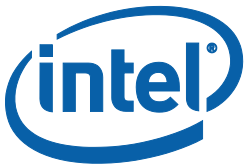


Tutorial: Detecting and Removing Unnecessary Serialization

Intel® Trace Analyzer and Collector

Contents

Legal Information

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.

This tutorial helps you improve the application performance by removing serialization. When your application is serialized, much time is spent on communication of processes slowing down the execution. Learn how to detect and remove serialization in your application using Intel® Trace Analyzer.

About This Tutorial	This tutorial demonstrates a workflow applied to a sample program. You can ultimately apply the same workflow to your own application(s): • Analyze the application with the help of the Intel® Trace Analyzer charts, to find communication problems • Detect serialization in communications between processes • Resynchronize the processes to remove serialization • Review the application
Estimated Duration	10-15 minutes
Learning Objectives	After you complete this tutorial, you should be able to: • Detect serialization in your application with the help of the Intel® Trace Analyzer • Improve overall performance
More Resources	Learn more about the Intel® Trace Analyzer and Collector in the User and Reference Guides: • *Intel® Trace Analyzer User and Reference Guide* • *Intel® Trace Collector User and Reference Guide* The guides are available at the Intel® Trace Analyzer and Collector Product Page.

You can submit your feedback on the documentation at http://www.intel.com/software/products/softwaredocs_feedback/.

1.1. Starting Intel® Trace Analyzer


Invoke the Intel® Trace Analyzer GUI.

Linux* OS:

Enter the command:

```
$ traceanalyzer poisson_sendrecv.single.stf
```

OS X*:

1. On the menu bar select **Go > Applications > Intel Trace Analyzer**.
2. Open the `poisson_sendrecv.single.stf` trace file from **File** > **Open**.

Windows* OS:

- From the command line prompt, enter the command:

  ```
  $ traceanalyzer poisson_sendrecv.single.stf
  ```

- From the graphical user interface, there are two options:
  - Go to **Start** > **All Programs** > **Intel Parallel Studio XE *version*** > **Analyzers** > **Intel Trace Analyzer *version*** and open the tracefile from **File** > **Open**.
  - Navigate to an `.stf` file and double-click to open it in Intel Trace Analyzer.

1.2. Introducing the Intel® Trace Analyzer GUI

A	Use the **Charts** 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	Use the Toolbar buttons to control the display of the currently open trace file.
C	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.
D	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 **Charts**.

E	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. Working with the Intel® Trace Analyzer and Collector Examples

You can find the Intel® Trace Analyzer and Collector examples directory at `<install_dir>/examples`. This topic explains how to create a trace file for the `vtjacobic` executable file. You can apply the same principles to other sample files and your own applications.

1.3.1. Compiling and Instrumenting Test Files on Linux* OS

Before analyzing sample trace files, set up your working directory as follows.

1. Copy the `examples` directory into a shared directory, which is accessible by all nodes of the cluster.
2. Clean up the directory content and compile and execute the C and Fortran executable files, entering the following commands:

   ```
   gmake distclean
   gmake all
   ```

   The resulting output will look as follows:

   ```
   vnallpair
   vnallpairc
   vnjacobic
   vnjacobif
   vtallpair
   vtallpairc
   vtcounterscopec
   vtjacobic
   vtjacobif
   ```

NOTE

The executable files listed above have already been linked with the appropriate Intel® Trace Collector libraries.

To analyze trace the `vtjacobic` executable, do the following:

1. Create a `vtjacobic_inst` directory and set the following environment variable:

   ```
   setenv VT_LOGFILE_PREFIX vtjacobic_inst
   ```

   This environment variable ensures that the trace files for the analysis appear in the created directory.
2. To run your MPI application, enter the command:

   ```
   mpirun -n 4 -trace ./vtjacobic
   ```

   The `vtjacobic_inst` directory should now contain the following files:

   ```
    .                    vtjacobic.stf.dcl          vtjacobic.stf.msg.anc
   ..                    vtjacobic.stf.frm          vtjacobic.stf.pr.0
   vtjacobic.prot        vtjacobic.stf.gop          vtjacobic.stf.pr.0.anc
   vtjacobic.stf         vtjacobic.stf.gop.anc      vtjacobic.stf.sts
   vtjacobic.stf.cache   vtjacobic.stf.msg
   ```

For details, refer to the Intel® MPI Library documentation.

1.3.2. Compiling and Instrumenting Test Files on Windows* OS

1. Create a shared directory which is accessible to all nodes of the cluster and copy the `examples` directory into it.
2. Clean up the directory content:

   ```
   nmake distclean
   ```

To compile and execute the C and Fortran executable files, enter the following command:

```
nmake all MPIDIR="<install-dir>\MPI\em64t"
```

The following C and Fortran executable files appear under the `examples` directory:

```
mpiconstants.exe
vnallpair.exe
vnallpairc.exe
vnjacobic.exe
vnjacobif.exe
vtallpair.exe
vtallpairc.exe
vtcounterscopec.exe
vtjacobic.exe
vtjacobif.exe
vttimertest.exe
```

NOTE

The `MPIDIR` makefile variable is explicitly set to the directory of the Intel MPI Library that supports 64-bit address extensions.

