

Get Started with Intel® Trace Analyzer and Collector

Contents

Chapter 1: Get Started with Intel® Trace Analyzer and Collector	
Trace Your MPI Application	3
Analyze the Most Active MPI Functions	5
Identify Disbalanced Communications	8
Improve Your Application Performance by Changing Communications	9
Learn More	12
Notices and Disclaimers.....	13

Get Started with Intel® Trace Analyzer and Collector

1

Use this Get Started document and [a pre-collected trace file](#) to walk through a basic MPI performance analysis with Intel® Trace Analyzer and Collector.

Intel Trace Analyzer and Collector helps explore message passing interface (MPI) usage efficiency and identify communication hotspots, synchronization bottlenecks, and load balancing. For more information about the product, see [Intel Trace Analyzer and Collector product page](#).

NOTE

Intel Trace Analyzer and Collector 2022.3 is the final version. No further feature improvements or security fixes will be available after this final release. For more information, see the [Intel Trace Analyzer and Collector transition notice](#).

Download Intel Trace Analyzer and Collector

- as a standalone tool

Prerequisites

Before running Intel Trace Analyzer and Collector, make sure you have installed the latest Intel MPI Library and Intel oneAPI DPC++/C++ Compiler or Intel Fortran Compiler.

This sets the required environment variables for compilers, the Intel MPI Library, and Intel Trace Analyzer and Collector, and you are ready to trace your applications.

For more information, see: [Intel oneAPI HPC Toolkit System Requirements](#).

Understand the Workflow

1. [Trace Your Application](#)
2. [Analyze the most active MPI functions](#)
3. [Identify problematic interactions](#)
4. [Improve your application performance by replacing the problem-causing function](#)

Trace Your MPI Application

Generate a trace file to collect event logs for the following application behavior analysis.

1. Set up the environment for launching the Intel® Trace Analyzer and Collector by running the `setvars` script from the oneAPI installation directory.

NOTE

By default, Intel Trace Analyzer and Collector is installed to `/opt/intel/oneapi/itac` for Linux* OS and to `Program Files (x86)\Intel\oneAPI\itac\latest` for Windows* OS.

On Linux:

```
$ source /opt/intel/oneapi/setvars.sh
```

On Windows:

```
> "C:\Program Files (x86)\Intel\oneAPI\setvars.bat"
```

2. Run your MPI application and generate a trace with the `-trace` option.

On Linux:

```
$ mpirun -trace -n 4 ./poisson_sendrecv.single
```

On Windows:

Compile the app and collect the trace.

For Intel oneAPI DPC++/C++ Compiler, run:

```
> mpiicc -trace poisson_sendrecv.single.c
```

For Intel Fortran Compiler, run:

```
> mpiifort -trace poisson_sendrecv.single.f
```

This example generates a trace (`.stf`) for a sample `poisson_sendrecv.single` MPI application running on 4 nodes.

3. Open the generated `.stf` file with Intel Trace Analyzer with Intel Trace Analyzer and Collector.

On Linux:

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

On Windows:


```
> traceanalyzer poisson_sendrecv.single.stf
```

NOTE

For testing purposes, you can download a pre-collected trace file [poisson_sendrecv.single.stf](#) for the poisson used in this document and open it with Intel Trace Analyzer and Collector.

The `.stf` file opens in the Summary Page view, which represents general information about your application performance:

Output	Description						
<div><div>Ratio</div><p>This section represents a ratio of all MPI calls to the rest of your code in the application.</p><table><tr><td>Serial Code - 1.86 sec</td><td>78 %</td></tr><tr><td>OpenMP - 0 sec</td><td>0 %</td></tr><tr><td>MPI calls - 0.521 sec</td><td>21.9 %</td></tr></table></div>	Serial Code - 1.86 sec	78 %	OpenMP - 0 sec	0 %	MPI calls - 0.521 sec	21.9 %	<p>This section represents a ratio of all MPI calls to the rest of your code in the application.</p>
Serial Code - 1.86 sec	78 %						
OpenMP - 0 sec	0 %						
MPI calls - 0.521 sec	21.9 %						

Output	Description
<p>Top MPI functions</p> <p>This section lists the most active MPI functions from all MPI calls in the application.</p> 	<p>This section lists the most active MPI functions from all MPI calls in the application.</p>

NOTE For more information about Intel Trace Analyzer and Collector functionality, see [Learn More](#).

