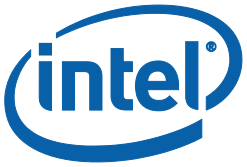

Frequently Asked Questions for Intel® Trace Analyzer and Collector

Document Number: 318118-004

Contents

Legal Information

1. General Questions about Intel® Trace Analyzer and Collector

1.1. What are some key things I can learn about my program using Intel® Trace Analyzer and Collector?

The Intel® Trace Analyzer and Collector is a graphical tool used primarily for MPI-based programs. It helps you understand your application's behavior across its full runtime. It can help find temporal dependencies in your code and communication bottlenecks across the MPI ranks. It also checks the correctness of your application and points you to potential programming errors, buffer overlaps, and deadlocks.

1.2. Will Intel® Trace Analyzer and Collector only work with Intel® MPI Library?

No, the Intel® Trace Analyzer and Collector support all major MPICH2-based implementations. If you're wondering whether your MPI library can be profiled using the Intel Trace Analyzer and Collector, you can run a simple ABI-compatibility check by compiling the provided mpiconstants.c file and verifying the values with the ones provided in the *Intel® Trace Collector Reference Manual*.

1.3. Can Intel® Trace Analyzer and Collector be used on applications for Intel® Many Integrated Core Architecture (Intel® MIC Architecture)?

Yes, Intel® MIC Architecture is fully supported by the Intel® Trace Analyzer and Collector.

1.4. What file and directory permissions are required to use Intel® Trace Analyzer and Collector?

You do not need to install special drivers, kernels, or extra permissions. Simply install the Intel® Trace Analyzer and Collector in the `$HOME` directory and link it with your application of choice from there.

1.5. Can I import or export trace data to/from Intel® Trace Analyzer and Collector?

Yes, you can export the data from any of the Profile charts (**Function Profile**, **Message Profile**, and **Collective Operations Profile**) as part of the Intel® Trace Analyzer interface. To do this, open one of these profiles in the GUI, right-click to bring up the Context Menu, and select the **Export Data** option. The data will be saved in simple text format for easy reading.

1.6. What size MPI application can I analyze with Intel® Trace Analyzer and Collector?

It depends on how large or complex your application is, how many MPI calls you are making, and for how long you are running. There are no internal limitations on the size of the MPI job but there are plenty of external ones. It all depends on how much memory is available on the system (per core) both for the application, the MPI library, and for the Intel® Trace Collector processes, as well as disk space availability. Any additional flags enabled (for example, storing call stack and source code locations) cause an increase in the size of the trace file. Filtering out unimportant information is always a good solution to reducing trace files. For more information, refer to the *Reducing Trace File Size Tutorial*.

1.7. Can I use Intel® Trace Analyzer and Collector with Intel® VTune™ Amplifier XE, Intel® Inspector XE, or other analysis tools?

While these tools would collect information separate from each other, in their own format, it's easy enough to use the Intel® VTune™ Amplifier XE and Intel® Inspector XE tools under an MPI environment. Check each tool's respective User's Guide for more info on Viewing Collected MPI Data.

You can use tools such as Intel VTune Amplifier XE and Intel Inspector XE for node-level analysis, and use the Intel Trace Analyzer and Collector for cluster-level analysis.

2. General Questions about Intel® Trace Collector

2.1. Should I recompile/relink my application to collect information?

It depends on your application.

- For Windows* OS, you have to relink your application by using the `-trace` link-time flag.

- For Linux* OS (and if your application is dynamically linked), you do not need to relink or recompile. Simply use the `-trace` option at runtime (for example: `mpirun -trace`).

2.2. What file format is the trace data collected in?

Intel® Trace Collector stores all collected data in Structured Tracefile Format (STF) which allows for better scalability across both time and processes. For more details, refer to the "Structured Tracefile Format" section of the *Intel® Trace Collector Reference Manual.*

2.3. How do I control which part of my application should be profiled?

The Intel® Trace Collector provides several options to control the data collection. By default, only information about MPI calls is collected. If you'd like to filter which MPI calls should be traced, create a configuration file and set the `VT_CONFIG` environment variable.

Set up time frames within which Intel® Trace Collector will save the events into the trace file by specifying the `TIME-WINDOWS` option in the configuration file.

