



Intel® Cluster Studio XE 2012 for Linux* OS

Tutorial

Copyright © 2011 Intel Corporation

All Rights Reserved

Document Number: 325977-001EN

Revision: 20111108

World Wide Web: <http://www.intel.com>

Contents

Disclaimer and Legal Information	5
2. Introduction.....	7
3. Intel Software Downloads and Installation of Intel® Cluster Studio XE on Linux* OS	11
3.1 Linux* OS Installation	12
4. Integrated Development Environments for Intel® Cluster Studio XE	40
5. Getting Started with Intel® MPI Library	41
5.1 Launching MPD Daemons.....	42
5.2 How to Set Up MPD Daemons on Linux* OS	43
5.3 The mpdboot Command for Linux* OS	44
5.4 Compiling and Linking with Intel® MPI Library on Linux* OS	44
5.5 Selecting a Network Fabric.....	45
5.6 Running an MPI Program Using Intel® MPI Library on Linux* OS.....	46
5.7 Experimenting with Intel® MPI Library on Linux* OS	47
5.8 Controlling MPI Process Placement on Linux* OS	49
5.9 Using the Automatic Tuning Utility Called <i>mpitune</i>	50
5.9.1 Cluster Specific Tuning.....	52
5.9.2 MPI Application-Specific Tuning	52
5.10 Extended File I/O System Support on Linux* OS	53
5.10.1 How to Use the Environment Variables I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST	53
6. Interoperability of Intel® MPI Library with the Intel® Debugger (IDB).....	55
6.1 Login Session Preparations for Using Intel® Debugger on Linux* OS	56
7. Working with the Intel® Trace Analyzer and Collector Examples	66
7.1 Experimenting with Intel® Trace Analyzer and Collector in a Fail-Safe Mode .	68
7.2 Using itcpin to Instrument an Application.....	70
7.3 Experimenting with Intel® Trace Analyzer and Collector in Conjunction with the LD_PRELOAD Environment Variable	72
7.4 Experimenting with Intel® Trace Analyzer and Collector in Conjunction with PAPI* Counters	74
7.5 Experimenting with the Message Checking Component of Intel® Trace Collector	77
7.6 Saving a Working Environment through a Project File.....	89
7.7 Analysis of Application Imbalance	92
7.8 Analysis with the Ideal Interconnect Simulator	95
7.9 Building a Simulator with the Custom Plug-in Framework	98
8. Getting Started in Using the Intel® Math Kernel Library (Intel® MKL)	99
8.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK* Examples with the Intel® Trace Analyzer and Collector.....	103
8.2 Experimenting with the Cluster DFT Software.....	108
8.3 Experimenting with the High Performance Linpack Benchmark*	114
9. Using the Intel® MPI Benchmarks	118
10. Uninstalling the Intel® Cluster Studio XE on Linux* OS.....	120
11. Hardware Recommendations for Installation on Linux* OS	121

12.	System Administrator Checklist for Linux* OS.....	123
13.	User Checklist for Linux* OS.....	124
14.	Using the Compiler Switch -tcollect.....	126
15.	Using Co-Array Fortran	137
	15.1 Running a Co-array Fortran Example on a Distributed System.....	138
	15.2 Trouble Shooting for the Absence of Multipurpose Daemons.....	140
16.	Using the CEAN Language Extension and Programming Model.....	142
17.	Using Intel® VTune™ Amplifier XE	145
	17.1 How do I get a List of Command-line Options for the Intel® VTune™ Amplifier XE Tool?	146
	17.2 What does a Programming Example Look Like that I might run with Intel® VTune™ Amplifier XE?	146
	17.3 How do I Run and Collect Intel® VTune™ Amplifier XE Performance Information within an Intel® MPI Library Application?.....	147
	17.4 What does the Intel® VTune™ Amplifier XE Graphical User Interface Look Like?	148
18.	Using Intel® Inspector XE	149
	18.1 How do I get a List of Command-line Options for the Intel® Inspector XE Tool?.....	150
	18.2 What does a Programming Example Look Like that has a Memory Leak?	150
	18.3 How do I Run and Collect Memory Leak Information within an Intel® MPI Library Application?	151
	18.4 What does the Intel® Inspector XE Graphical User Interface Look Like?.....	151
19.	Using Intel® Parallel Advisor for non-MPI C/C++ Software Applications	156

Revision History

Document Number	Revision Number	Description	Revision Date
325977-001EN	20111108	Updated Intel® Cluster Studio XE 2012 for Linux OS Tutorial to reflect changes and improvements to the software components.	11/08/2011

Disclaimer and Legal Information

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.

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 processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, AAC, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.167, G.168, G.169, G.723.1, G.726, G.728, G.729, G.729.1, GSM AMR, GSM FR are international standards promoted by ISO, IEC, ITU, ETSI, 3GPP and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, Puma, skool, the skool logo, SMARTi, Sound Mark, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and other countries.

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

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Copyright (C) [2011], Intel Corporation. All rights reserved.

Optimization Notice

Intel's compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

[Back to Table of Contents](#)

2. Introduction

The Intel® Cluster Studio XE 2012 release on Linux* OS consists of:

1. Intel® C++ Compiler XE 12.1
2. Intel® Debugger 12.1
3. Intel® Fortran Compiler XE 12.1
4. Intel® Inspector XE 2011 Update 6
5. Intel® Integrated Performance Primitives 7.0 Update 5
6. Intel® Math Kernel Library 10.3 Update 6
7. Intel® MPI Benchmarks 3.2.3
8. Intel® MPI Library 4.0 Update 3
9. Intel® Threading Building Blocks 4.0
10. Intel® Trace Analyzer and Collector 8.0 Update 3
11. Intel® VTune™ Amplifier XE 2011 Update 5

The software architecture of the Intel Cluster Studio XE for Linux OS is illustrated in Figure 2.1:

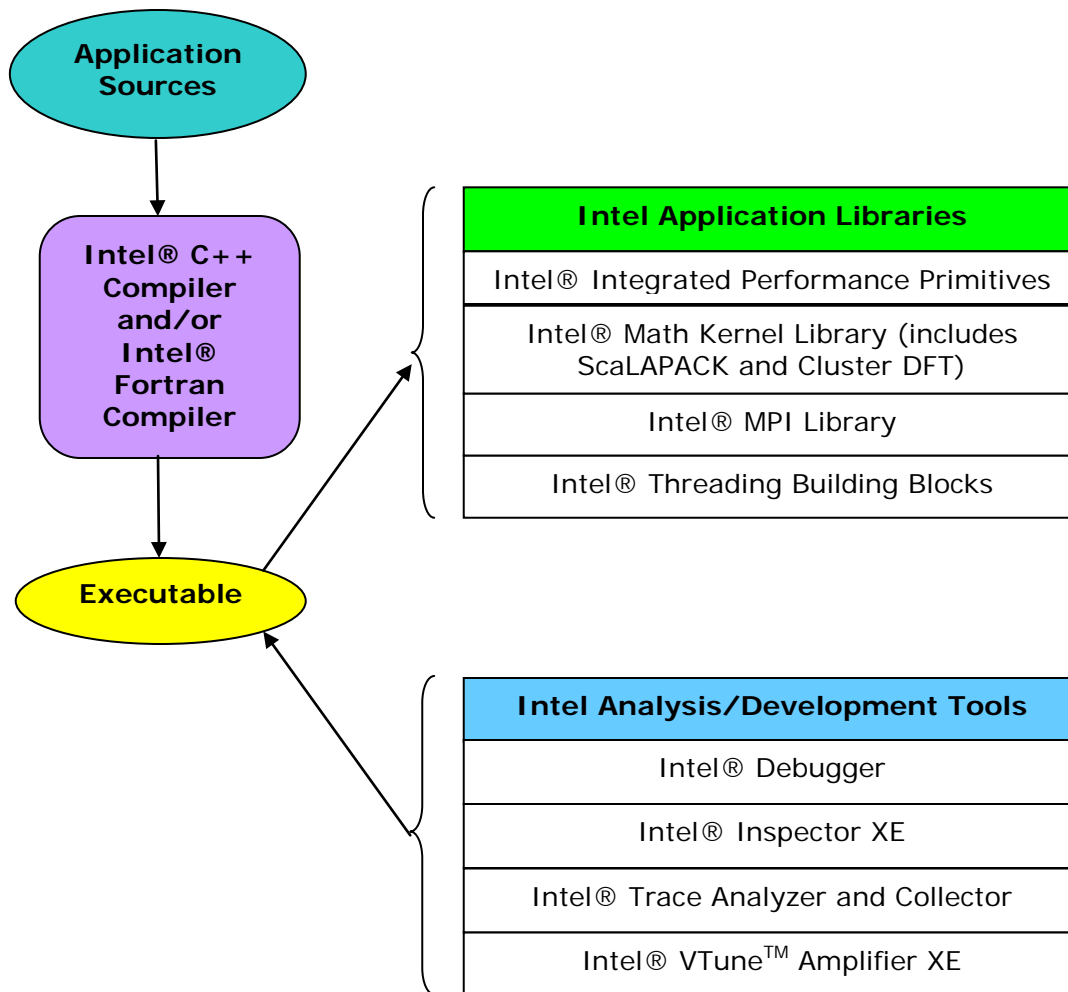


Figure 2.1 – The Software Architecture of Intel® Cluster Studio XE on Linux* OS

The following are acronyms and definitions of those acronyms that may be referenced within this document.