Analyze the Most Active MPI Functions

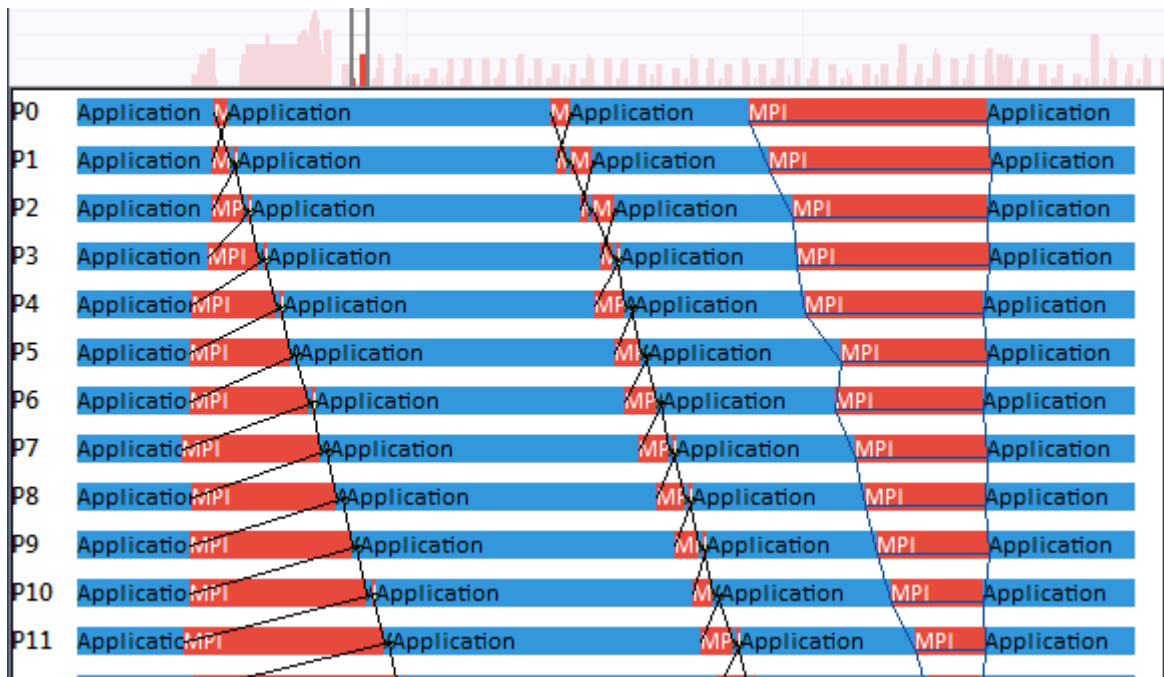
Analyze an MPI application behavior, find bottlenecks and identify serialization to find the ways to improve the application performance.

1. From the **Summary Page** open the Event Timeline view by clicking **Continue > Charts > Event Timeline** for deep analysis of the top MPI functions.

The chart displays individual process activities over time.

Application work is iterative, where each iteration consists of a computational part and MPI communications.