For the executable files above, the following STF files are created:

```
timertest.stf
vtallpair.stf
vtallpairc.stf
vtcounterscopec.stf
vtjacobic.stf
vtjacobif.stf
```

1.3.3. Analyzing Trace Data

To analyze trace files, do the following:

1. To analyze the newly created `vtjacobic.stf` trace file, enter the following commands.

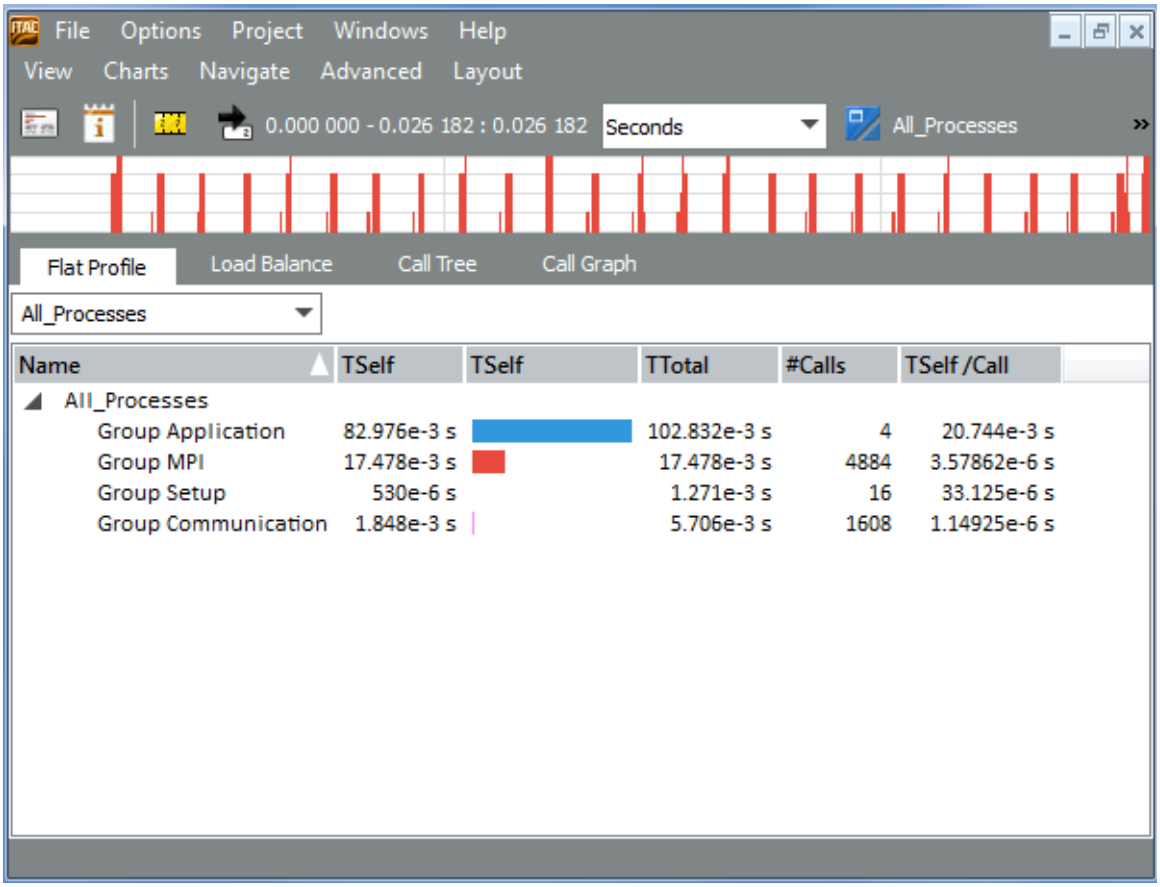
   On Linux* OS:

   ```
   $ traceanalyzer vtjacobic_inst/vtjacobic.stf
   ```

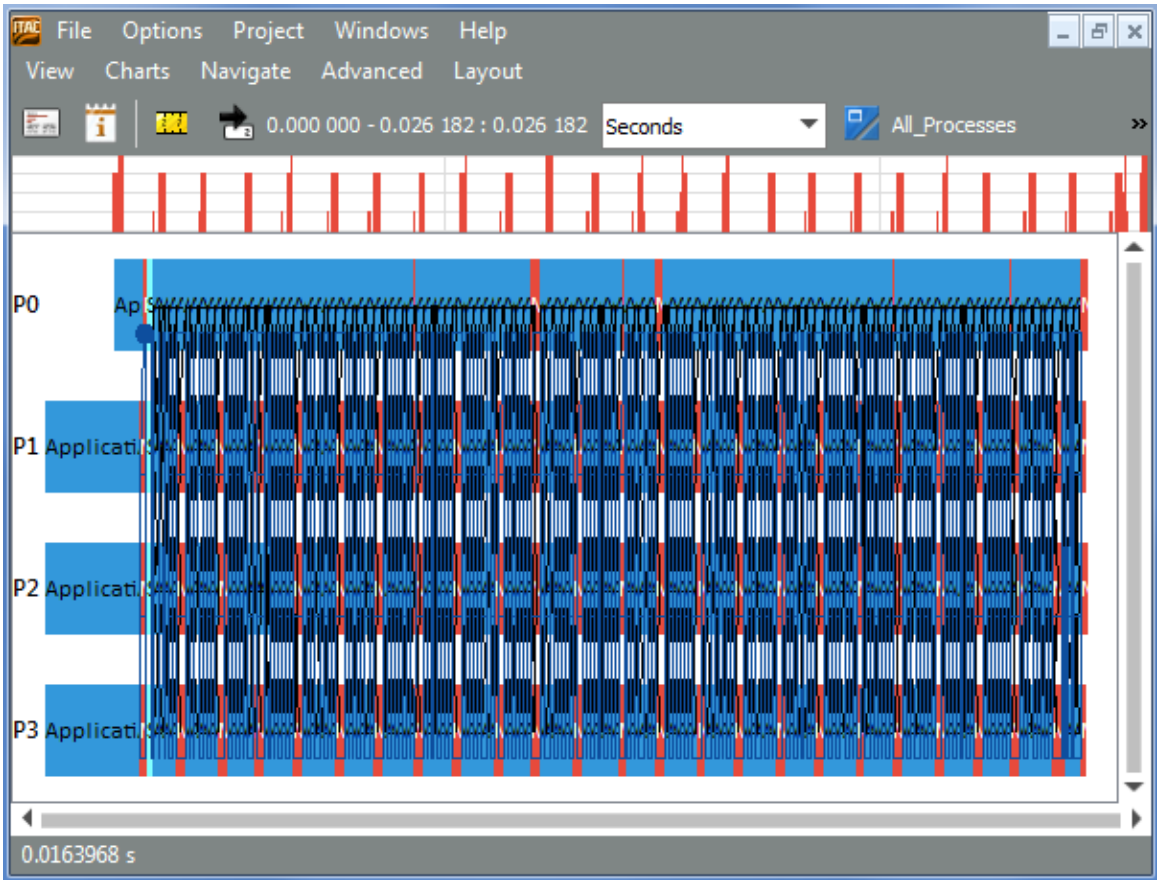
   On Windows* OS:

   ```
   > traceanalyzer vtjacobic.stf
   ```

   Intel® Trace Analyzer displays the **Flat Profile** tab for `vtjacobic.stf`:



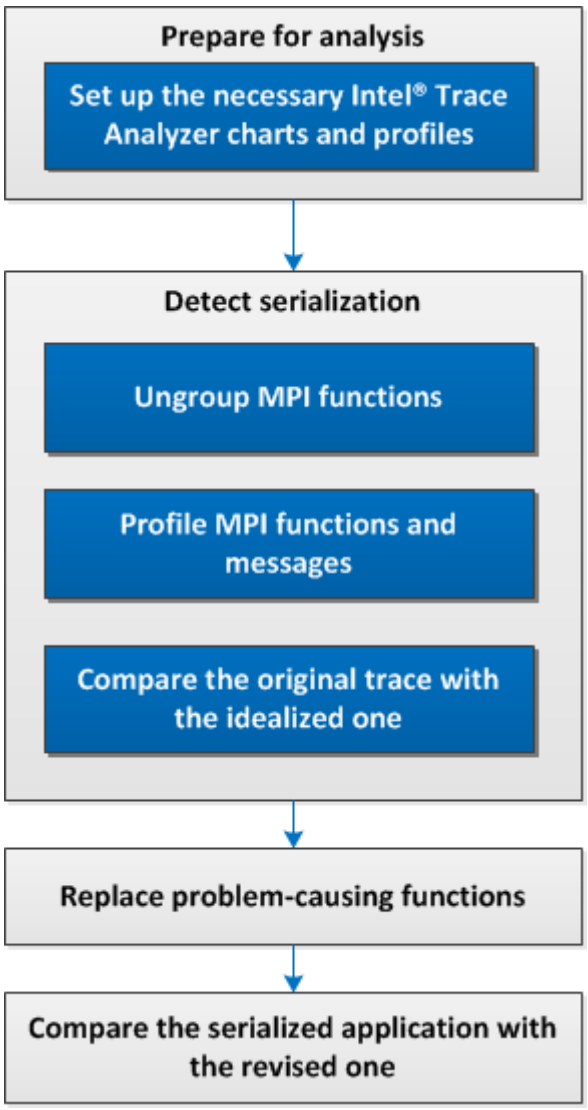
2. To view the **Event Timeline** chart, go to **Charts** > **Event Timeline**:



2. Detecting and Removing Serialization

Use the Intel® Trace Analyzer to analyze an MPI application behavior to improve the application performance.

This tutorial uses the sample trace files `poisson_sendrecv.single.stf`, `poisson_sendrecv.ideal.stf` and `poisson_icomm.single.stf` to demonstrate how to detect and remove serialization in your application.



Step 1: Prepare for analysis	Use the Intel Trace Analyzer Event Timeline chart to have a closer look at a single iteration of your application
Step 2: Detect serialization	• Ungroup MPI functions to analyze MPI process activity in your application • Analyze your application with Function Profile and Message Profile charts opened at the same time • Compare the original trace file with the idealized trace to identify problematic interactions
Step 3: Remove	Improve your application performance by replacing the problem-causing

serialization	function
Step 4: Check your work	Use the Intel Trace Analyzer Comparison chart to compare the serialized application with the revised one

Key Terms

Idealized Tracefile
Serialization

2.1. Prepare for Analysis

To analyze the application, start with the following steps:

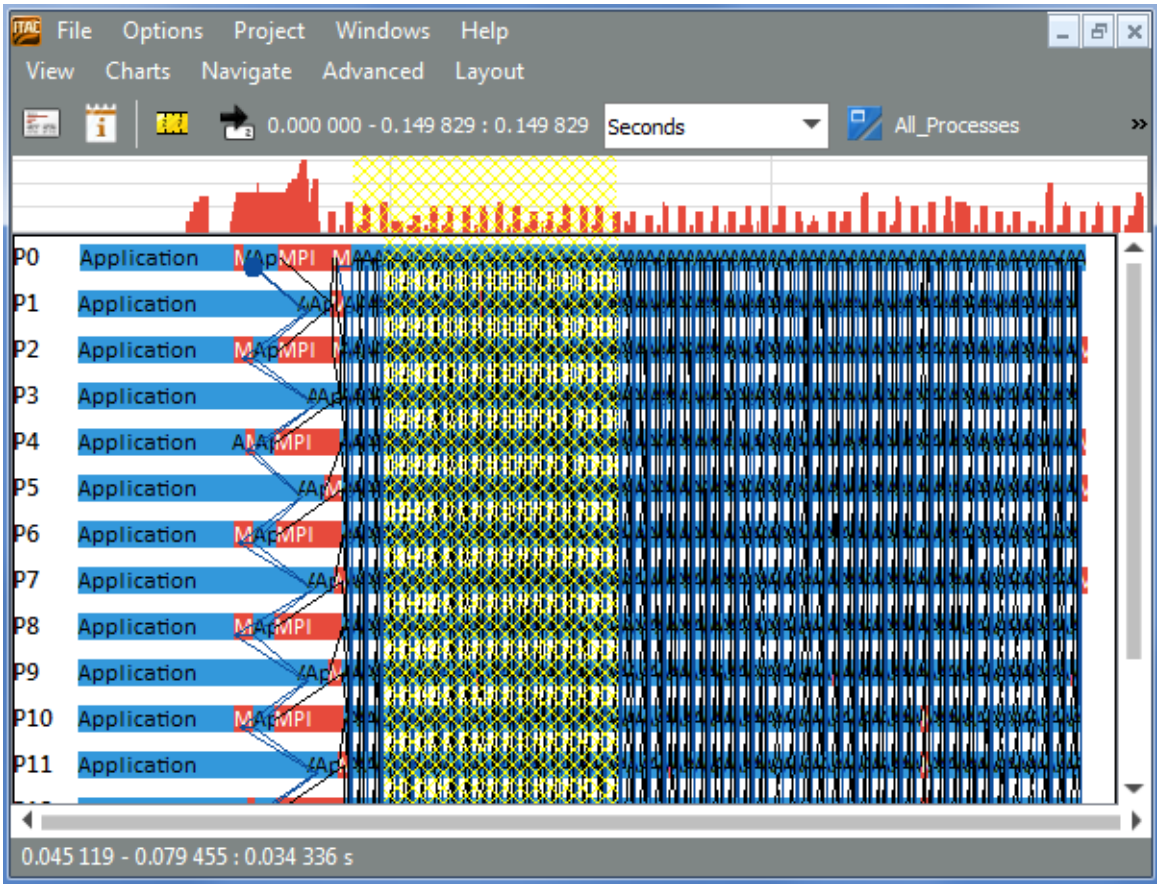
1. Open the `poisson_sendrecv.single.stf` sample trace file.
2. Go to **Charts > Event Timeline** to open the Event Timeline.

