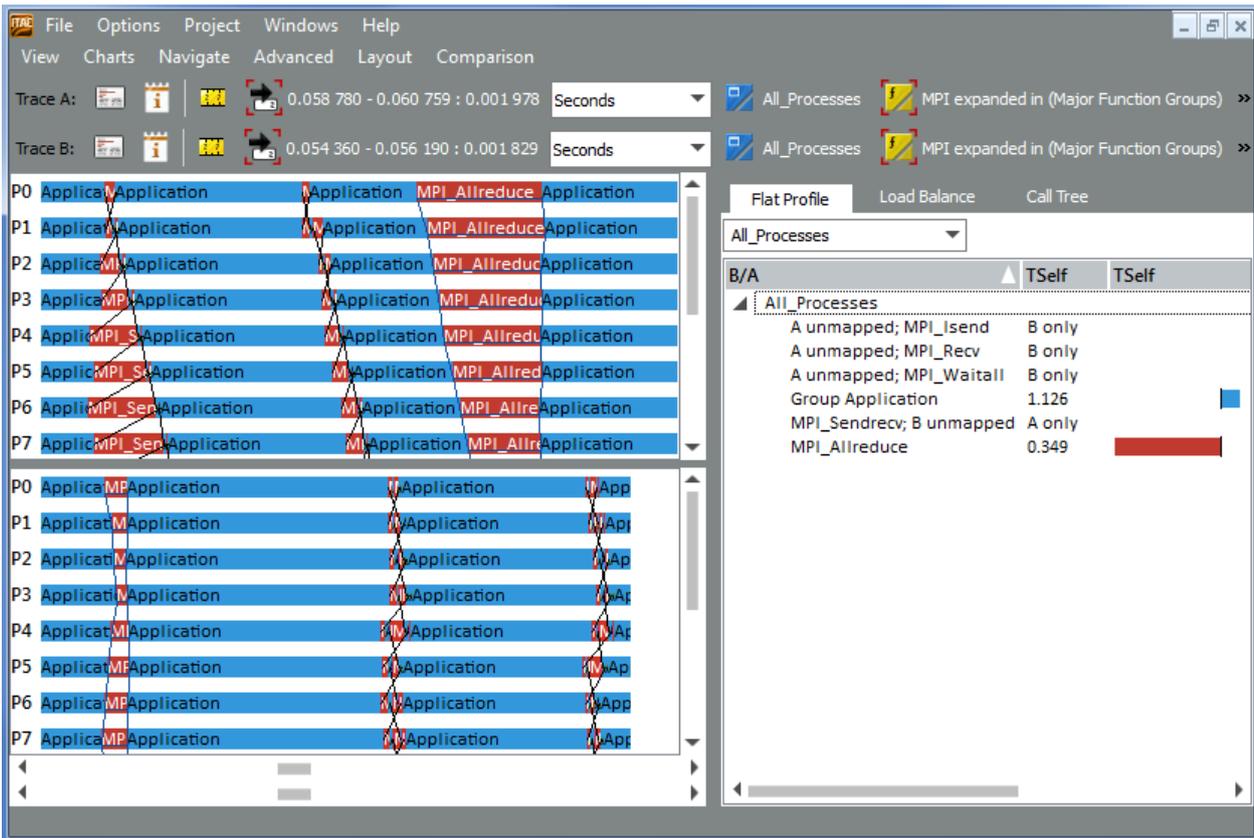
### Key Terms

Serialization

## 2.1.6. Compare Two Trace Files



Compare two trace files with the help of the **Comparison View**. To open a Comparison View for the original application trace file (`poisson_sendrecv.single.stf`), go to **View > Compare**. In the dialog that appears, choose the trace file of the revised application (`poisson_icomm.single.stf`). The Comparison View shows an Event Timeline for each trace file and a Comparison Function Profile chart. Zoom into the first iteration in each trace file. The Comparison View looks similar to:



In the Comparison View, you can see that using non-blocking communication helps to remove serialization and decrease the time of communication of processes.

## Key Terms

Serialization

### 2.1.7. Analyze Optimized Communications



Investigate other instances of time-consuming MPI calls in the revised application. To do this:

1. Open the **Summary Page** for the `poisson_icomm.single.stf` to see the MPI- and CPU-time ratio and get information about the most time-consuming MPI functions. To open the **Summary Page**, click the toolbar button.