2. Identify a single iteration to focus on and zoom into it by dragging your mouse over the required time interval:

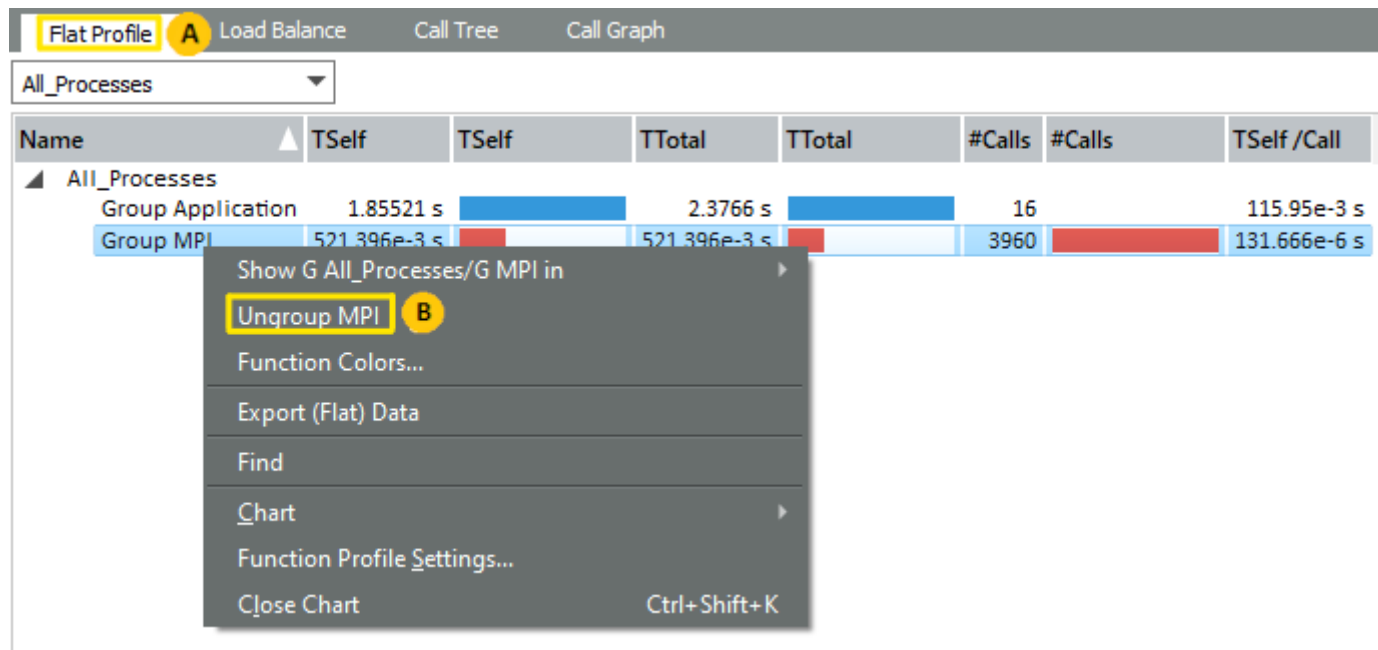


The trace view shows the section within the trace that you selected. The Event Timeline chart shows the events that were active during the selected iteration.

- Horizontal bars represent the processes with the functions called in these processes.
- 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.

3.

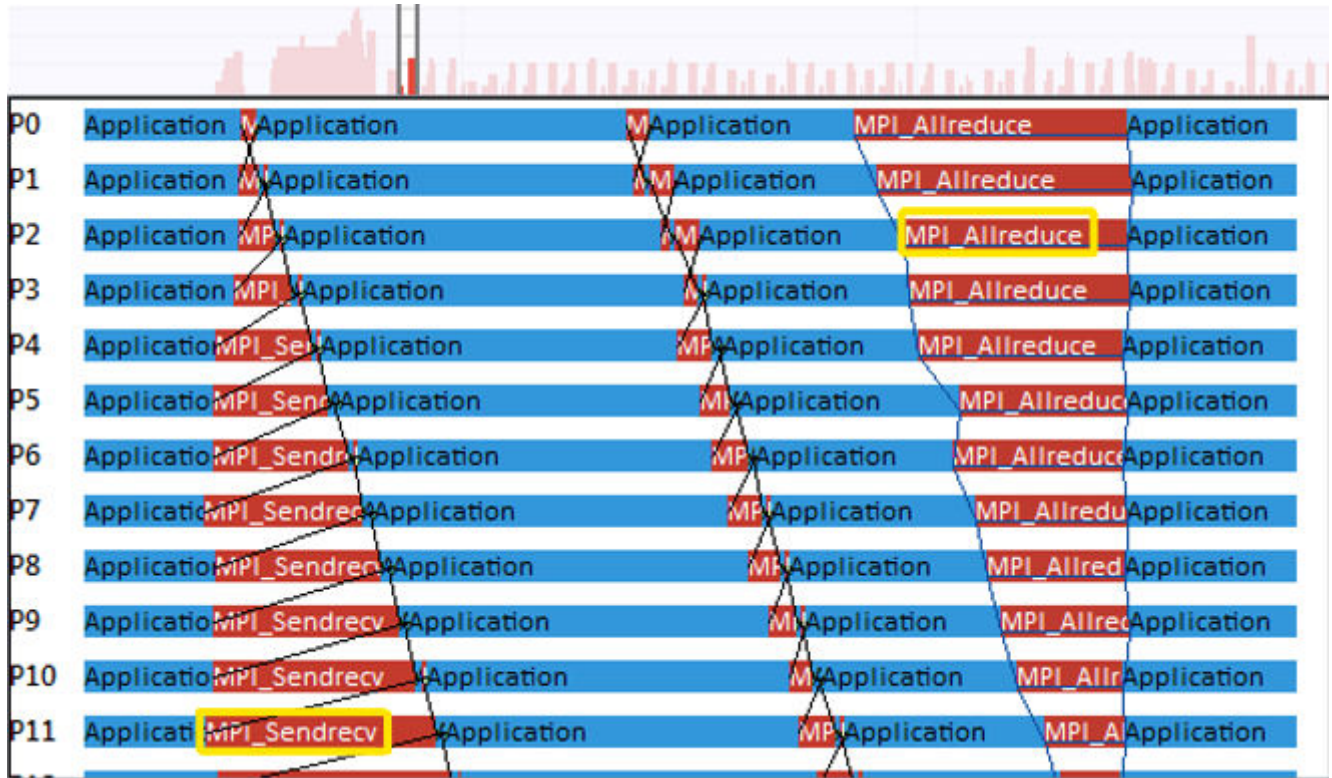
Switch to the **Flat Profile** tab (**A**) to have a closer look at functions executing in the time point you selected in the Event Timeline.