If you'd like to expand the information collected beyond MPI and include all user-level routines, recompile your application with the `-tcollect` switch available as part of the Intel® Compilers. In this case, Intel Trace Collector will gather information about all routines in the application, not just MPI. You can similarly filter this via the `-tcollect-filter` compiler option. Note that since all user functions are instrumented and profiled, the trace file may become huge. Use this capability with short or small runs only.

If you'd like to be explicit about which parts of the code to be profiled, use the Intel Trace Collector API calls. You can manually turn tracing on and off via a quick API call. You can also turn trace collection on and off by inserting the `MPI_Pcontrol` function into the source code.

For more Information on all of these methods, refer to the *Intel® Trace Collector Reference Manual.*

2.4. How can I control the amount of data collected to a reasonable amount? What is a reasonable amount?

Each application is different in terms of the profiling data it can provide. The longer an application runs, and the more MPI calls it makes, the larger the STF files will be. You can filter some of the unnecessary information out by applying appropriate filters (see Question 2.3 for more details, check out some tips on Intel® Trace Collector Filtering or refer to the tutorial on *Reducing Trace File Size*). Additionally, you can be restricted by the resources allocated to your account; consult your cluster administration about quotas and recommendations.

2.5. How can I limit the memory consumption of Intel® Trace Collector?

During the application run, Intel® Trace Collector first stores trace data in memory buffers. There are two options that control the allocation of these buffers: `MEM BLOCKSIZE` specifies the size of each memory block in bytes, and `MEMMAXBLOCKS` determines the maximum number of memory blocks. Intel Trace Collector will not exceed the memory limits set by `MEMMAXBLOCKS` multiplied by `MEM BLOCKSIZE`. When this trace data memory is exhausted, one of three actions is taken:

- If the `AUTOFLUSH` option is enabled (default), the collected trace data is flushed to disk, and the trace collection continues. The spill files are automatically merged when the application finalizes, so that all records will appear in the trace file.
- If `AUTOFLUSH` is disabled and `MEM OVERWRITE` is enabled, the trace buffers will be overwritten from the beginning, in effect recording the last n records.
- Otherwise, the trace collection stops, in effect collecting the first `n` records.

Placing trace data in main memory can slow down the application if it needs the memory itself. Setting `MEM MAXBLOCKS` puts a hard limit on the amount of memory used by Intel Trace Collector, but can disrupt the application when a process waits for flushing of trace data. To avoid this, Intel Trace Collector can be told to start flushing earlier in the background with the `MEM FLUSHBLOCKS` option. This option is only available in more recent thread-safe versions of Intel Trace Collector.

To understand how much memory is currently in use, Intel Trace Collector can add counter data to the trace. If enabled, each process will store its own values for these counters in the trace each time they change. This makes it possible to take the effect of buffer handling into account when doing the analysis of the trace. These counters are not enabled by default. It is necessary to add the following lines to a configuration file (see usage of `VT_CONFIG`) to enable each counter:

```
COUNTER data_in_ram ON COUNTER data_in_file ON COUNTER flush_active ON
```

At runtime, Intel Trace Collector also provides feedback on the amount of data collected: with the default setting of 500MB for the `MEM INFO` configuration option a message is printed each time more than this amount of new data is recorded by a process. The value is chosen so that the message serves as a warning when the amount of trace data exceeds the amount that can usually be handled without problems. In order to use it as a kind of progress report a much lower value would be more appropriate.

Use the `KEEP_RAW_EVENTS` setting to disable the `MERGE` phase and reduce memory/CPU consumption at the `Finalize` phase.

2.6. How can I manage the Intel® Trace Collector API calls?

The API routines greatly extend the functionality of Intel® Trace Collector. Unfortunately, manual instrumenting the application source code with the Intel Trace Collector API makes code maintenance harder. An application that contains calls to the Intel Trace Collector API requires the Intel Trace Collector library to link and incurs a certain profiling overhead. The dummy API library `libVTnull.a` helps in this situation: all the API calls map to empty subroutines, and no trace data is ever gathered if an application is linked to it. Still, the extraneous function calls remain and may cause a slight overhead.

It is recommended that the C pre-processor (or an equivalent tool for Fortran) is used to guard all the calls to the Intel Trace Collector API by `#ifdef` directives. This will allow easy generation of a plain vanilla version and an instrumented version of a program.