Top MPI functions

This section lists the most active MPI functions from all MPI calls in the application.

MPI_Recv	0.121 sec (5.47 %)
MPI_Allreduce	0.0713 sec (3.22 %)
MPI_Isend	0.029 sec (1.31 %)
MPI_Bcast	0.0265 sec (1.19 %)
MPI_Comm_rank	0.0201 sec (0.904 %)

You can see that it is the `MPI_Recv` function that consumes 5.47% of the MPI time. To take a closer look at the function, click **Continue** here and go to the **Event Timeline**.

2. Open the **Event Timeline**, zoom into the first bunch of MPI communications and ungroup grouped MPI calls.
3. Zoom deeper to see the `MPI_Recv` calls closer:

## Analyzing MPI Applications



You can see that the processes starting from P1 have already called MPI\_Recv, but are waiting for P0 to send data. This kind of communication generates imbalance which is later reduced by MPI\_Allreduce resynchronizing all the MPI\_Recv calls.

- To prove that it is exactly the MPI\_Recv call that causes imbalance, analyze the application with Intel® VTune™ Amplifier XE.

## Key Terms

Serialization

## 2.2. Improving Intra-process Performance



Use the Intel® VTune™ Amplifier to analyze an MPI application behavior to improve the application performance on the intra-process level.

This part of the tutorial uses the `poisson` sample code to demonstrate how to detect and remove hotspots in your application.

<b>Step 1: Prepare for and run Basic Hotspots analysis</b>	Run the Basic Hotspots analysis to identify the functions that took a relatively long time to execute.
<b>Step 2: Identify hotspots and interpret results</b>	Explore the application-level performance, analyze the most time-consuming functions, and identify the hotspot code region.

### 2.2.1. Run Basic Hotspots Analysis



Open the **Summary Page** in Intel® Trace Analyzer and copy the command line for Intel® VTune™ Amplifier XE:

Use the following command line to run Intel® VTune™ Amplifier XE for the most CPU-bound rank:

```
mpirun -gtool "amplxe-cl -collect hotspots -r result:1" -n 4
./poisson inp
```

## NOTE

This information is available only for tracefiles generated on Linux\* OS. On Windows\* OS, make up the command line manually, see the example below.

Run the command line to perform the hotspot analysis with Intel VTune Amplifier.

### Linux OS:

```
& mpirun -gtool "amplxe-cl -collect hotspots -r result:1" -n 16 ./poisson inp
```

### Windows OS:

```
> mpiexec -gtool "amplxe-cl -collect hotspots -r result:1" -n 16 ./poisson inp
```

Navigate to the `result.1` directory and open the `poisson.1.amplxe` file in Intel VTune Amplifier.

## Key Terms

Hotspot

## 2.2.2. Interpret Results



Explore the application-level performance:

1. Intel® VTune™ Amplifier XE opens with the Summary page. Use this page as a starting point for the analysis of your application. In the **Elapsed Time** section of the Summary page, find out the elapsed time. For the current application it is 0.463 seconds:

### Elapsed Time: 0.463s

Total Thread Count:	1
Overhead Time:	0s
Spin Time:	0s
CPU Time:	0.080s
Paused Time:	0s

This display also indicates that this is a single-threaded application with the CPU time equal to 0.080 seconds.

2. In the **Top Hotspot** section, see the most time-consuming functions. For the `poisson` application, they are `poisson_red_black_` and `mpi_recv`.

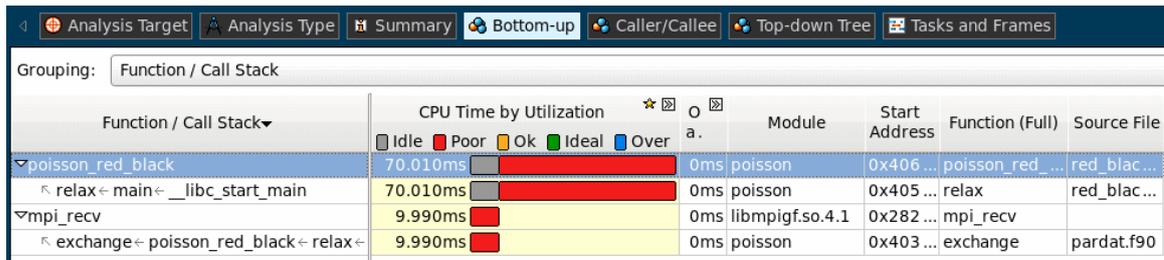
### Top Hotspots

This section lists the most active functions in your performance.

