Analyzing MPI programs with Intel® VTune™ Amplifier XE and Intel® Inspector XE tools

May 9, 2011

Abstract

This document is a supplement to the Intel® Cluster Studio XE and Intel® Parallel Studio XE documentation. It describes how to use the Intel® VTune™ Amplifier XE and Intel® Inspector XE tools for analyzing MPI workloads.
Introduction

Intel Cluster Studio XE 2012 helps exploit scalable parallelism of a modern cluster at all levels of hybrid parallel or sequential computing for a Fortran/C/C++ MPI program: message passing, threading and SIMD / data levels. The Intel® MPI library is used at the process messaging level. The Intel OpenMP® library, Intel® Threading Building Blocks (Intel® TBB), and Intel® Cilk™ plus extensions can be used for thread parallelism. The Intel® Math Kernel Library (Intel® MKL) can be used to automatically exploit threading, message passing through ScaLAPACK, and SIMD data parallelism capabilities of Intel® hardware. This paper describes usage models and capabilities for doing performance and correctness analysis of MPI programs using the VTune Amplifier XE and Inspector XE tools available in Cluster Studio XE 2012.

Note that the version of the Intel MPI library included with the Intel Cluster Studio XE 2012 makes an important switch to use the Hydra process manager by default for mpirun which provides high scalability across the big number of nodes. The implications relevant to the scope of this document are described below. The Beta release uses the MPD manager as the default.

Installation

Intel Cluster Studio XE 2012 supports both Microsoft Windows® and Linux® OS cluster environments. The Intel Cluster Studio ships both command line and GUI interfaces for the tools. The installation of the Intel Cluster Studio on a cluster can be done across the nodes automatically:

- The Windows and Linux OS installers detect whether the machine is an MPI master and, if it is, they suggest automatic installation on all cluster nodes.
- On Windows OS, a special system API is used to detect whether the machine is an MPI master node and to launch slave silent installations.
- On Linux OS, the user provides the installation with the MPI configuration file used to configure the cluster -- that is used to get the list of slave MPI machines. Then, the installer uses SSH to launch slave installations on the nodes (using key-based authentication usually).
- An alternative installation technique on Linux clusters is to use RPM files that are shipped in the product with a standard distributed install tool. In this case, when installing the VTune Amplifier XE the system administrator will also need to manually invoke SEP driver building and installation on each node in the cluster.

On Linux OS, the tools are installed by default at /opt/intel/vtune_amplifier_xe_2011 and /opt/intel/inspector_xe_2011. Similarly on Windows OS, VTune Amplifier XE is installed at C:\Program Files (x86)\Intel\VTune Amplifier XE 2011, and Intel Inspector XE is installed at C:\Program Files (x86)\Intel\Inspector XE 2011.

When installing the Intel Cluster Studio XE on a Linux cluster, it is highly recommended to turn on the per-user Hardware Event-based Sampling mode for the VTune Amplifier XE tool to make sure that during the collection only data for the current user is collected. Once enabled by the administrator during the installation, this mode cannot be turned off by a regular user which is intentional to preclude individual users from observing the performance data over the whole node including other users' activities. Please note that the per-user Hardware Event-based Sampling mode is not available on the Windows operating system at the moment.

Once installed, the user can use the respective Linux or Windows OS *-vars.{csh|sh|bat} files to setup the
appropriate environment (PATH, MANPATH) in the current terminal session. The examples below focus on a Linux OS cluster installation since that is more common. The tools capabilities are the same for Windows OS unless explicitly noted and described.

**Usage Model Concepts**

To analyze the performance and correctness of an MPI program at the inter-process level, use the Intel® Trace Analyzer and Collector tool (located at `<installdir>/itac` directory after installation). The Intel Trace Analyzer and Collector attaches to the program through linkage (statically, dynamically, also through `LD_PRELOAD` or via the Intel® Compiler `-tcollect` and `-tcollect-filter` options), or by using the “itcpin” tool. The tools collect information about events at the MPI level between processes and allow analyzing the performance and correctness of the MPI calls, deadlock detection, data layout errors, as well as risky or incorrect MPI constructs. The Intel Trace Analyzer and Collector data is correlated and aggregated across all processes and all nodes, that participated in the execution run.

Beyond the inter-process level of MPI parallelism, the processes that make up the programs on a modern cluster often also use fork-join threading through OpenMP* and Intel® TBB. This is where the VTune Amplifier XE performance analyzer and the Intel Inspector XE checker should respectively be used to analyze the performance and correctness of an MPI program. The rest of the paper is about the usage model of these tools for analyzing an MPI program.