NOTE

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

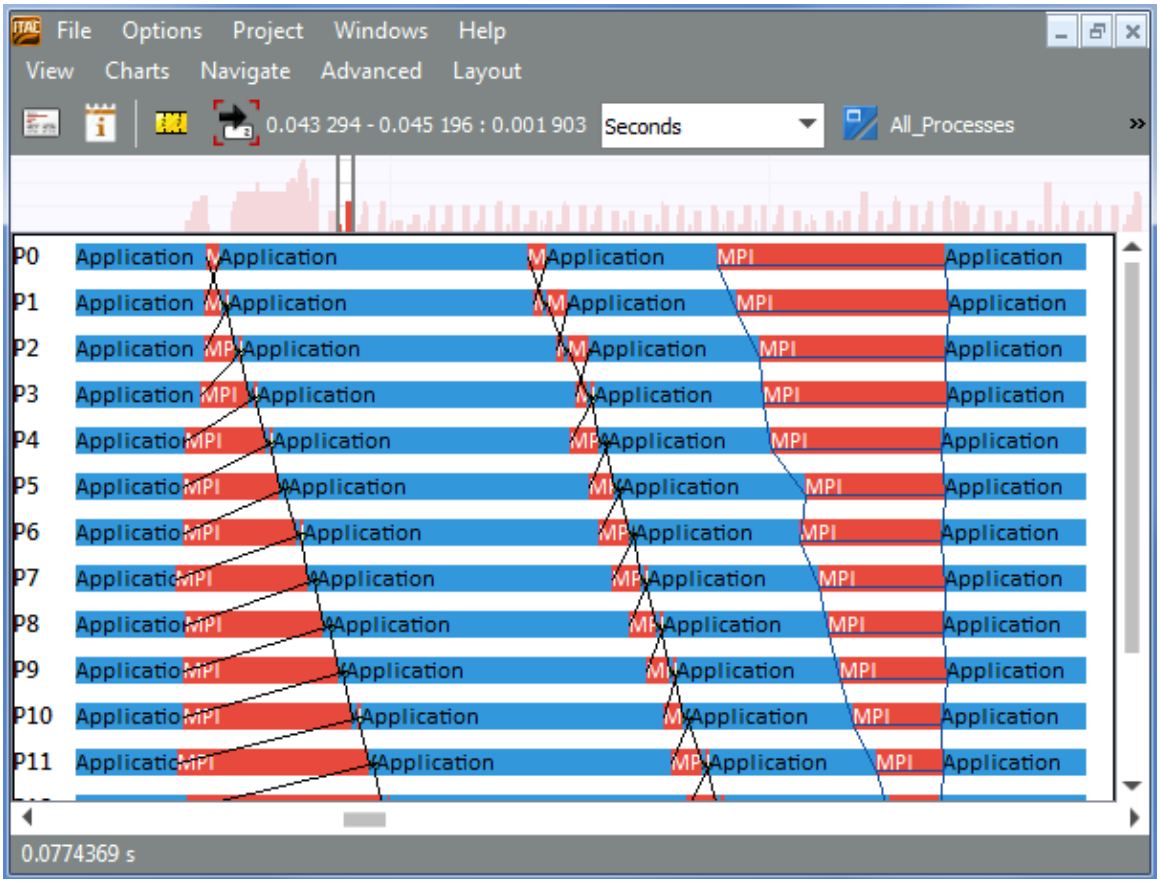
---

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



4. Zoom deeper into the trace by selecting the single iteration.

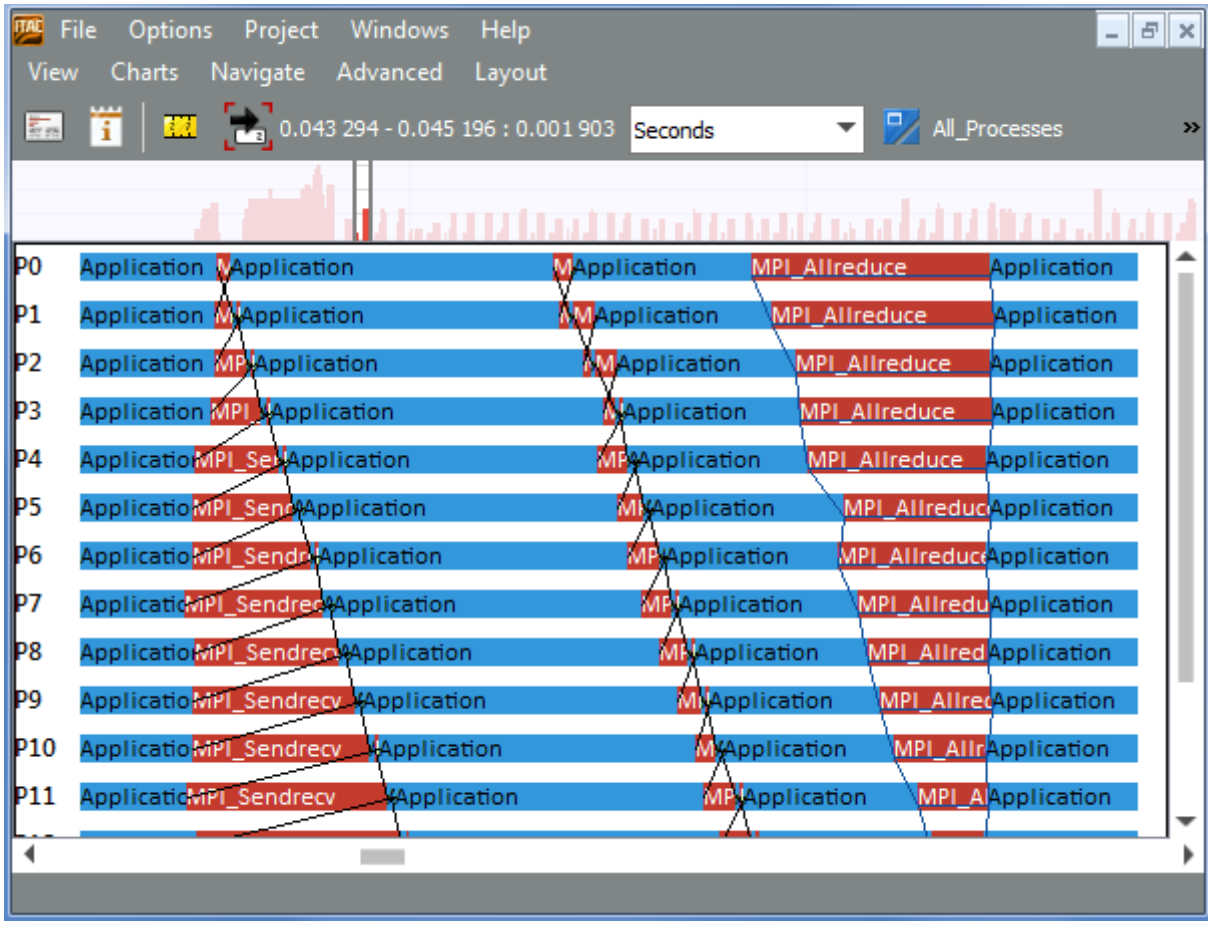
   This is the view of the zoom. The trace map shows the section within the trace that is displayed. The Event Timeline chart shows the events that were active during the selected time.



2.2. Ungroup MPI Functions