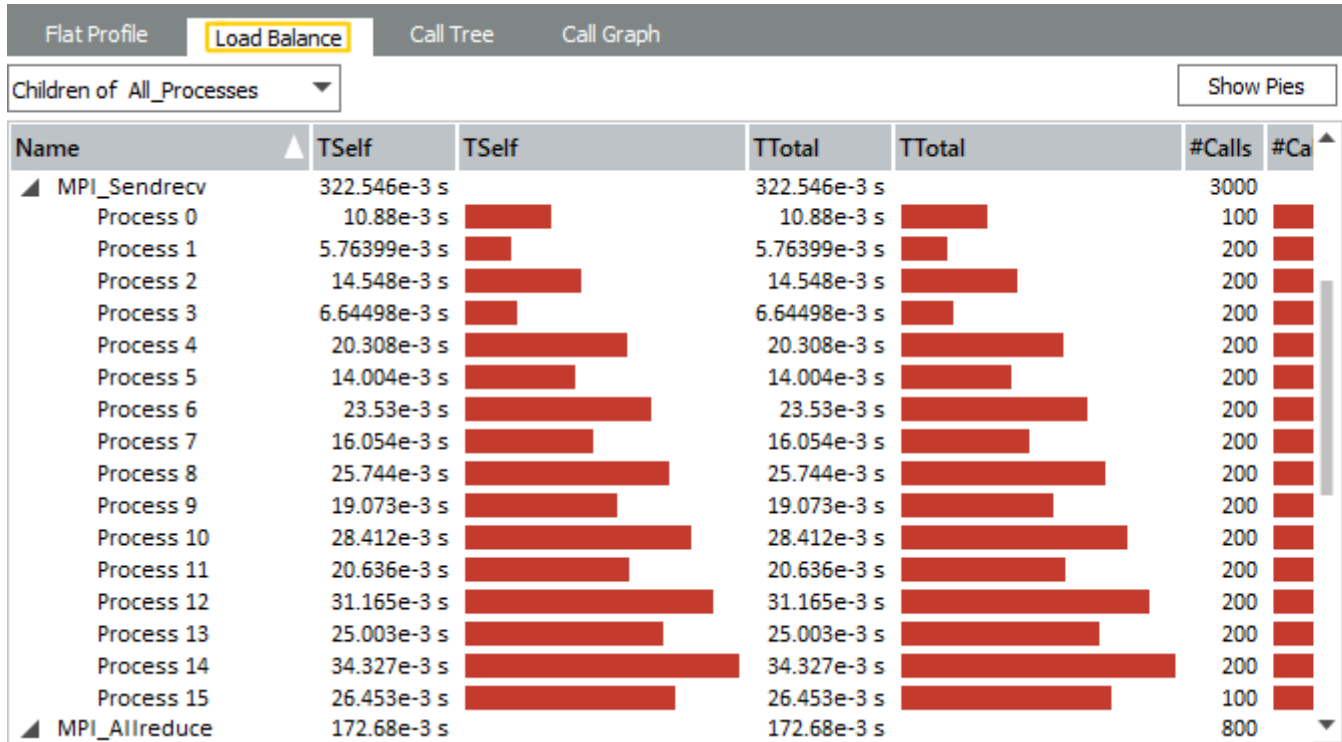
4. Ungroup MPI functions to analyze MPI process activity in your application.