Acronym	Definition
ABI	Application Binary Interface – describes the low-level interface an application program and the operating system, between an application and its libraries, or between component parts of an application.
BLACS	Basic Linear Algebra Communication Subprograms – provides a linear algebra oriented message passing interface for distributed memory computing platforms.
BLAS	Basic Linear Algebra Subroutines
DAPL*	Direct Access Program Library - an Application Program Interface (API) for Remote Data Memory Access (RDMA).
DFT	Discrete Fourier Transform
Ethernet	Ethernet is the predominant local area networking technology. It transports data over a variety of electrical or optical media. It transports any of several upper layer protocols through data packet transmissions.
GB	Gigabyte
ICS	Intel® Cluster Studio
ICSXE	Intel® Cluster Studio XE
IMB	Intel® MPI Benchmarks
IP	Internet protocol
ITA or ita	Intel® Trace Analyzer
ITAC or itac	Intel® Trace Analyzer and Collector
ITC or itc	Intel® Trace Collector
MPD	Multi-purpose daemon protocol – a daemon that runs on each node of a cluster. These MPDs configure the nodes of the cluster into a “virtual machine” that is capable of running MPI programs.
MPI	Message Passing Interface - an industry standard, message-passing protocol that typically uses a two-sided send-receive model to transfer messages between processes.
NFS	The Network File System (acronym NFS) is a client/server application that lets a computer user view and optionally store and update file on a remote computer as though they were on the user's own computer. The user's system needs to have an NFS client and the other computer needs the NFS server. Both of them require that you also have TCP/IP installed since the NFS server and client use TCP/IP as the program that sends the files and updates back and forth.
PVM*	Parallel Virtual Machine

RAM	Random Access Memory
RDMA	Remote Direct Memory Access - this capability allows processes executing on one node of a cluster to be able to "directly" access (execute reads or writes against) the memory of processes within the same user job executing on a different node of the cluster.
RDSSM	TCP + shared memory + DAPL* (for SMP clusters connected through RDMA-capable fabrics)
RPM*	Red Hat Package Manager* - a system that eases installation, verification, upgrading, and uninstalling Linux packages.
ScaLAPACK*	SCALable LAPACK - an acronym for Scalable Linear Algebra Package or Scalable LAPACK.
shm	Shared memory only (no sockets)
SMP	Symmetric Multi-processor
ssm	TCP + shared memory (for SMP clusters connected through Ethernet)
STF	Structured Trace Format – a trace file format used by the Intel Trace Collector for efficiently recording data, and this trace format is used by the Intel Trace Analyzer for performance analysis.
TCP	Transmission Control Protocol - a session-oriented streaming transport protocol which provides sequencing, error detection and correction, flow control, congestion control and multiplexing.
VML	Vector Math Library
VSL	Vector Statistical Library

[Back to Table of Contents](#)

3. Intel Software Downloads and Installation of Intel® Cluster Studio XE on Linux* OS

The Intel Cluster Studio XE installation process on Linux OS is comprised of eight basic steps. The Intel Cluster Studio XE 2012 package consists of the following components:

Software Component	Default Installation Directory on IA-32 Architecture for Linux OS	Default Installation Directory on Intel® 64 Architecture for Linux OS
Intel® C++ Compiler XE 12.1	/opt/intel/composer_xe_2011_sp1.6.0xx	/opt/intel/composer_xe_2011_sp1.6.0xx
Intel® Debugger 12.1	/opt/intel/composer_xe_2011_sp1.6.0xx	/opt/intel/composer_xe_2011_sp1.6.0xx
Intel® Fortran Compiler XE 12.1	/opt/intel/composer_xe_2011_sp1.6.0xx	/opt/intel/composer_xe_2011_sp1.6.0xx
Intel® Inspector XE 2011 Update 4	/opt/intel/inspector_xe_2011	/opt/intel/inspector_xe_2011
Intel® Integrated Performance Primitives 7.0 Update 5	/opt/intel/composer_xe_2011_sp1.6.0xx/ipp	/opt/intel/composer_xe_2011_sp1.6.0xx/ipp
Intel® Math Kernel Library (MKL) 10.3 Update 6	/opt/intel/composer_xe_2011_sp1.6.0xx/mkl	/opt/intel/composer_xe_2011_sp1.6.0xx/mkl
Intel® MPI Benchmarks 3.2.3	/opt/intel/icsxe/2012.0.0xx/imb	/opt/intel/icsxe/2012.0.0xx/imb
Intel® MPI Library 4.0 Update 3	/opt/intel/icsxe/2012.0.0xx/impi	/opt/intel/icsxe/2012.0.0xx/impi
Intel®	/opt/intel/	/opt/intel/

Threading Building Blocks 4.0	composer_xe_2011_sp1.6.0xx/tbb	composer_xe_2011_sp1.6.0xx/tbb
Intel® Trace Analyzer and Collector 8.0 Update 3	/opt/intel/icsxe/2012.0.0xx/itac	/opt/intel/icsxe/2012.0.0xx/itac
Intel® Vtune™ Amplifier XE 2011 Update 3	/opt/intel/vtune_amplifier_xe_2011	/opt/intel/vtune_amplifier_xe_2011

For the table above, references to 0xx in the directory path represents a build number such as 037.

NOTE: The Intel Cluster Studio XE installer will automatically make the appropriate selection of binaries, scripts, and text files from its installation archive based on the Intel processor architecture of the host system where the installation process is initiated. You do not have to worry about selecting the correct software component names for the given Intel® architecture.

As a user of the Intel Cluster Studio XE on Linux OS, you may need assistance from your system administrator in installing the associated software packages on your cluster system, if the installation directory requires system administrative write privileges (for example, /opt/intel on Linux OS). This assumes that your login account does not have administrative capabilities.

[Back to Table of Contents](#)

3.1 Linux* OS Installation

To begin installation on Linux*:

1. For Linux Systems, the Intel® Cluster Studio XE installer can do:
 - a. An install of the software on a single file server that is accessible to all nodes of the cluster.
 - b. A distributed install where the software components are installed on each node of the cluster.

For a distributed install, a machines.LINUX file will either need to be created, or an existing machines.LINUX file can be used by the Intel Cluster Studio XE installer to deploy amongst the nodes of the cluster, the appropriate Cluster Studio XE software components. This machines.LINUX file contains a list of the

computing nodes (for example, the hostnames) for the cluster. The format is one hostname per line:

```
hostname
```

The hostname should be the same as the result from the Linux command "hostname". An example of the content for the file `machines.LINUX`, where a contrived cluster consists of eight nodes might be:

```
clusternode1
clusternode2
clusternode3
clusternode4
clusternode5
clusternode6
clusternode7
clusternode8
```

A line of text above is considered a comment line if column one contains the "#" symbol. It is always assumed that the first node in the list is the master node. The remaining nodes are the compute nodes. The text `clusternode1` and `clusternode2`, for example, represent the names of two of the nodes in a contrived computing cluster. You can also use the contents of the `machines.LINUX` file to construct an `mpd.hosts` file for the multi-purpose daemon (MPD) protocol. The MPD protocol is used for running MPI applications that utilize Intel MPI Library.

2. In preparation for the installation, you may want to create a staging area. On the system where the Intel Cluster Studio XE software components are to be installed, it is recommended that a staging area be constructed in a directory such as `/tmp`. An example folder path staging area might be:

```
/tmp/icsxe_staging_area
```

where `icsxe_staging_area` is an acronym for Intel Cluster Studio XE staging area.

3. Upon registering for Intel Cluster Studio XE 2012, you will receive a serial number (for example, C111-12345678) for this product. Your serial number can be found within the email receipt of your product purchase. Go to the [Intel® Software Development Products Registration Center](#) site and provide the product serial number information. Once the admission has been granted into the registration center, you will be able to access the Intel® Premier Web pages for software support.
4. The license for the Intel Cluster Studio XE license file that is provided to you should be placed in a directory pointed to by the `INTEL_LICENSE_FILE` environment variable. Do not change the file name because the ".lic" extension is critical. Common locations for the attached license file are:

`<installation_path>/licenses`

where `licenses` is a sub-directory. For example, on the cluster system where the Intel Cluster Studio XE software is to be installed, all licenses for Intel-based software products might be placed in:

`/opt/intel/licenses`

It is also imperative that you and/or the system administrator set the environment variable `INTEL_LICENSE_FILE` to the directory path where the Intel software licenses will reside *prior* to doing an installation of the Intel Cluster Studio XE. For Bourne* Shell or Korn* Shell the syntax for setting the `INTEL_LICENSE_FILE` environment variable might be:

```
export INTEL_LICENSE_FILE=/opt/intel/licenses
```

For C Shell, the syntax might be:

```
setenv INTEL_LICENSE_FILE /opt/intel/licenses
```

5. Patrons can place the Intel Cluster Studio XE software package into the staging area folder.
6. The installer package for the Intel Cluster Studio XE has the following general nomenclature:

`l_ics_<major>.<update>.<package_num>.tar.gz`

where `<major>.<update>.<package_num>` is a string such as:

`2012.0.xxx`

The `<package_num>` meta-symbol is a string such as `037`. This string indicates the package number.