Analyze MPI process activity in your application.

To see the particular MPI functions called in the application, right-click on MPI in the Event Timeline and choose **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.

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 resulting with the processes waiting for each other.

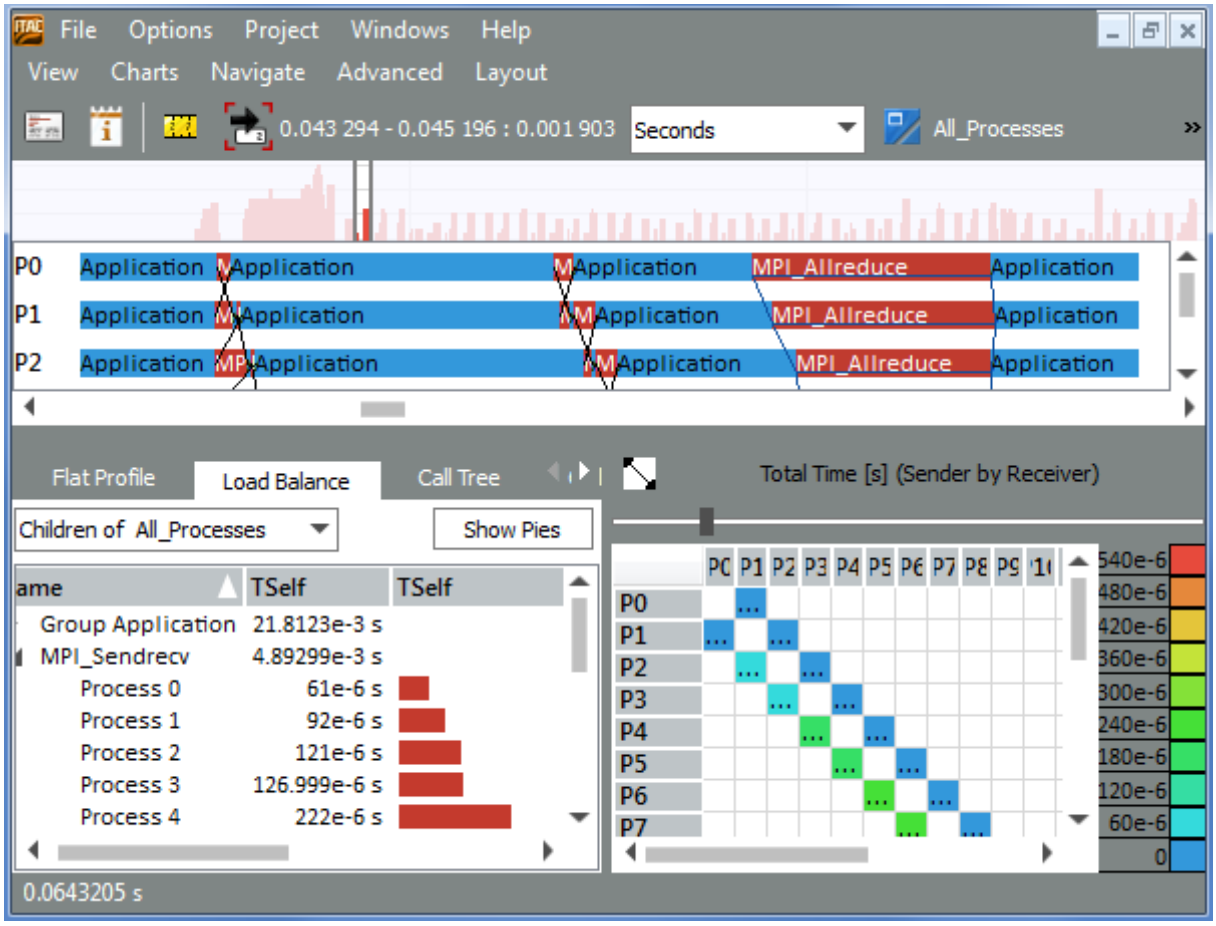
The `MPI_Allreduce` at the end of the iteration resynchronizes all processes; that is why this block has the reverse staircase appearance.