To do this, right-click the **All Processes > Group MPI** (**B**) in the Flat Profile and choose **Ungroup MPI**. This operation exposes the individual MPI calls.

5. Analyze the processes communicating with their direct neighbors using `MPI_Sendrecv` at the start of the iteration. For example:



- a. In the sample, the `MPI_Sendrecv` data exchange has a bottleneck: the process does not exchange data with its next neighbor until the exchange with the previous one is complete. The Event Timelines view displays this bottleneck as a staircase.
 - b. The `MPI_Allreduce` at the end of the iteration resynchronizes all processes; that is why this block has the reverse staircase appearance.
6. Identify serialization, using the Function Profile and Message Profile views.
- a. Open the charts at the same time:
 - In the Function Profile chart, open the **Load Balance** tab.
 - Go to the **Charts** menu to open a Message Profile.
 - b. 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.
 - c. Examine the Message Profile Chart down to the lower right corner.
- The color coding of the blocks indicates that messages traveling from a higher rank to a lower rank need proportionally more time while the messages traveling from a lower rank to a higher rank reveal a weak even-odd kind of pattern:

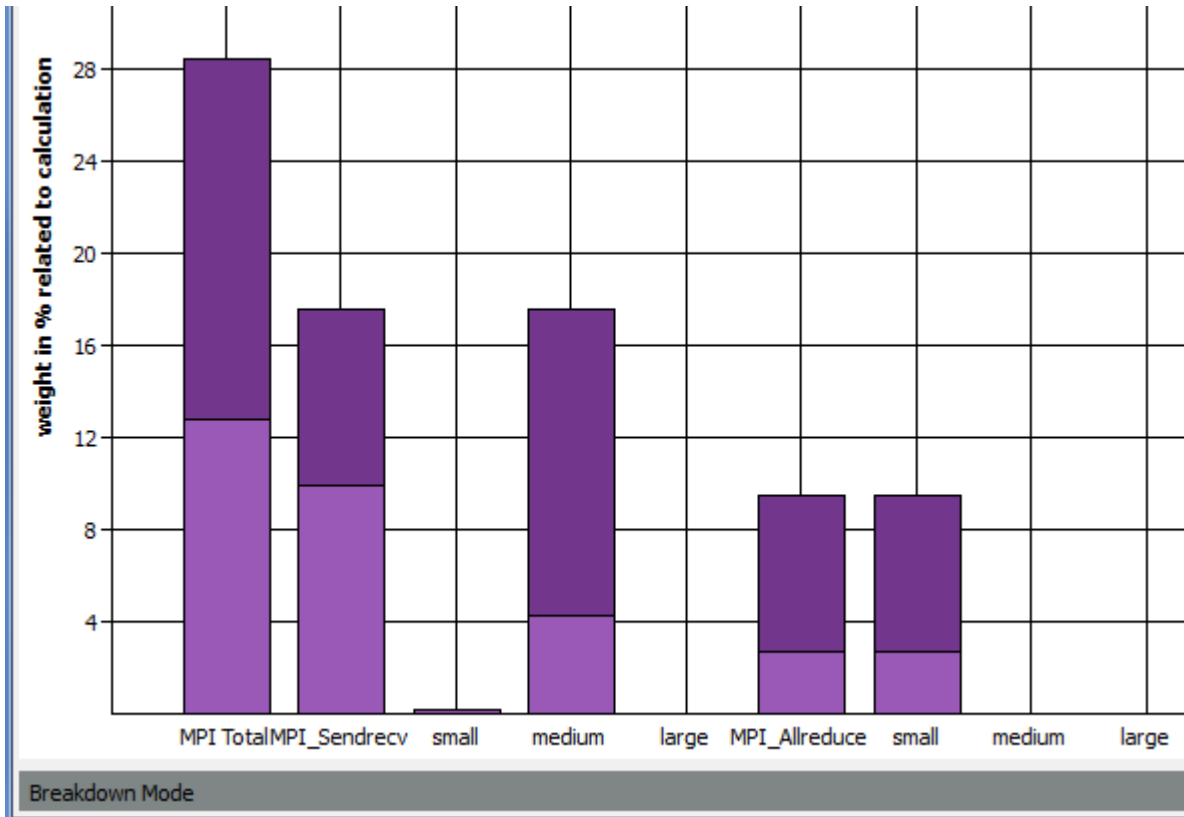


The results of the comparative analysis shows that there are no complex exchange patterns in the application, the exchange is carried out only with neighboring processes. The information will be essential for [Improve Your Application Performance by Changing Communications](#) step to optimize the communication model of the application.