At the high level the analysis usage model consists of 3 steps:

1. Use the `amplxe-cl` and `inspxe-cl` command line tools to collect data about a program. By default, all processes are analyzed, but it is possible (and sometimes required for VTune Amplifier XE – there are certain collection technology limitations discussed below) to filter the data collection to limit it to a subset of processes. An individual result directory is created for each spawned MPI program process that was analyzed with MPI process rank value captured.

2. Post-process the result, which is also called “finalization” or “symbol resolution”. This is done automatically for each result directory once the collection has finished.

3. Open the content of each result directory through the GUI standalone viewer to analyze the data for the specific process. The GUI viewers are independent: each product has its own user interface (Trace Analyzer and Collector, VTune Amplifier XE, and Inspector XE).

The following three sections describe these steps in details. Please note that only homogeneous clusters are currently supported:

- The file system contents should be the same on all nodes to make sure that the modules referenced in the collected data are available automatically on the host where the collection was initiated. This limitation can be overcome by manual copying of the modules for analysis from the nodes and adjusting the VTune Amplifier XE / Inspector XE project search directories to make the modules found.
- For the VTune Amplifier XE the CPU model & stepping should be the same on all nodes so that the hardware Event-based sampling operates with the same Performance Monitoring Unit (PMU) type on all nodes.
Collecting Performance/Correctness Data

To collect performance or correctness data for an MPI program with the VTune Amplifier XE / Inspector XE on a Windows or Linux OS, the following command should be used:

```
$ mpirun -n <N> <abbr>-cl -r my_result -collect <analysis type> my_app [my_app_ options]
```

where `<abbr>` is “amplxe” or “inspxe” respectively. The list of analysis types available can be viewed using “amplxe-cl –help collect” command. The most reasonable analysis type to start with for the VTune Amplifier XE is “hotspots”, so an example of full command line for collection would be:

```
$ mpirun -n 4 amplxe-cl -r my_result -collect hotspots -- my_app [my_app_ options]
```

A similar command line for the Inspector XE and its “ti1”/”mi1” analysis types (the lowest overhead threading and memory correctness analysis types respectively) would look like:

```
$ mpirun -n 4 inspxe-cl -r my_result -collect mi1 -- my_app [my_app_ options]
$ mpirun -n 4 inspxe-cl -r my_result -collect ti1 -- my_app [my_app_ options]
```

Note that in the examples this document uses the `mpirun` command as opposed to `mpiexec` and `mpiexec.hydra` while real-world MPI jobs might use the `mpiexec*` ones. `mpirun` is a higher-level command that dispatches to `mpiexec` or `mpiexec.hydra` depending on the current default and options passed. All the examples listed in the paper work for the `mpiexec*` commands as well as the `mpirun` command.

As a result of using the collection commands above, a number of result directories will be created in the current directory, named as `my_result.0` - `my_result.3`. The numeric suffix is the corresponding MPI process rank that is detected and captured by the collector automatically. The usage of the suffix makes sure that multiple `amplxe-cl / inspxe-cl` instances launched in the same directory on different nodes do not overwrite the data of each other and can work in parallel. So, a separate result directory is created for each analyzed process in the MPI job. The details of how the process rank is captured are described in the section “Support of Non-Intel MPI Implementations” below.

Sometimes it is necessary to collect data for a subset of the MPI processes in the workload. In this case the per-host syntax of `mpirun/mpiexec*` should be used to specify different command lines to execute for different processes. For example, here is an example where there are 16 processes in the job distributed across the hosts and hotspots data should be collected for only two of them:

```
$ mpirun -host myhost -n 14 ./a.out : -host myhost -n 2 amplxe-cl -r foo -c hotspots ./a.out
```

As a result, two directories will be created in the current directory: `foo.14` and `foo.15` (given that process ranks 14 and 15 were assigned to the last 2 processes in the job). As an alternative to specifying the command line above, it is possible to create a configuration file with the following content:

```
# config.txt configuration file
-host myhost -n 14 ./a.out
-host myhost -n 2 amplxe-cl -quiet -collect hotspots -r foo ./a.out
```

and run the data collection as

```
$ mpirun -configfile ./config.txt
```
to achieve the same result as above (foo.14 and foo.15 result directories will be created). Similarly, you
can use specific host names to control where the analyzed processes are executed:

```
# config.txt configuration file
-host myhost1 -n 14 ./a.out
-host myhost2 -n 2 amplxe-cl -quiet -collect hotspots -r foo ./a.out
```

When the host names are mentioned, consecutive MPI ranks are allocated to the specified hosts. In the
case above, ranks 0 to 13, inclusive, will be assigned to myhost1, the remaining ranks 14 and 15 will be
assigned to myhost2. On Linux, it is possible to omit specifying the exact hosts, in which case the
distribution of the processes between the hosts will be done in round-robin fashion. That is, myhost1
will get MPI ranks 0, 2, and 4 thru 15, while myhost2 will get MPI ranks 1 and 3. The latter behavior may
change in the future.