2.3. Detect Serialization in Function Profile and Message Profile

Analyze your application with several charts opened at the same time.

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

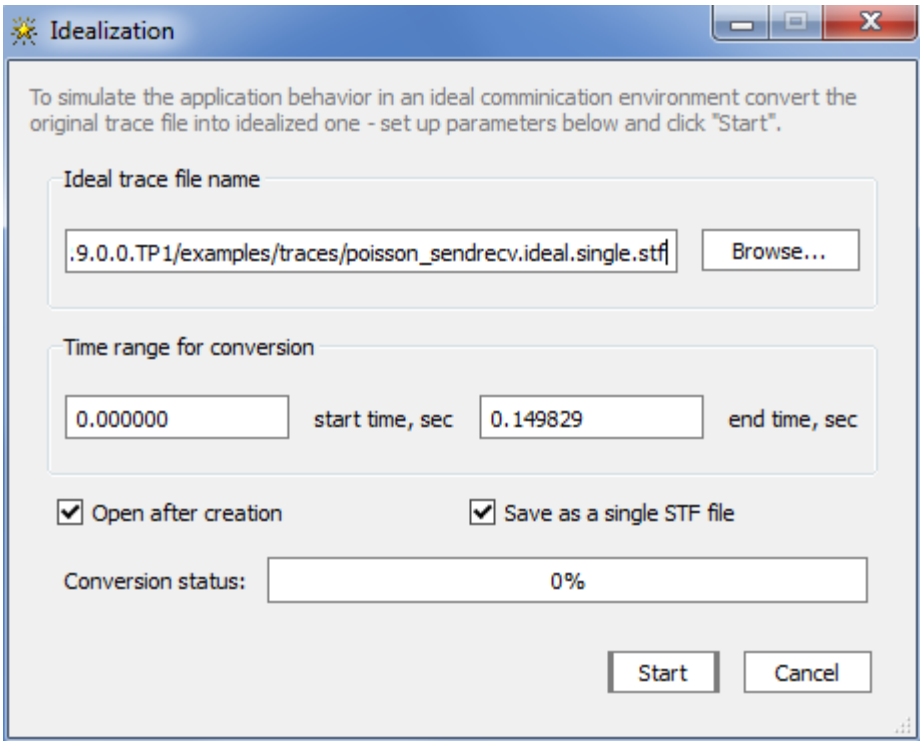
Serialization

2.4. 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:

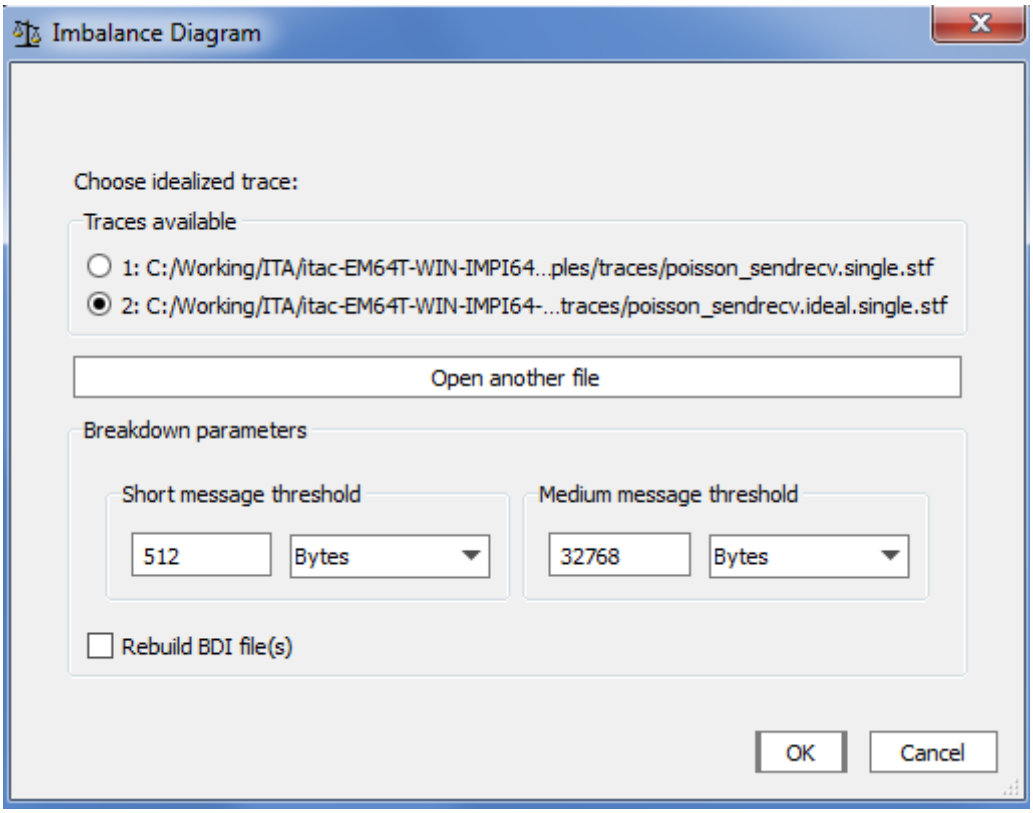
1. In the `poisson_sendrecv.single.stf` view, select **Advanced** > **Idealization**, or use the  toolbar button.
2. In the Idealization dialog box, check the idealization parameters. By default, Intel® Trace Analyzer stores the idealized trace in the examples folder under the name of the input trace file 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 Reference Manual*.