Identify Disbalanced Communications

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

1. Create an idealized file:
 - a. Select **Advanced > Idealization** or click the (**Idealization**) toolbar button.
 - b. Check the idealization parameters in the Idealization dialog box (ideal trace file name and time range for conversion).
 - c. Click **Start** to idealize your trace.
2. Compare the original trace with the idealized trace:
 - a. Select **Advanced > Imbalance Diagram** or click the (**Imbalance Diagram**) toolbar button.
 - b. In the Imbalance Diagram dialog box, click the **Open Another File** button, navigate to the idealized trace, and select it.
 - c. 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 light color and comprises about 10% for the `MPI_Sendrecv` function. This is the time the processes spend waiting for each other.

Improve Your Application Performance by Changing Communications

Improve the performance of the MPI application by changing blocking to non-blocking communications.

1. In your code replace the serial `MPI_Sendrecv` with non-blocking communication: `MPI_Isend` and `MPI_Irecv`. For example:

Original code snippet:

```
// boundary exchange
void exchange(para* p, grid* gr)
{
    int i,j;
    MPI_Status status_100, status_200, status_300, status_400;
    // send down first row
    MPI_Send(gr->x_new[1], gr->lcol+2, MPI_DOUBLE, gr->down, 100, MPI_COMM_WORLD);
    MPI_Recv(gr->x_new[gr->lrow+1], gr->lcol+2, MPI_DOUBLE, gr->up, 100, MPI_COMM_WORLD,
    &status_100);
    // send up last row
    MPI_Send(gr->x_new[gr->lrow], gr->lcol+2, MPI_DOUBLE, gr->up, 200, MPI_COMM_WORLD);
    MPI_Recv(gr->x_new[0], gr->lcol+2, MPI_DOUBLE, gr->down, 200, MPI_COMM_WORLD, &status_200);
    // copy left column to tmp arrays
    if(gr->left != MPI_PROC_NULL)
    {
```

```

for(i=0; i< gr->lrow+2; i++)
{
    left_col[i] = gr->x_new[i][1];
}
MPI_Send(left_col, gr->lrow+2, MPI_DOUBLE, gr->left, 300, MPI_COMM_WORLD);
}
if(gr->right != MPI_PROC_NULL)
{
    MPI_Recv(right_col, gr->lrow+2, MPI_DOUBLE, gr->right, 300, MPI_COMM_WORLD, &status_300);
    // copy right column to tmp
    // copy received left column to ghost cells
    for(i=0; i< gr->lrow+2; i++)
    {
        gr->x_new[i][gr->lcol+1] = right_col[i];
        right_col[i] = gr->x_new[i][gr->lcol];
    }
    // send right
    MPI_Send(right_col, gr->lrow+2, MPI_DOUBLE, gr->right, 400, MPI_COMM_WORLD);
}
if(gr->left != MPI_PROC_NULL)
{
    MPI_Recv(left_col, gr->lrow+2, MPI_DOUBLE, gr->left, 400, MPI_COMM_WORLD, &status_400);
    for(i=0; i< gr->lrow+2; i++)
    {
        gr->x_new[i][0] = left_col[i];
    }
}
}
}