The command:

```
tar -xvzf l_ics_<major>.<update>.<package_num>.tar.gz
```

will create a sub-directory called `l_ics_<major>.<update>.<package_num>`. Change to that directory with the shell command:

```
cd l_ics_<major>.<update>.<package_num>
```

For example, suppose the installation package is called `l_ics_2012.0.037.tar.gz`. In the staging area that has been created, type the command:

```
tar -xvzf l_ics_2012.0.037.tar.gz
```

This will create a sub-directory called `l_ics_2012.0.037`. Change to that directory with the shell command:

```
cd l_ics_2012.0.037
```

In that folder, make sure that `machines.LINUX` file, as mentioned in item 1 above, is either in this directory or you should know the directory path to this file.

7. Also within the `l_ics_<major>.<update>.<package_num>` directory staging area, the `expect` shell script file called `"sshconnectivity.exp"` can be used to help you establish secure shell connectivity on a cluster system, where `expect` is a tool for automating interactive applications. To run `"sshconnectivity.exp"`, the `expect` runtime software needs to be installed on your Linux system. To make sure that the `expect` runtime software is properly installed, type:

```
which expect
```

If you encounter a "Command not found." error message, you can download the `expect` software package from the following URL:

<http://expect.nist.gov/>

The syntax for the `"sshconnectivity.exp"` command is:

```
./sshconnectivity.exp machines.LINUX
```

This `expect` shell script will create or update a `~/ .ssh` directory on each node of the cluster beginning with the master node which must be the first name listed in the `machines.LINUX` file. This script will prompt you for your password twice.

```
Enter your user password:  
Re-enter your user password:
```

To provide security each time you enter your user password, asterisks will appear in lieu of the password text. Upon successful completion of the script, the following message fragment will appear:

```
*****  
Node count = 4  
Secure shell connectivity was established on all nodes.  
*****  
*****
```

A log of the transactions for this script will be recorded in:

```
/tmp/sshconnectivity.<login-name>.log
```

where *<login-name>* is a meta-symbol for your actual login.

NOTE: The shell script `sshconnectivity.exp` will remove the write access capability on the group and other “permission categories” for the user’s home directory folder. If this is not done, a password prompt will continue to be issued for any secure shell activity.

This process of establishing secure shell connectivity in step 7 above is demonstrated by the following complete graph¹ (Figure 3.1) illustration where a vertex in the graph represents a cluster computing node, and an edge between two vertices connotes that the two cluster computing nodes have exchanged public keys for secure shell connectivity. Secure shell connectivity is intended to provide secure, encrypted communication channels between two or more cluster nodes over an insecure network.

The script `sshconnectivity.exp` will call the appropriate secure shell utilities to generate a private key and a public key for each node of the cluster.

¹ A mathematical definition of a complete graph in graph theory is a simple graph where an edge connects every pair of vertices. The complete graph on n vertices has n vertices and $n(n - 1) / 2$ edges, and is denoted by K_n . Each vertex in the graph has degree $n - 1$. All complete graphs are their own cliques (a maximal complete graph). A graph of this type is maximally connected because the only vertex cut which disconnects the graph is the complete set of vertices.

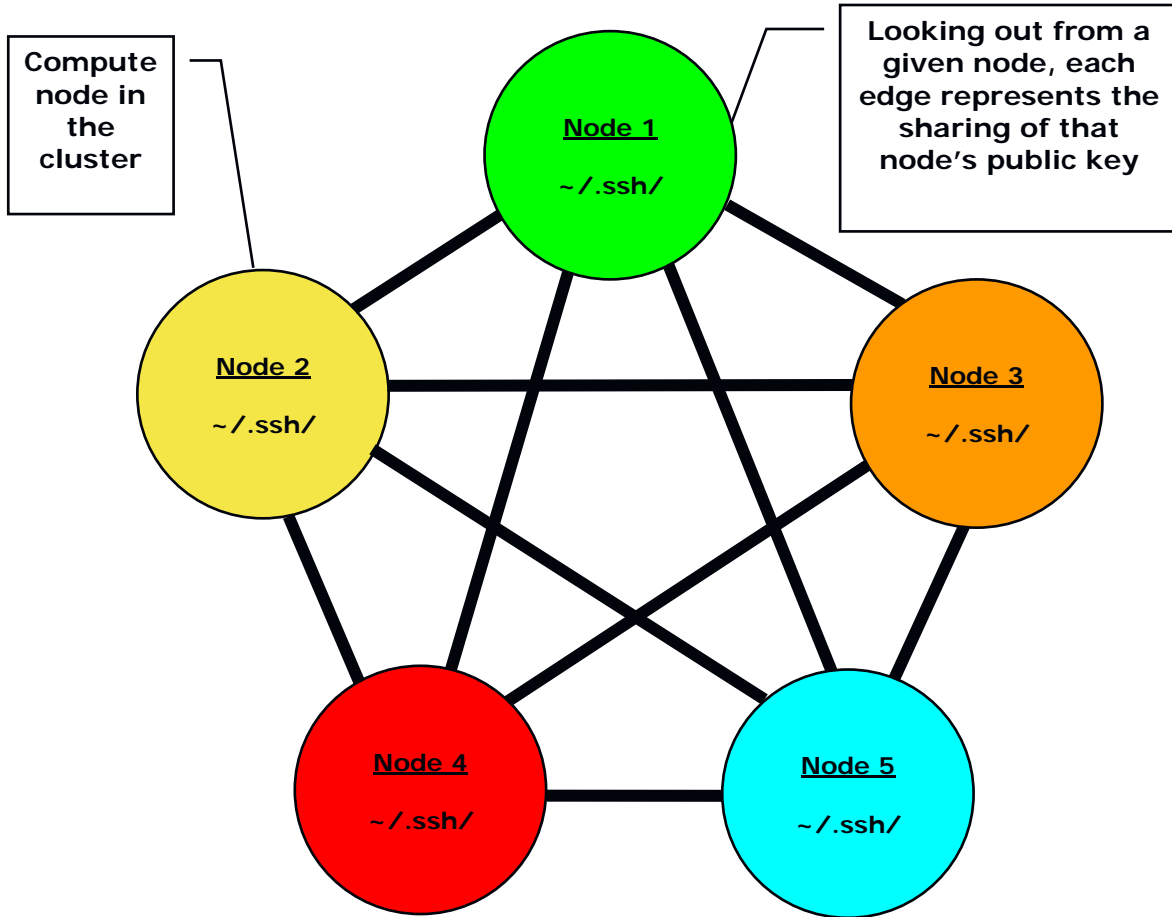


Figure 3.1 – Illustration of Secure Shell Connectivity for a Computing Cluster

For the complete graph example in Figure 3.1, suppose there are nodes (vertices) 1 to n in the cluster. For a given node i , nodes 1 to $i - 1$ and nodes $i + 1$ to n are provided with the public key from node i . The user's public keys for a given node will be stored in the `~/.ssh/` folder associated with the user's home directory for that computing node. Since there are $n - 1$ edges to a given node i in Figure 3.1, that node i will have $n - 1$ public keys in the `~/.ssh/` folder that were provided by the other $n - 1$ nodes in the cluster. The example in Figure 3.1 represents a computing cluster that has at total of five nodes. The edges connecting a node indicate that that node has received four public keys from the remaining computing nodes. Also looking out from a given node indicates that

the given node has provided its own public key to the remaining nodes that are reachable through the four edge paths.

If the home directory for a cluster is shared by all of the nodes of the cluster, for example, all of the nodes use the same `~/ .ssh` folder, the connectivity illustrated in Figure 3.1 is represented through the contents of the `~/ .ssh/known_hosts` file.

8. Make sure that the Java* Runtime Environment package is installed on your system. The directory path for where the Java* Runtime Environment may reside might be:

```
/usr/java
```

If you cannot find the Java* Runtime Environment library installation on your system, visit the URL:

<http://www.java.com/en/download/>

to download the appropriate version of the Java* Runtime Environment. After doing the download, install the Java* Runtime Environment on your system. You may need a system administrator to help you with the installation.

If you have located an existing and compatible Java* Runtime Environment library on your system, or you have proceeded to visit the URL above and completed a download and installation, set your PATH environment variable to include the directory path to the Java* Runtime Environment library. The Bourne* and Korn* Shell syntax for setting the PATH environment variable might be something like the following:

```
export PATH=/usr/java/jre1.5.0_22/bin:$PATH
```

For C Shell, the syntax for setting the PATH environment variable might be something like:

```
setenv PATH /usr/java/jre1.5.0_22/bin:$PATH
```

Once secure shell connectivity is established and the Java* Runtime Environment is verified, type a variation of the `install.sh` as illustrated in Figure 3.2.