When launching the collection on Windows OS, we recommend passing “-genvall” option to the mpiexec
tool to make sure that the user environment variables are passed to all instances of the profiled process.
Otherwise, by default the processes are launched in the context of a system account and some
environment variables (USERPROFILE, APPDATA) do not point where the tools expect them to point to.

There are also some specialties about stdout / stdin behavior in MPI jobs profiled with the tools:

- It is recommended to pass the -quiet / -q option to amplxe-cl / inspxe-cl to avoid diagnostic
  output like progress messages being spilled to the console by every tool process in the job.
- The user may want to use the “-I” option for mpiexec/mpirun to get stdout lines marked with
  MPI rank.

## Finalizing the Collected Data

The finalization of the data (symbol resolution, conversion to the database) happens automatically after
the collection has finished. It happens on the same compute node where the command line collection
was executing, and so the binaries and symbol files will be located automatically by the amplxe-cl /
inspxe-cl tools. In cases where the search algorithm needs to be adjusted (common reason: need to point
to symbol files stored elsewhere), the -search-dir option should be used with amplxe-cl / inspxe-cl as
follows:

```
$ mpirun -np 128 amplxe-cl -q -collect hotspots -search-dir sym=/home/foo/syms ./a.out
```

## Viewing the Collected Data

Once the results are collected, the user can open any of them in the standalone GUI or generate a
command line report. Use inspxe-cl -help report or amplxe-cl -help report to see the
options available for generating reports. Here is an example of viewing the text report for functions and
modules after a VTune Amplifier XE analysis:

```
$ amplxe-cl -R hotspots -q -format text -r r003hs
Function  Module  CPU Time
--------  ------  --------
f        a.out   6.070
main      a.out   2.990

$ amplxe-cl -R hotspots -q -format text -group-by module -r r003hs
Module    CPU Time
------  --------
```
To view the results through GUI, launch the \texttt{amplxe} or \texttt{inspxe-gui} command or launch the *-gui tool and use the File > Open > Result... menu item to point to the result. Sometimes it is also convenient to copy the result to another system and view it there (for example, to open a result collected on a Linux cluster on a Windows workstation):

MPI functions are classified by the VTune Amplifier XE as system ones making its level of support in this regard similar to Intel TBB and OpenMP. This helps the user to focus on his/her code rather than MPI internals. Appendix A lists the function and module patterns used for the classification. VTune Amplifier XE GUI “Call Stack Mode” and CLI -stack-mode switches can be used to turn on the mode where the system functions are displayed and thus the internals of the MPI implementation can be viewed and analyzed. The call stack mode “User functions+1” is especially useful to find the MPI functions that consume most of CPU Time (Hotspots analysis) or waited the most (Locks and Waits analysis). For example, assume there is a call chain “main() -> foo() -> MPI_Bar() -> MPI_Bar_Impl() -> ...” where MPI_Bar() is the actual MPI API function you use and the deeper functions are MPI implementation details. The call stack modes behave as follows:

- The default “Only user functions” call stack mode will attribute time spent in the MPI calls to the user function foo() so that you can see which of your functions you can change to actually improve the performance.
- The “User functions+1” mode will attribute the time spent in the MPI implementation to the top-level system function - MPI_Bar() so that you can easily see outstandingly heavy MPI calls.
The “User/system functions” mode will show the call tree without any reattribution so that you can see where exactly in the MPI library the time was spent.

VTune Amplifier XE / Inspector XE provide the Intel TBB and OpenMP support. It is recommended to use these thread-level parallel solutions in addition to MPI-style parallelism to maximize the CPU resource usage across the cluster, and to use the VTune Amplifier XE / Inspector XE to analyze the performance / correctness of that level of parallelism. The MPI, OpenMP, and Intel TBB features in the tools are functionally independent, so all usual features of OpenMP and Intel TBB support are applicable when looking into a result collected for an MPI process.