```

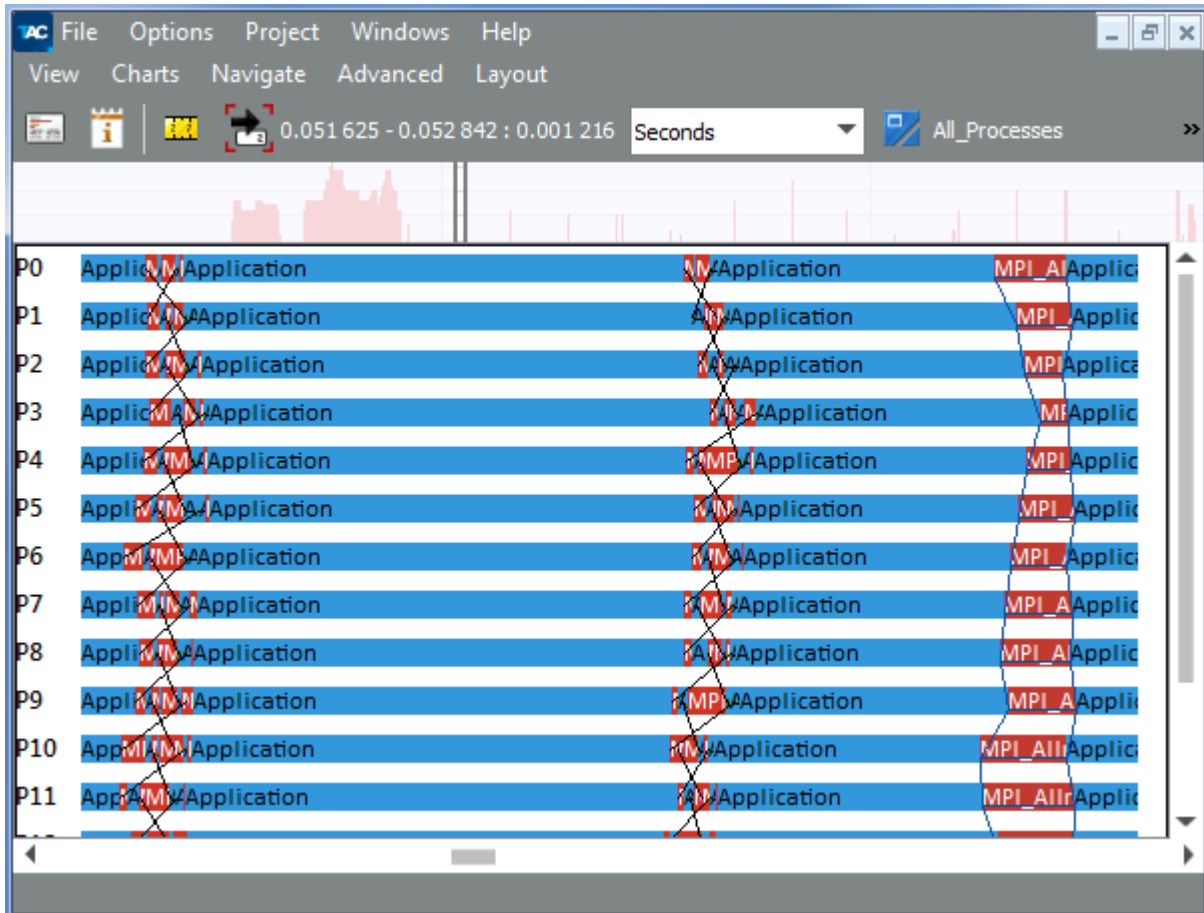
Updated code snippet:

```

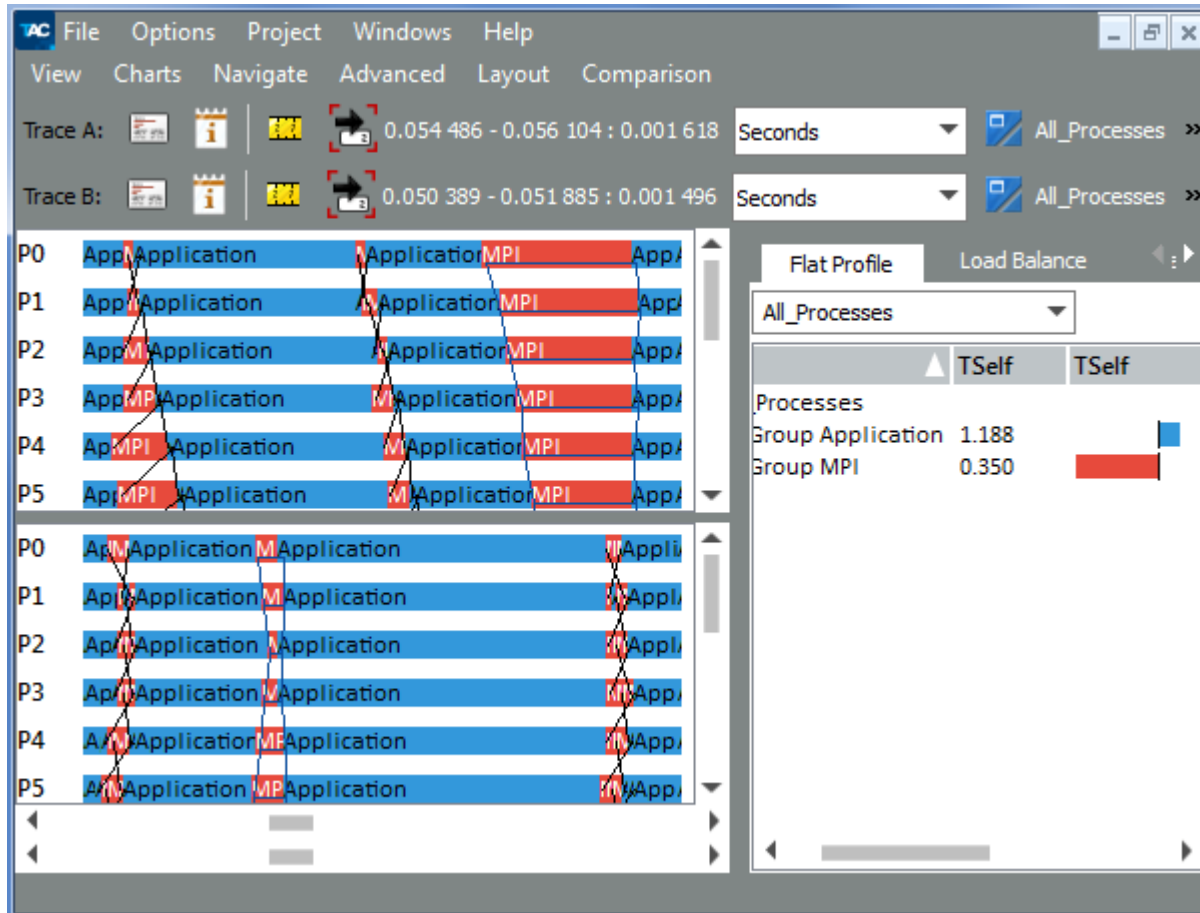
MPI_Request req[7];
// send down first row
MPI_Isend(gr->x_new[1], gr->lcol+2, MPI_DOUBLE, gr->down, 100, MPI_COMM_WORLD, &req[0]);
MPI_Irecv(gr->x_new[gr->lrow+1], gr->lcol+2, MPI_DOUBLE, gr->up, 100, MPI_COMM_WORLD, &req[1]);
.....
MPI_Waitall(7, req, MPI_STATUSES_IGNORE);