Function	CPU Time
<code>poisson_red_black</code>	0.070s
<code>mpi_recv</code>	0.010s

3. To analyze the most time-consuming functions, click the **Bottom Up** tab. Take a look at the CPU Time column, in which you can see that it took 70.010 milliseconds to execute the most time consuming function of the application and 9.990 milliseconds to execute `MPI_Recv`.

## Analyzing MPI Applications



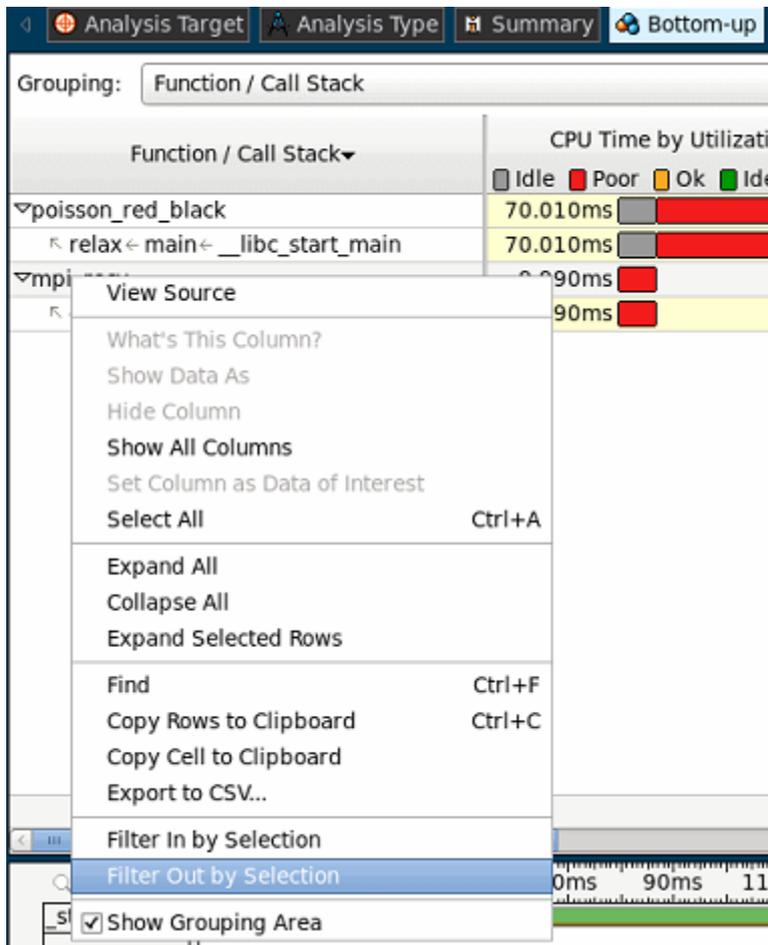
Function / Call Stack	CPU Time	Utilization	Module	Start Address	Function (Full)	Source File
poisson_red_black	70.010ms	Poor	poisson	0x406...	poisson_red_...	red_blac...
relax ← main ← _libc_start_main	70.010ms	Poor	poisson	0x405...	relax	red_blac...
mpi_recv	9.990ms	Poor	libmpigf.so.4.1	0x282...	mpi_recv	
exchange ← poisson_red_black ← relax ←	9.990ms	Poor	poisson	0x403...	exchange	pardat.f90

### NOTE

To see MPI functions under the **Bottom-Up** tab, make sure that **Call Stack Mode** at the bottom of the tab is set to **User Functions + 1**

It proves that the result we saw in the Intel® Trace Analyzer Event Timeline is correct: this is the `MPI_Recv` call that generates imbalance in the application. Since there is no need to optimize this kind of logical imbalance, proceed with the analysis.

- To see the imbalance created by the other function, filter the `MPI_Recv` out of the analysis scope. To do this, right-click the function at the **Bottom-Up** tab and select **Filter Out By Selection**, as shown in the example:



- Take a look at the function with poor CPU usage. Double-click the `poisson_red_black_` function to open the source and identify the hotspot code regions. The beginning of the hotspot function is highlighted. The source code in the **Source** pane is not editable.