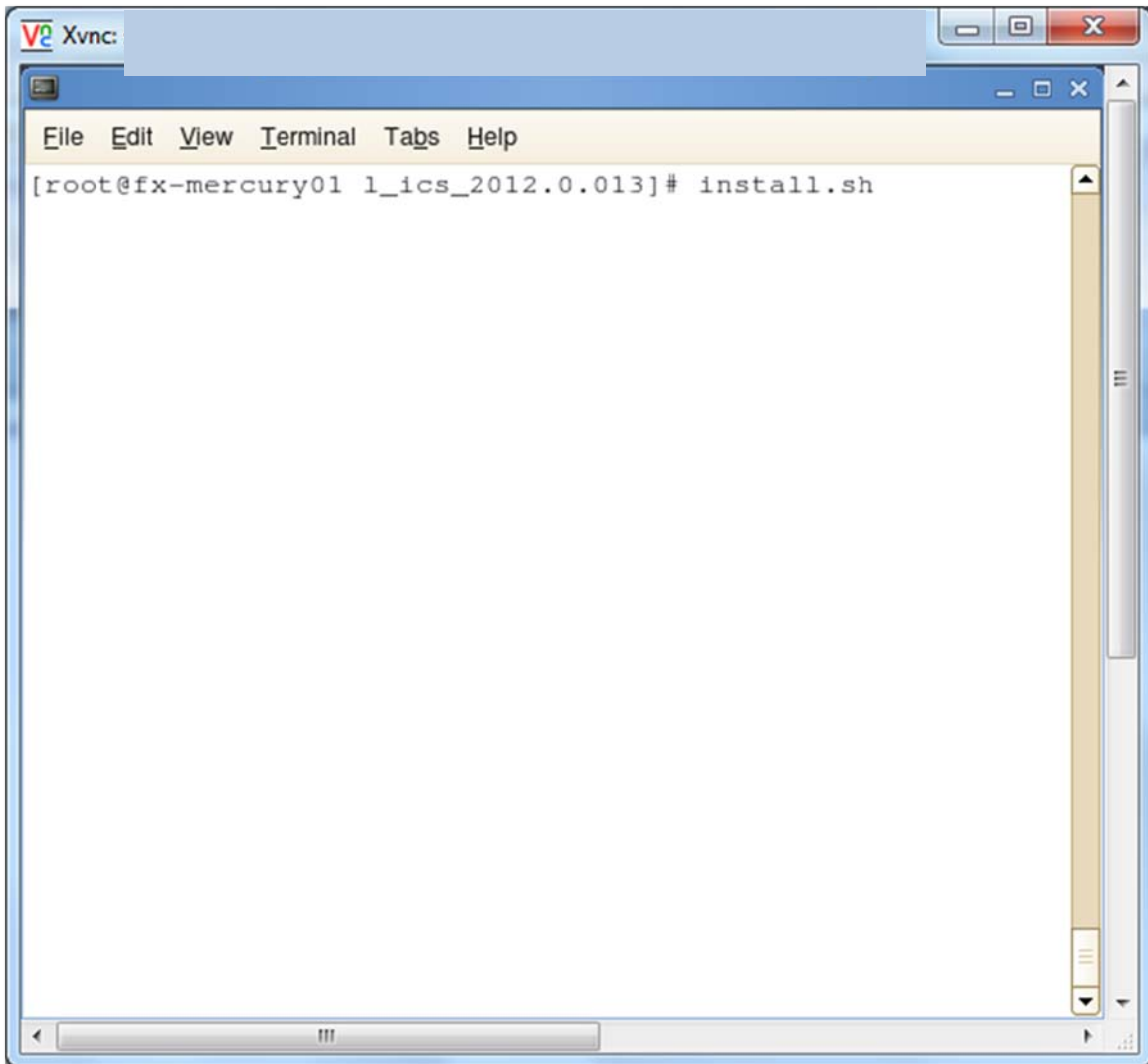


Figure 3.2 – Initiating the installation process with the command `install.sh`

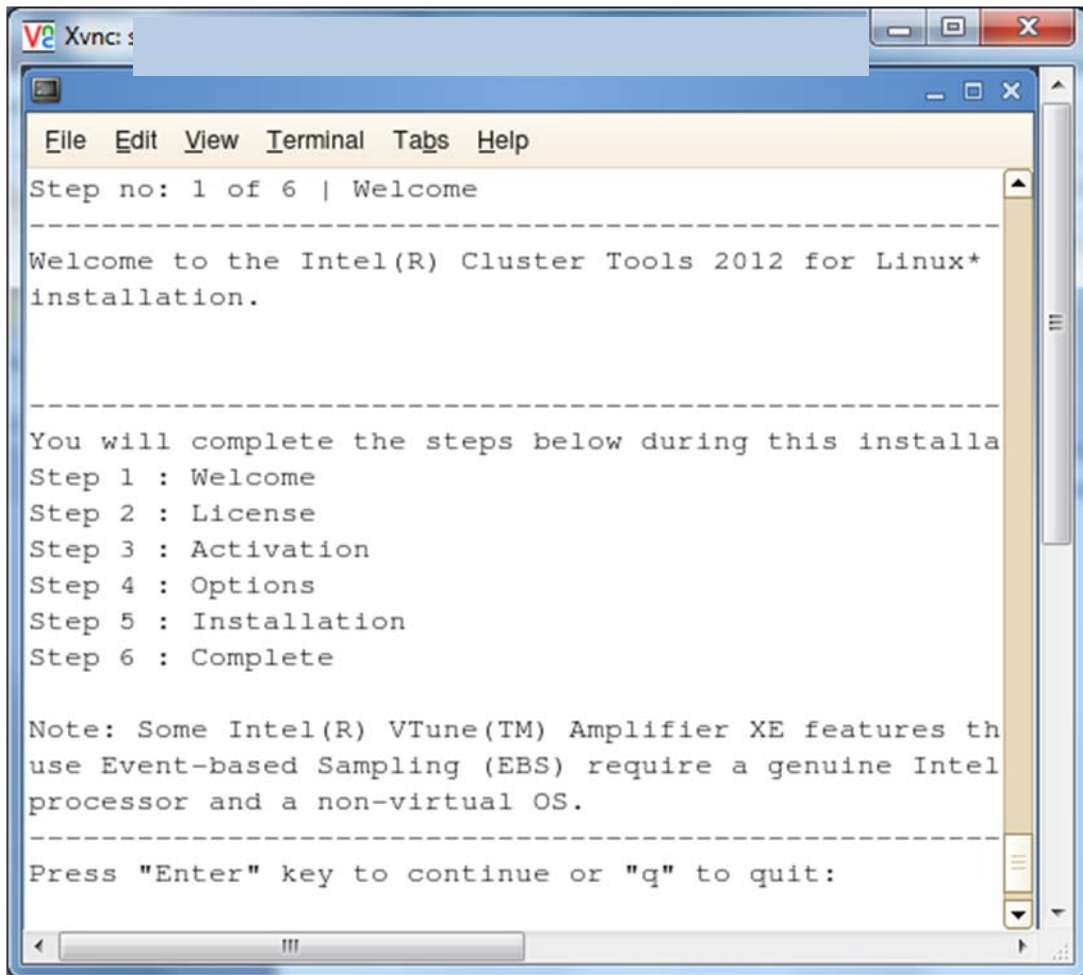


Figure 3.3 – The six steps in the installation process

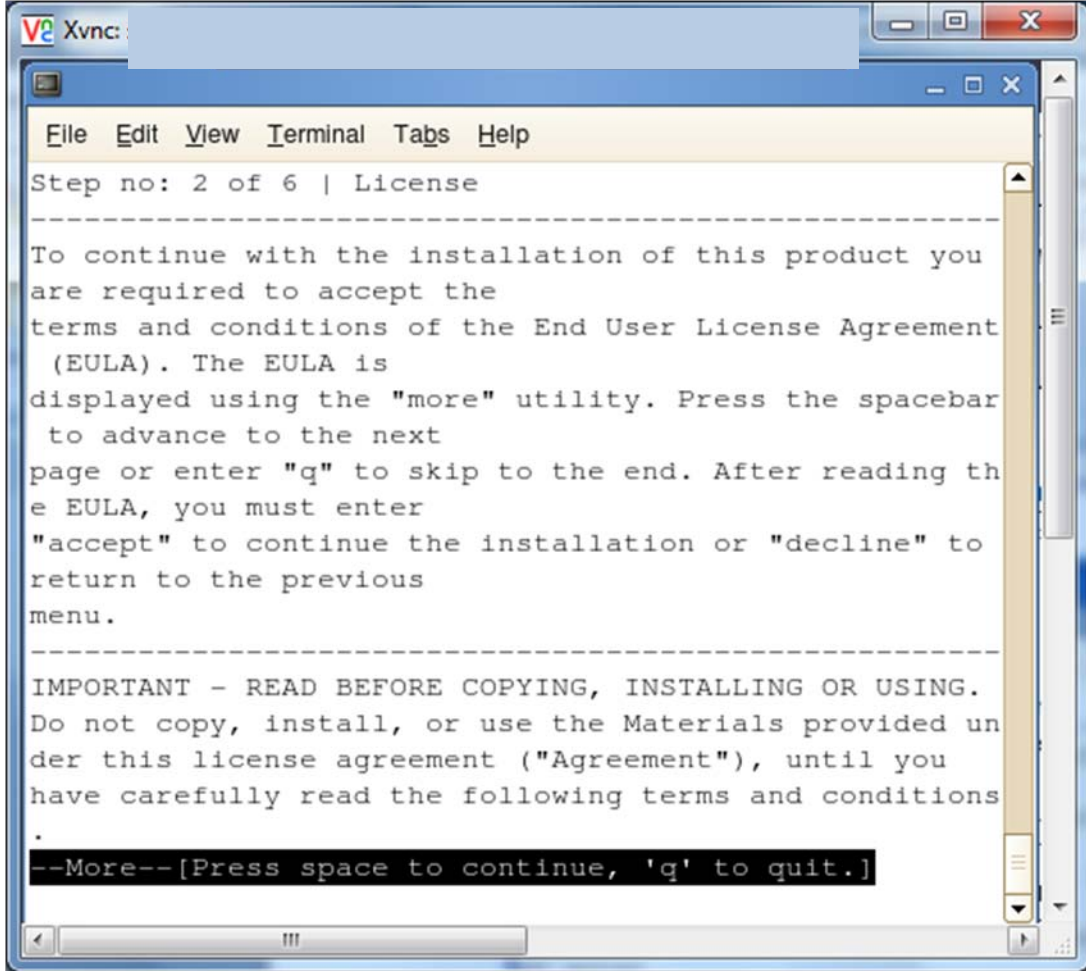


Figure 3.4 – License agreement