```

Once corrected, the single iteration of the revised application will look like the following example:



2. Use the Intel Trace Analyzer Comparison view to compare the serialized application with the revised one. Compare two traces with the help of the **Comparison View**, going to **View > Compare**. 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.

NOTE For more information about node-level performance of your application, see documentation for the respective tools: [Intel® VTune™ Profiler MPI Code Analysis](#) and [Analyzing Intel® MPI applications using Intel® Advisor](#).

Learn More

Explore the following resources for more information about Intel® Trace Analyzer and Collector.

Document	Description
Intel Trace Analyzer User and Reference Guide	Trace Analyzer and Collector User and Reference Guide provides information how to generate trace files, usage instructions on Intel Trace Analyzer, and description of all its GUI elements.
Tutorial: Analyzing an OpenMP* and MPI Application	This tutorial guides you through basic steps required to analyze hybrid OpenMP* and MPI code for inefficiencies using Intel® VTune™ Profiler's Application Performance Snapshot, Intel Trace Analyzer and Collector, and Intel VTune Profiler.

Document	Description
Tutorial: Detecting and Resolving Errors with MPI Correctness Checker	This tutorial demonstrates the correctness checking workflow applied to sample MPI applications with deadlock and data type mismatch errors.
Tutorial: Reducing Trace File Size	This tutorial demonstrates a workflow applied to the poisson application and shows the options to reduce the trace file size.
Release Notes	The Release Notes document contains the most up-to-date information about the product, including system requirements, product description, technical support, and known limitations and issues.

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

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

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

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.