2.7. What happens when a program fails?

The Intel® Trace Collector library stores trace data first in buffers in the application memory, and then in flush files (one per MPI process) when the buffers have been filled. In normal operation, the library will merge the trace data from each process during execution of the `MPI_Finalize()` routine, and write the trace data into a single trace file suitable for input to Intel® Trace Analyzer. If a program fails, `MPI_Finalize()` is never executed, and the traditional Intel Trace Collector does not write a trace file. To get a trace file until the program crash, you can link against the fail-safe version `libVTfs`.

Troubleshooting:

The Intel Trace Collector library can report four basic error classes:

1. Setup errors
2. Invalid configuration file format
3. Erroneous use of the API routines
4. Insufficient memory

The first category includes invalid settings of the `VT_ environment` variables, failure to open the specified trace file and so on. A warning message is printed; the library ignores the erroneous setup and tries to continue with default settings.

For the second class, a warning message is printed, the faulty configuration file line is ignored, and the parser continues with the next line.

When an Intel Trace Collector API routine is called with invalid parameters, a negative value is returned (as a function result in C, in the error parameter in Fortran), and operation continues. Invoking any API routines before `MPI_Init()` or after `MPI_Finalize()` is considered erroneous, and the call is silently ignored.

An insufficient memory error can occur during execution of an API routine or within any MPI routine if tracing is enabled. In the first case, an error code (`VT_ENOMEM` or `VTENOMEM`) is returned to the calling process; in any case, Intel Trace Collector prints an error message and attempts to continue by disabling the collection of trace data. Within `MPI_Finalize()`, the library will try to generate a trace file from the data gathered before the insufficient memory error.

Although Intel Trace Collector tries to handle out-of-memory situations gracefully, library calls in the application might not be as tolerant, or the operating system does not handle such a situation well enough. To avoid a memory error in the first place, try to limit the amount of trace data as explained in the section Limiting Memory Consumption. The memory requirements of Intel Trace Collector can be reduced with the `MEM BLOCKSIZE` and `MEM MAXBLOCKS` configuration options. The `AUTOFLUSH` option needs to remain enabled if you want to see a trace of the whole application run.

2.8. I cannot find the trace file. Where is it?

Unless told otherwise in the configuration file, Intel® Trace Collector will write the trace data to the file `argv[0].stf`, with `argv[0]` being the application name in the command line (same as `getarg(0)` in Fortran). Note that your MPI library or the MPI execution script may interfere with `argv[0]`, and that only the process actually writing the trace file (usually the one with rank `0` in `MPI_COMM_WORLD`) will look at it. A relative pathname will be interpreted relative to that process current working directory.

You can however change the trace file name with the `LOGFILE NAME` directive in a configuration file.

If it turns out that Intel Trace Collector cannot create the specified trace file, it will attempt to write to the file `/tmp/VT <PID>.stf`, with `<PID>` being the Unix* process id of the trace file-writing MPI process.

In any case, an information message with the actual trace file name will be printed by Intel Trace Collector within `MPI_Finalize()`.

On systems where not all processes see the same files (especially in environment where Intel® Xeon Phi™ Co-processor cards are present), be sure to look for the trace file in the correct process file system. You can influence which process will write the file by setting an environment variable or by a directive in the configuration file.

2.9. What is this "Bad Clock Resolution" all about?

If the clock resolution is very low (the timer function returns the same value for a long period of time), then many events will be recorded on the same time stamp and analysis of such a trace becomes very hard. In particular the Global Timeline becomes useless.

If Intel® Trace Collector detects this problem, it issues a warning like:

```
minimum clock increment 1e-3s is very high, please fix system setup to obtain
better traces
```

The minimum clock increment is always stored in the trace file info, because the timer base also listed there may be lower than the real value.

2.10. I get "Error:libVT.a: file not recognized". What's wrong?

Make sure that the libraries are installed using the `./install` scripts provided with the tool. Otherwise, the libraries will be locked and will be in a format not recognized as valid, thus showing the error above.

3. General Questions about Intel® Trace Analyzer

3.1. How can I analyze the collected information?

Once you have collected the trace data, you can analyze it via the Graphical Interface called the Intel® Trace Analyzer. Simply call the command (`$ traceanalyzer`) or double-click on the Intel Trace Analyzer icon and navigate to your STF files via the File Menu.