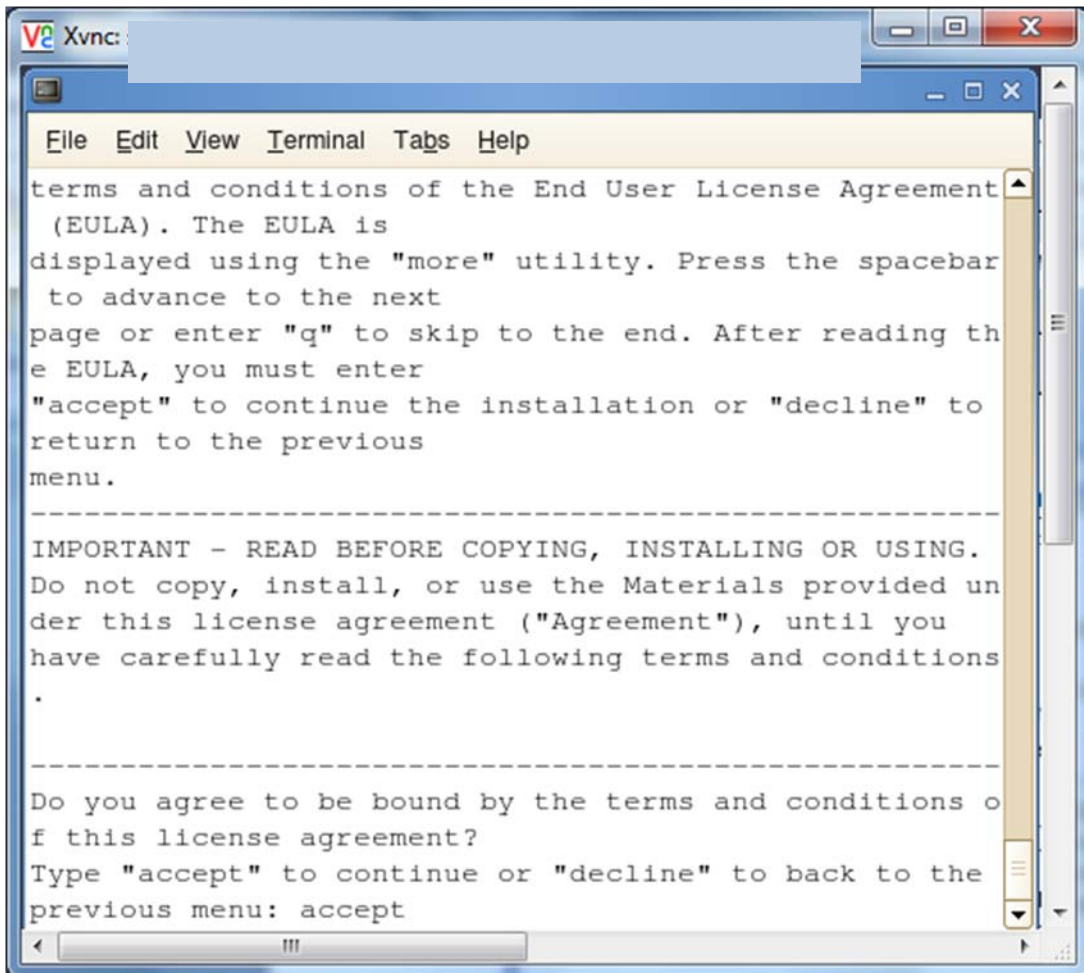


Figure 3.5 – Enter the accept word to acknowledge the terms of the license agreement

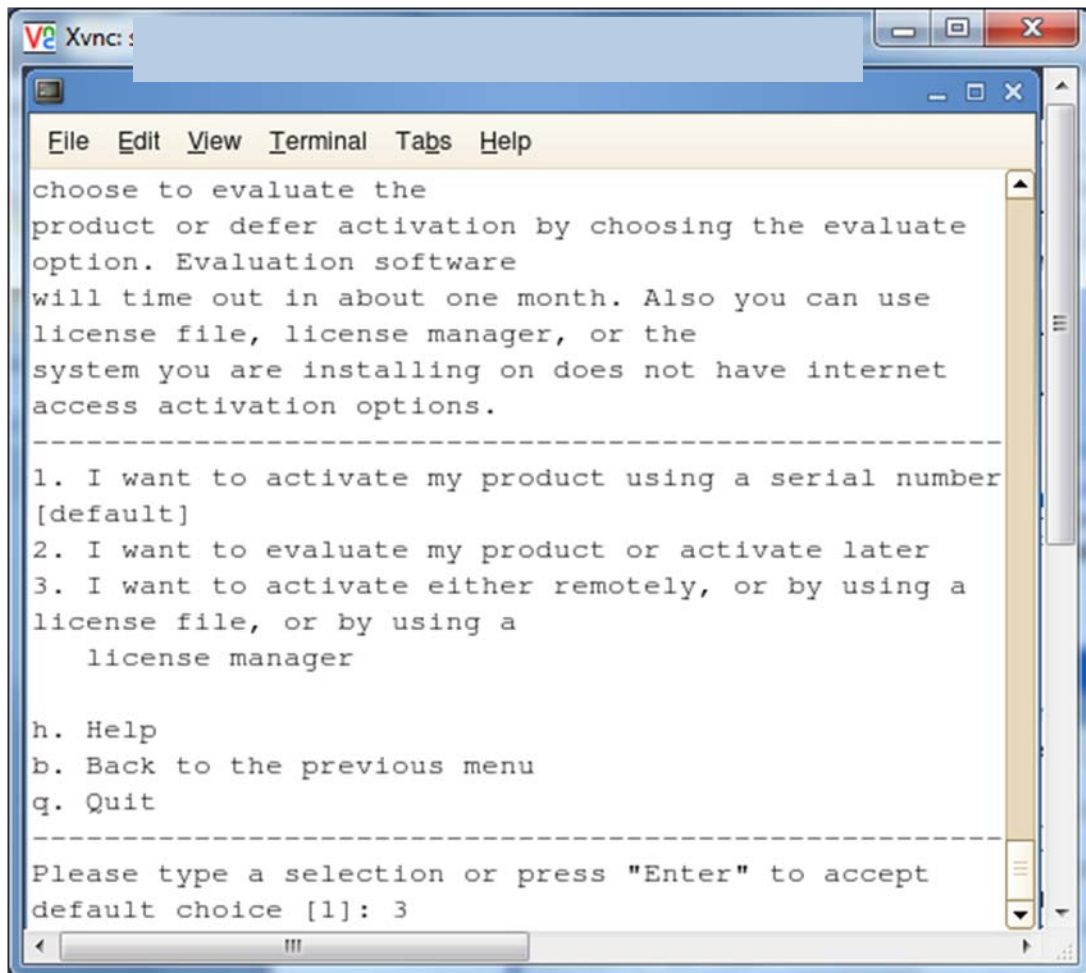


Figure 3.6 – Step 3 – Select option 3 where you want to provide a license file to complete the installation process