**Limitations**

There are certain limitations in the current MPI profiling support provided by the VTune Amplifier XE / Inspector XE:

- MPI dynamic processes are not supported by the VTune Amplifier XE / Inspector XE. An example of dynamic process API is MPI_Comm_spawn.
- The data collections that use the hardware event-based sampling collector are limited to only one such collection allowed at a time on a system. When the VTune Amplifier XE is used to profile an MPI program as described above, it is the responsibility of the user to make sure that only one SEP data collection session is launched on a given host. Common ways to achieve this is using the host syntax and distribute the ranks running under the tool over different hosts.
- By default, Intel MPI implementation uses spinning when polling for the work to do. VTune Amplifier XE doesn’t recognize that time as wait time currently so it will show the CPU time spent in that polling as useful payload increasing the concurrency metric while actually this time is wait time. To avoid that, set I_MPI_WAIT_MODE environment variable to “enable” value. This will make sure the MPI implementation uses regular waits instead of spinning.

**Support of Non-Intel MPI Implementations**

The above examples assume the usage of the Intel MPI library implementation, but the described usage model will work with other MPI implementations as well if the following is kept in mind:

- VTune Amplifier XE and Inspector XE tools extract the MPI process rank from the environment variables PMI_RANK or PMI_ID (whichever is set) to detect that the process belongs to an MPI job and to capture the rank in the result directory name. If the alternative MPI implementation does not set those environment variables, the tools do not capture the rank in the result directory name and a usual automatic naming of result directories should be used. Default value for the –result-dir option is “r@@@[at]”, which results in sequence of result directories like r000hs, r001hs, and so on.
- The function/module patterns from Appendix A used for classification of time spent inside of the MPI library as system one may not cover all of modules and functions in the used MPI implementation. This may result in displaying some internal MPI functions and modules by default.
- The command-line examples listed below may need to be adjusted to work – especially when it comes to specifying different command lines to execute for different process ranks to limit the amount of processes in the job being analyzed.
- The MPI implementation needs to operate in case when there is a tool process between the launcher process (mpirun/mpiexec) and the application process. This essentially implies that the communication information should be passed using environment variables, as most MPI
implementations do. The tools would not work on an MPI implementation that tried to pass the communication information from its immediate parent process. We are unaware of any implementations that have this limitation.

Other than this, any MPI implementation should be supported for the analysis described in the paper.

Appendix A. MPI System Functions/Modules Recognized by the VTune™ Amplifier XE

Below there is a list of regular expressions in the Perl syntax that are used to classify MPI implementation modules and functions in the VTune Amplifier XE. See “Viewing the Collected Data” section for details.

The list is given for reference only. It may change from version to version without any notice.

<table>
<thead>
<tr>
<th>Function patterns</th>
<th>Linux® module patterns</th>
<th>Windows® module patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>^<em>?mpi</em>.*</td>
<td>\blibmpi_dbg\mt\so\d</td>
<td>impi\dll</td>
</tr>
<tr>
<td>^MPI_.*</td>
<td>\blibmpi_dbg\so\d</td>
<td>impid\dll</td>
</tr>
<tr>
<td>^MPI::.*</td>
<td>\blibmpigc3\so\d</td>
<td>impil\dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpigc4\so\d</td>
<td>impil\dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpigf\so\d</td>
<td>impimt\dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpi_\lp64\so\d</td>
<td>impi.dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpi_log\mt\so\d</td>
<td>libimalloc\dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpi_log\so\d</td>
<td>libmpi_\lp64\dll</td>
</tr>
<tr>
<td></td>
<td>\blibmpi_lustre\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibmpi_m\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibmpi_panfs\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibmpi_pvfs2\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibmpi\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibtmpi mx\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibtmpi_psm\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibtmpi\so\d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\blibtvmpi\so\d</td>
<td></td>
</tr>
</tbody>
</table>

Conclusion

This guide shows how to use the VTune Amplifier XE and Inspector XE tools for analyzing MPI workloads.

Additional Resources

See the VTune™ Amplifier XE, Inspector XE documentation, online MPI documentation for more details.

There are also other resources available on Intel Web site related to the usage of the VTune Amplifier XE and Inspector XE with the Intel MPI library.

- VTune Amplifier XE 2011 installation on clusters
- Why can’t Hotspots Analysis trace spawned processes of MPI job?
- Hybrid applications: Intel MPI Library and OpenMP®
Acknowledgements

Alexander Supalov and the whole Intel MPI team reviewed the information provided in the document. There was a number of other reviewers too numerous to name here – thanks to all who helped.

About the Authors

Alexei Alexandrov is a Senior Software Engineer in Intel’s Developer Products Division. Michael D’Mello is a Principal Engineer in Intel’s Developer Products Division.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

This document contains information on products in the design phase of development.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:

http://www.intel.com/design/literature.htm

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

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

Copyright© 2011 Intel Corporation. All rights reserved.
Optimization Notice

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20110307