You can get started by opening up the **Event Timeline** chart (under the **Charts** Menu) and zooming in at an appropriate level.

Check out the *Detecting and Removing Unnecessary Serializatio Tutorial* on ideas how to get started. For details on all Intel Trace Analyzer functionality, refer to the *Intel® Trace Analyzer Reference Manual*.

3.2. What are Views in the Intel® Trace Analyzer?

A View holds a collection of Charts in a single window. Those Charts, inherent in the same View, use the same perspective on the data. This perspective is made up of the following attributes: the time interval, process aggregation, function aggregation and filters. It helps to easily analyze a trace file by looking at multiple partitions of the data from various points of view.

Whenever an attribute in the current perspective is changed for one of the Charts, all other Charts change, too. You can use several Views at the same time, as it is a very flexible and variable mechanism for exploring, analyzing and comparing trace data.

3.3. What are Charts in the Intel® Trace Analyzer?

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

The Charts supported by Intel Trace Analyzer are divided into:

- Timelines: the **Event Timeline**, the **Qualitative Timeline**, the **Quantitative Timeline** and the **Counter Timeline**.

- Profiles: the **Function Profile**, the **Message Profile**, and the **Collective Operations Profile**.

While the Timelines show trace data in graphical form over a horizontal axis representing runtime, the Profiles show statistical data. You can find all these Charts under the **Charts** menu item. When you open a new file in the Intel Trace Analyzer, the default display is a View that contains the **Function Profile** Chart for the opened file.

3.4. What is aggregation? How many types of it are available in the Intel® Trace Analyzer?

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

Process Aggregation focuses on the processes that are important and aggregates the results into Process Groups. To construct and/or choose the appropriate process groups, use the **Process Aggregation** dialog box (go to **Advanced > Process Aggregation**).

Function Aggregation focuses on a subset of functions and aggregates them into Function Groups, without causing a distraction for other functions that are currently not significant. Use the **Function Aggregation** dialog box (go to **Advanced > Function Aggregation**) to do this.

3.5. My message profile shows ridiculously great numbers. What went wrong?

Most probably the message was recorded in reverse order meaning that the message was received before it was sent. This can happen on systems with coarse clock resolution and/or very fast communication hardware. If the clock resolution cannot be increased, there is currently no solution for it.

3.6. Why do user-defined activities not work?

In order to minimize the instrumentation overhead, Intel® Trace Collector does not check for global consistency of the activity codes specified by calls to `VT_symdef()` or `VTSYMDEF()`. When using user defined activities you should ensure that:

- The same code is used for the same activity in all processes.
- Two different symbols never share the same code.

If these rules are violated, Intel® Trace Analyzer might complain about duplicate activities, or activities may be mislabeled in the Intel Trace Analyzer displays.

3.7. Why are some messages not shown?

In order for messages to be indicated in the Intel® Trace Analyzer displays, both the calls to the sending and the receiving MPI routines must be traced. For non-blocking receives, the call to the MPI wait or test routine that did complete the receive request must be logged.

If tracing has been disabled during runtime it can happen that for some messages, either the sending or the receiving call has not been traced. As a consequence, these messages are not shown by Intel Trace Analyzer, and other messages can appear to be sent to or received at the wrong place. Similarly, filtering out some of the above mentioned MPI routines has the same effect.

3.8. I have a large application and my Charts are a bit crowded. Is there some way to filter particular functions, messages, and collective operations in Intel® Trace Analyzer?

Use the filtering and tagging capability of Intel® Trace Analyzer to look for specific functions, messages and/or collective operation.

To access the **Filtering** dialog box, go to **Advanced > Filtering**. This dialog box enables you to specify filter expressions that describe which function events, messages, and collective operations are to be analyzed and shown. You can generate the filter expression through a graphical interface or type it in manually. If the current expression cannot be converted into a proper filter definition, then the dialog shows a red warning that indicates the reason. After applying a filtering expression, the Charts will display only events that satisfy the condition of the expression. All other events will be suppressed.

Tagging works the same way as filtering and uses the same grammar to create the tagging expression. The only difference is that tagging highlights events that satisfy the specific user-defined conditions. That emphasis varies depending on the Chart to which the tagging expression is applied. For specific details, refer to the *Intel® Trace Analyzer Reference Manual*.