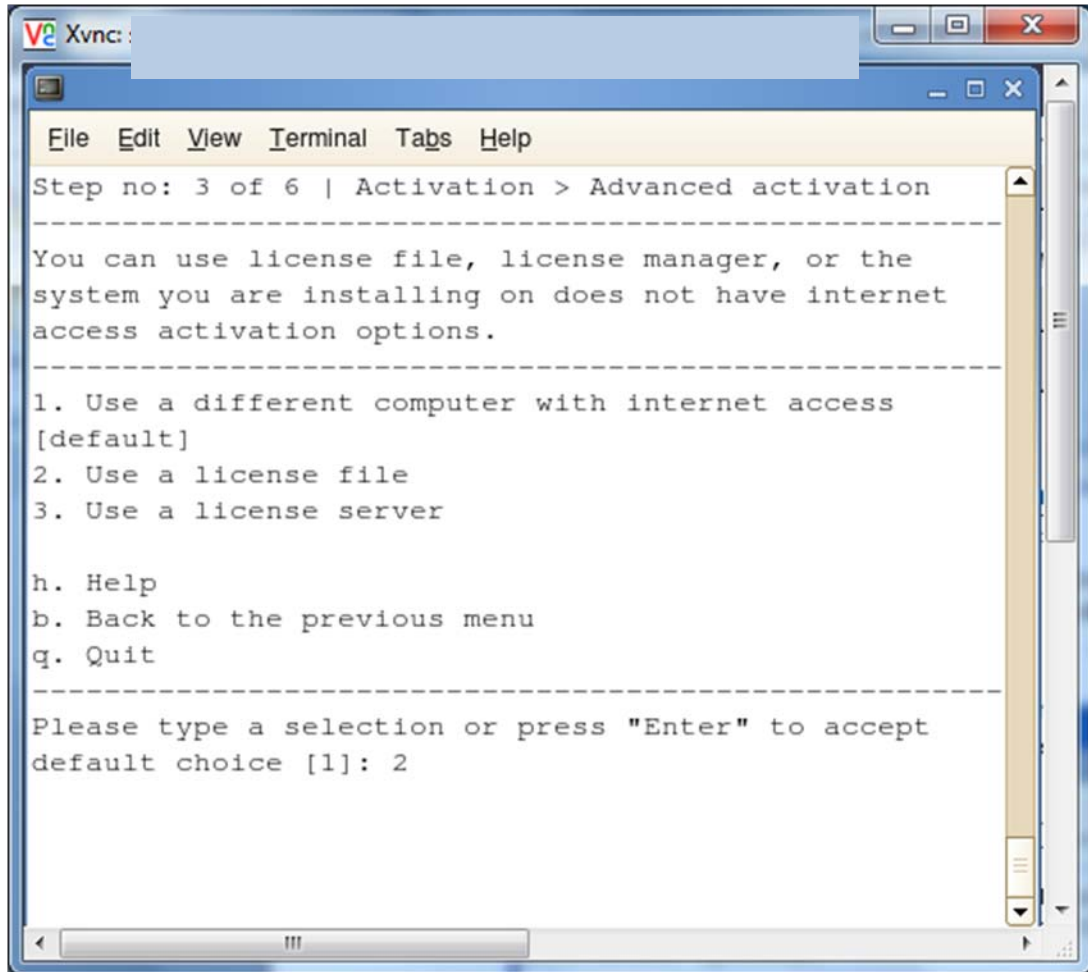


Figure 3.7 - Step 3 Continued – Selection option 2 to direct the installer to ask for a license file

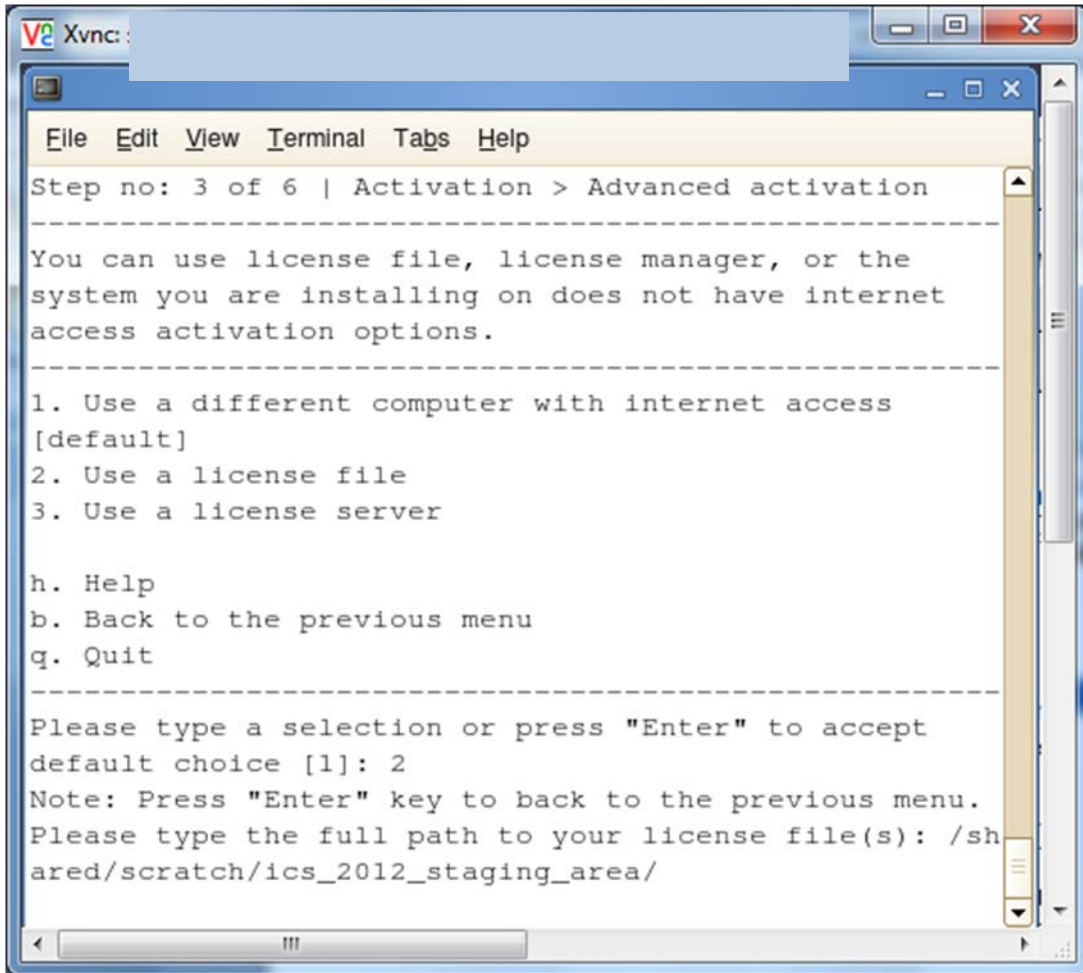


Figure 3.8 – Step 3 Continued – Provide a directory path to where the license file resides