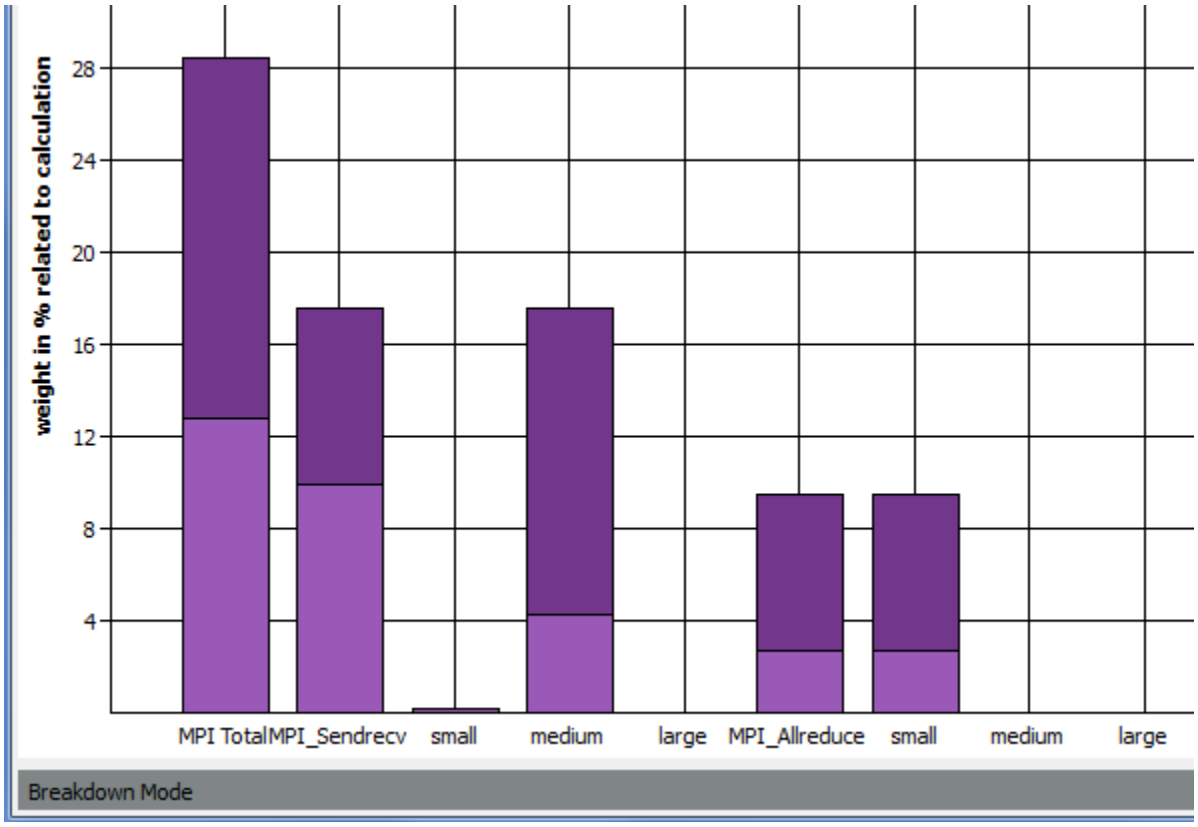
Compare the original trace file with the idealized trace:

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**.



You can see that `MPI_Sendrecv` is the most time-consuming function. The imbalance weight is displayed in pink color and comprises about 10% for the `MPI_Sendrecv` function. This is the time the processes spend waiting for each other.