### NOTE

To enable the **Source** pane, make sure to build the target with debugging symbols using the `-g` (Linux\* OS) and `/zi` (Windows\* OS) compiler flags.

6. For the `poisson` application, you can see the cycle in which computation took most of the CPU time.

247			
248	<code>x = x_iter</code>	19.996ms	19.996ms
249			
250	<code>WHERE( RED )</code>		
251	<code>u = quart*(f_disk+ u_north + u_west + u_east + u_south)</code>	10.000ms	10.000ms
252	<code>ENDWHERE</code>		
253			
254	<code>!Boundary exchange red points</code>		
255	<code>CALL exchange('RED', x_iter, imin, imax, jmin, jmax)</code>		
256			
257	<code>WHERE( .NOT.RED )</code>		
258	<code>u = quart*(f_disk+ u_north + u_west + u_east + u_south)</code>	29.998ms	29.998ms
259	<code>ENDWHERE</code>		
---			

Two options for resolving the issue are `vectorize`, or parallelize the cycle.

For more detailed explanations and more methods for analysis of your application, see the [Intel® Software Documentation Library](#) or [Intel® VTune™ Amplifier XE product page](#) and refer to the *Finding Hotspots* tutorials.

## Key Terms

[CPU time](#)  
[Elapsed time](#)  
[Hotspot](#)  
[Target](#)

## 3. Summary

You have completed the *Analyzing Application with Intel® Trace Analyzer and Collector and Intel® VTune™ Amplifier* tutorial. The following is the summary of important things to remember when using these tools to analyze and tune your application.

Step	Tutorial Recap	Key Tutorial Take-aways
<b>1. Optimize MPI communications</b>	<ul style="list-style-type: none"> <li>• Prepared for the application analysis.</li> <li>• Used the Event Timeline, Function Profile, Message Profile and Imbalance Diagram to detect serialization that slows down the application.</li> <li>• Removed serialization by replacing the problem-causing function.</li> <li>• Compared the original trace file with the trace file of the revised application.</li> <li>• Analyzed the improved communications in the Event Timeline.</li> </ul>	<ul style="list-style-type: none"> <li>• Ungroup MPI functions to identify which functions slow down the application.</li> <li>• Use the Function Profile and Message Profile charts to see how much time is spent in MPI.</li> <li>• Generate the idealized trace and compare it with the original trace to get an insight on your application under the ideal circumstances and isolate problematic interactions.</li> <li>• In the real-world cases, it may be necessary to formulate a hypothesis regarding how the program should behave and to check this hypothesis using the most suitable chart.</li> </ul>
<b>2. Improve intra-process performance</b>	<ul style="list-style-type: none"> <li>• Built the target and launched the Basic Hotspots data collection using the interoperability features of the tools.</li> <li>• Analyzed function calls and CPU time spent in each program unit of your application and identified the function that took the most CPU time.</li> <li>• Found possible way to resolve the issue and optimize the source code.</li> </ul>	<ul style="list-style-type: none"> <li>• Start analyzing the performance of your application from the <b>Summary</b> window to explore the performance metrics for the whole application.</li> <li>• Then, move to the <b>Bottom-up</b> window to analyze the performance per function. Focus on the <i>hotspots</i> - functions that took the most CPU time. By default, they are located at the top of the table.</li> <li>• Double-click the hotspot function in the <b>Bottom-up</b> pane or <b>Call Stack</b> pane to open its source code.</li> </ul>

**Next step:** Use the Intel® Trace Analyzer and Collector and Intel® VTune™ Amplifier to analyze your own application.

## 4. Key Terms

---

The following terms are used throughout this tutorial:

**CPU time:** The amount of time a thread spends executing on a logical processor. For multiple threads, the CPU time of the threads is summed. The application CPU time is the sum of the CPU time of all the threads that run the application.

**Elapsed time:** The total time your target ran, calculated as follows: Wall clock time at end of application - Wall clock time at start of application.

**hotspot:** A section of code that took a long time to execute. Some hotspots may indicate bottlenecks and can be removed, while other hotspots inevitably take a long time to execute due to their nature.

**idealized trace file:** A trace file of the application under ideal circumstances - infinite bandwidth and zero latency.

**serialization:** An effect in which a parallel program is reduced to serial execution due to blocking effects between execution units.

**target:** An executable file you analyze using the Intel® VTune™ Amplifier.