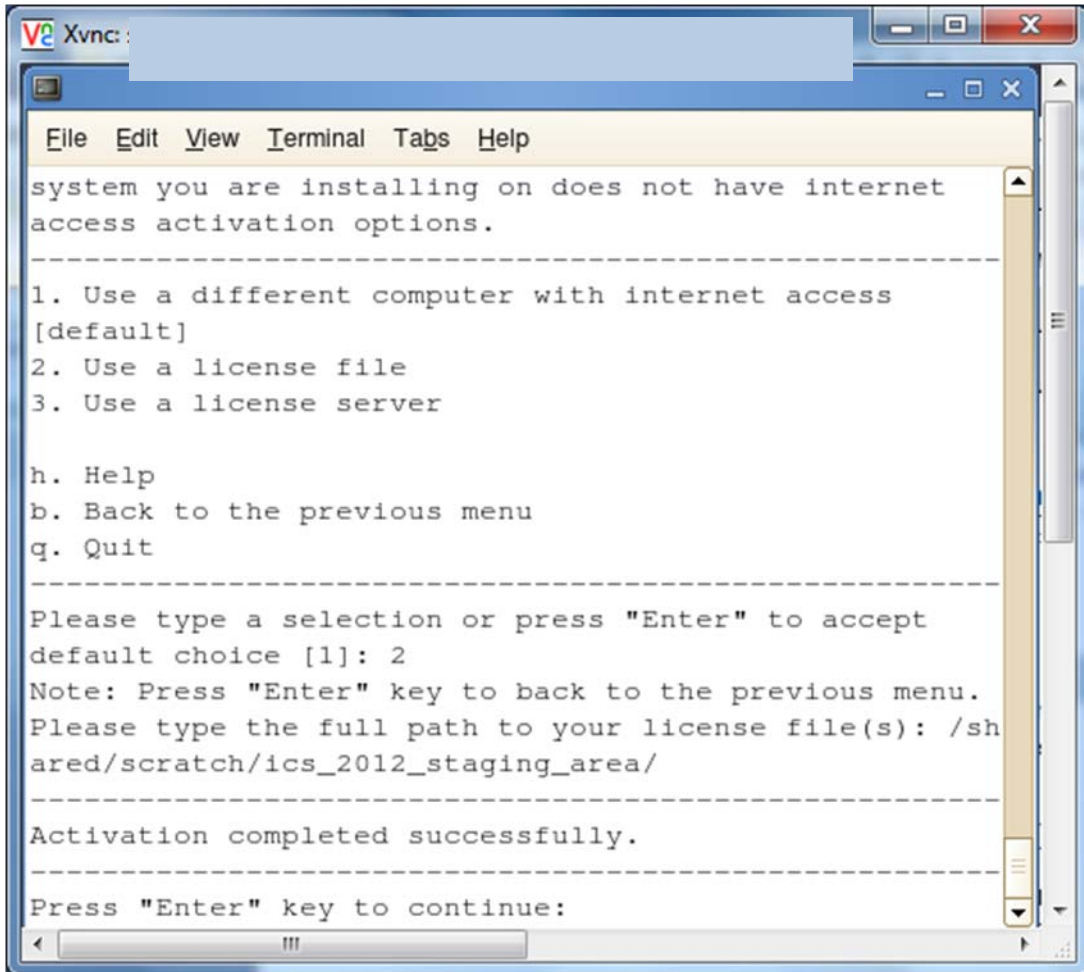


Figure 3.9 – Verification of license activation

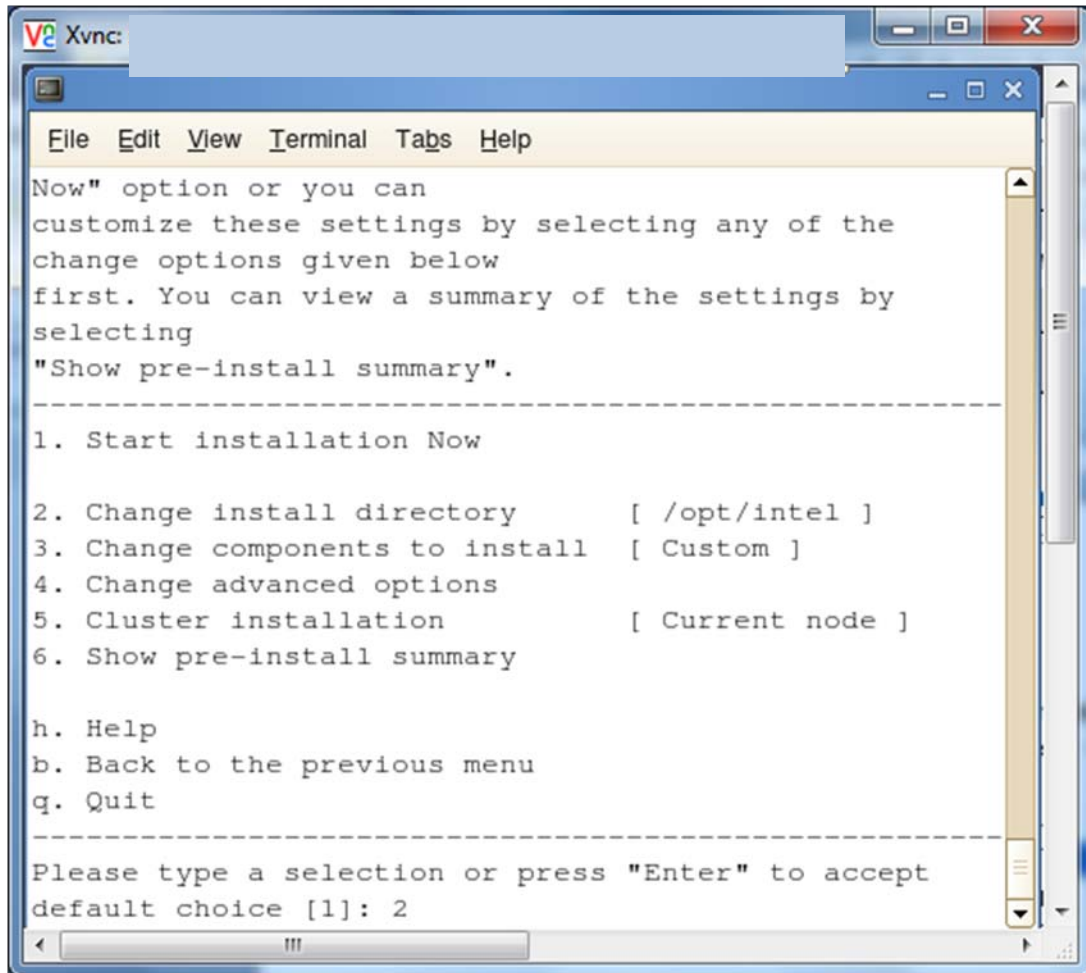


Figure 3.10 – Step 4 – Select option 2 in order to change the install directory from the default which is /opt/intel