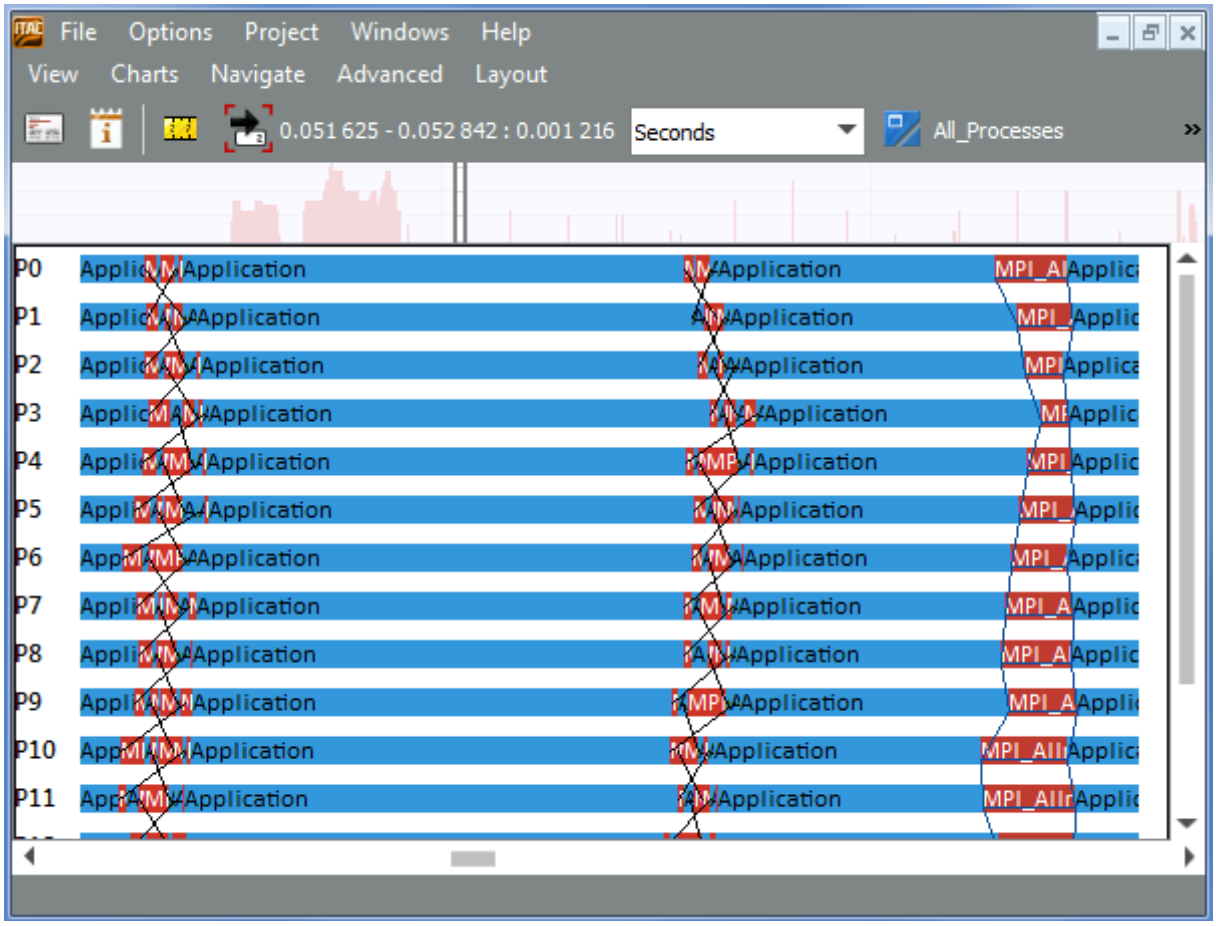
Key Terms

Idealized trace file

2.5. Remove Serialization

You can improve the performance of the `poisson` sample program by replacing the serial `MPI_Sendrecv` with non-blocking communication: `MPI_Isend` and `MPI_Irecv`.

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



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.

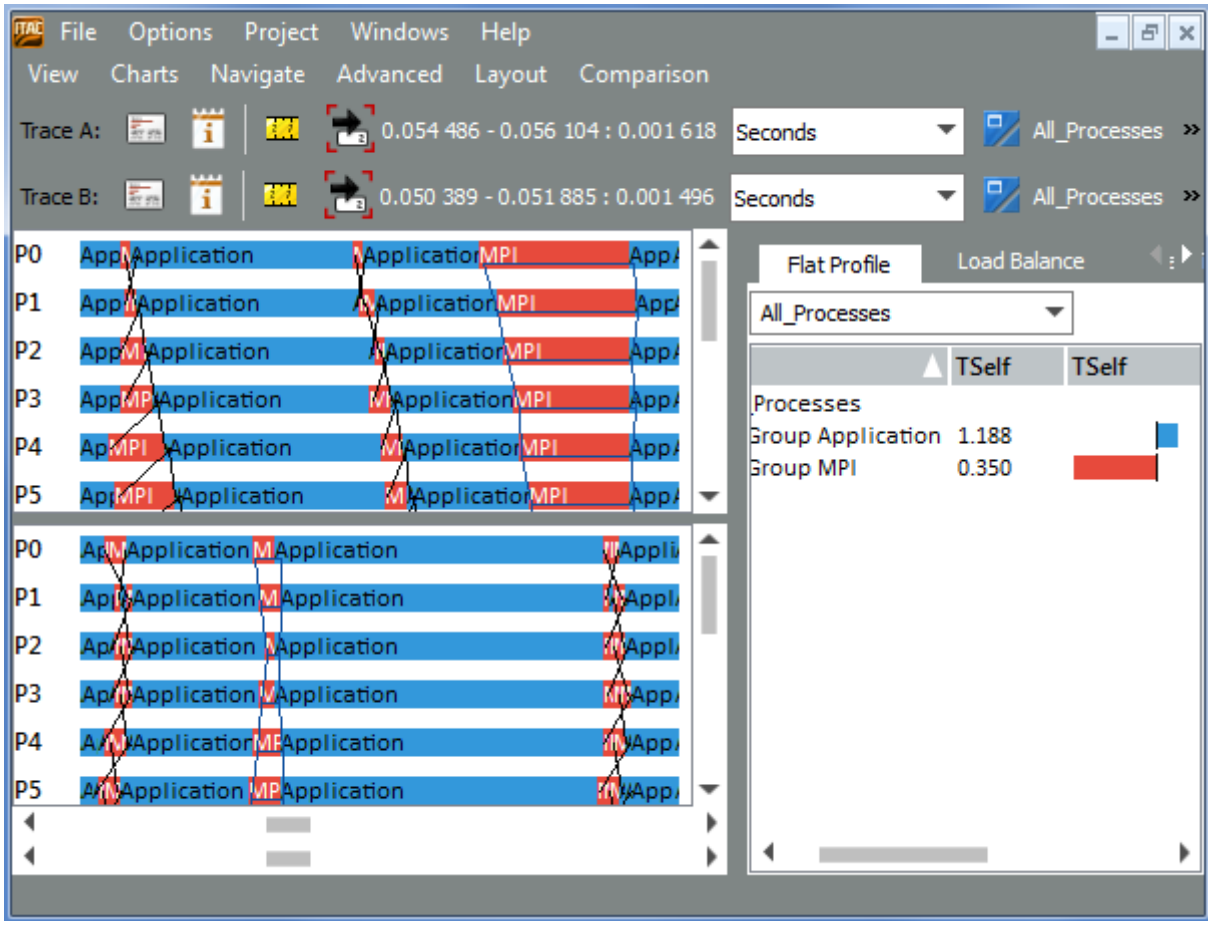
Key Terms

Serialization

2.6. 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

3. Summary

You have completed the *Detecting and Removing Unnecessary Serialization* tutorial. The following is the summary the important things to remember when using the Intel® Trace Analyzer for detecting and avoiding serialization in your application.

Step	Tutorial Recap	Key Tutorial Take-aways
1. Prepare for Analysis	Prepared for the application analysis.	Use the Event Timeline chart to see overall process activity and zoom into the trace to look at the single iteration of the application.
2. Detect Serialization	Used the Event Timeline, Function Profile, Message Profile and Imbalance Diagram to detect serialization that slows down the application.	• Ungroup MPI functions to identify which functions slow down the application. • Use the Function Profile and Message Profile charts to see how much time is spent in MPI. • 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.
3. Remove Serialization	Removed serialization by replacing the problem-causing function.	• Generate the trace file of the revised application to see if the application spends less time on communication now. • 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.
4. Check Your Work	Compared the original trace file with the trace file of the revised application.	

Next step: Generate your own trace file for analysis. Then use Intel® Trace Analyzer to detect and remove serialization in your application.

4. Key Terms

The following terms are used throughout this tutorial:

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

Serialization: The result of usage of blocking communication, in which 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 in the Event Timeline chart.