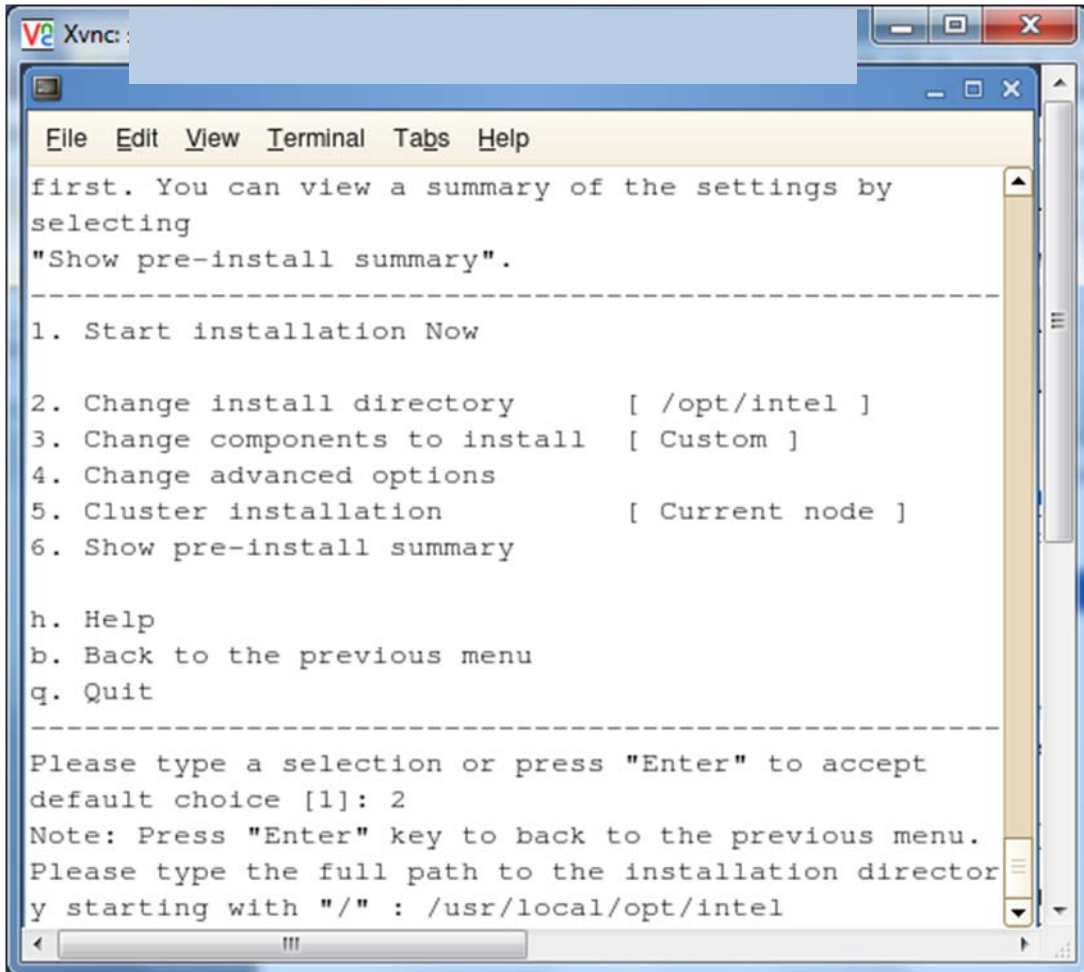


Figure 3.11 - Step 4 Continued – Provide the alternative directory path

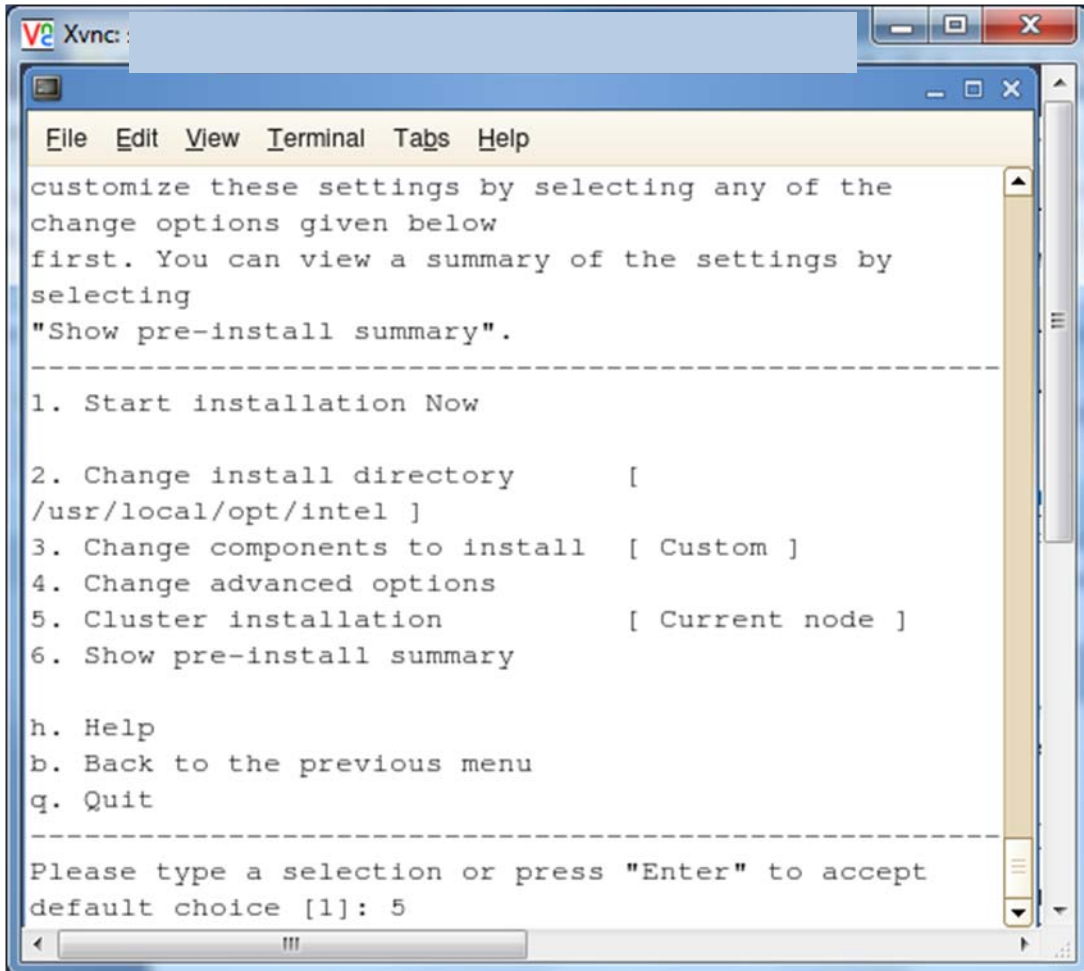


Figure 3.12 – Step 4 Continued – Select option 5 so as to do a distributed install as opposed to installing only on the current (I.e., the master) node

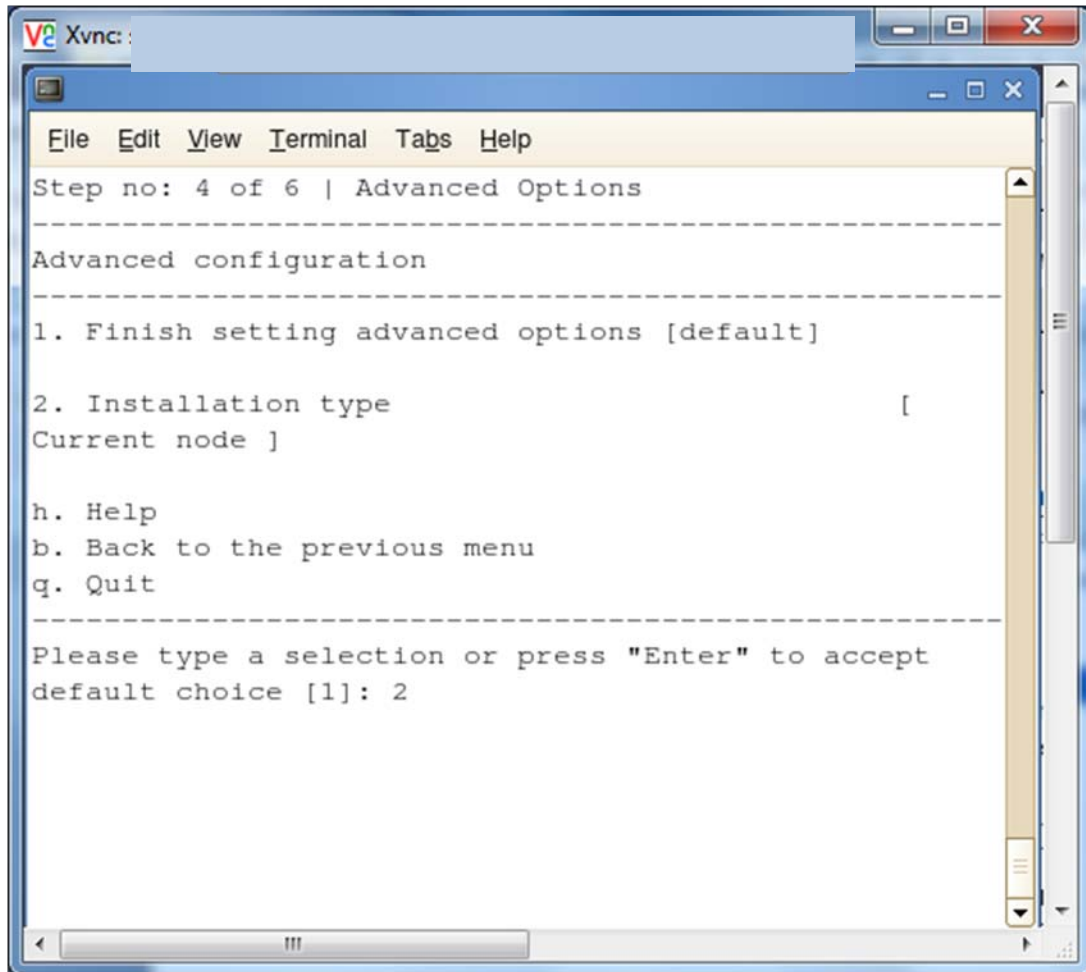


Figure 3.13 – Step 4 Continued – Select option 2 to continue the process of doing a distributed install

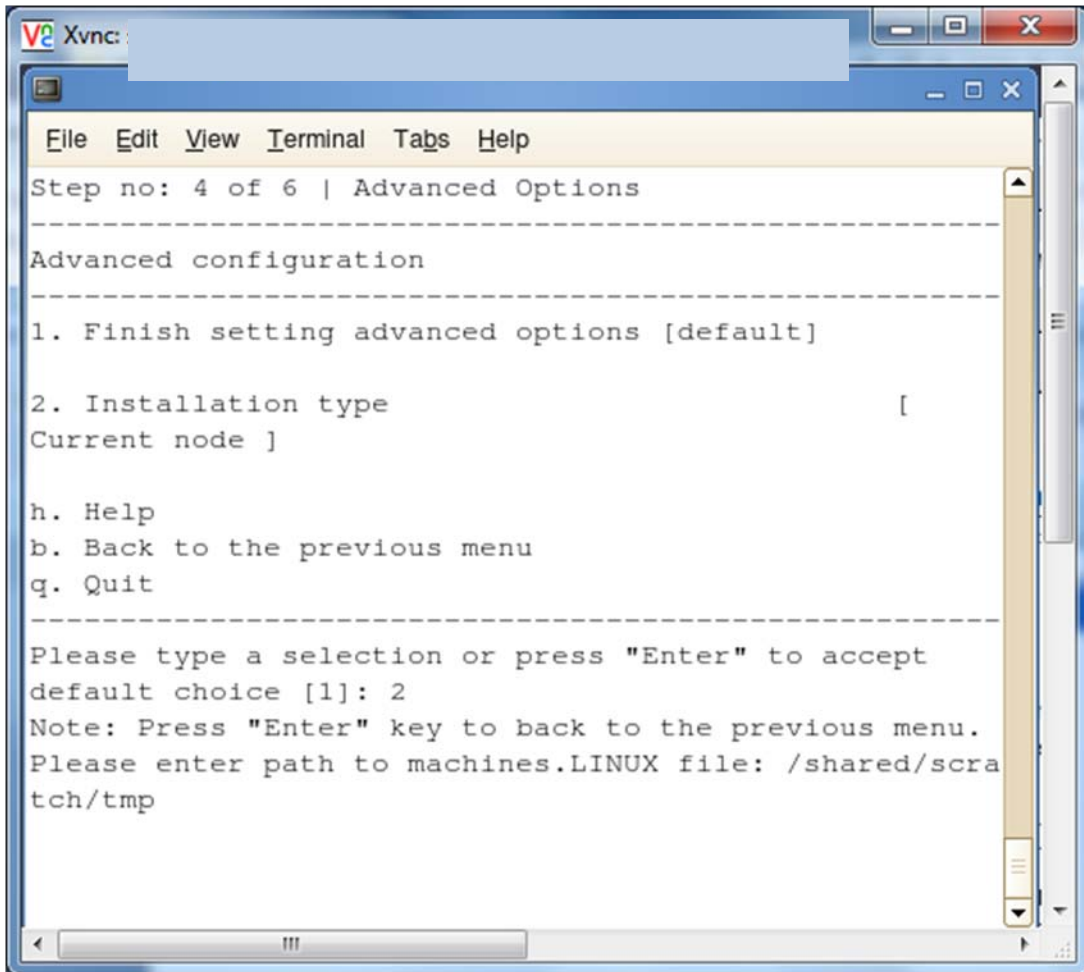


Figure 3.14 – Step 4 Continued – Provide a directory path to a file that contains a list of the nodes for the cluster

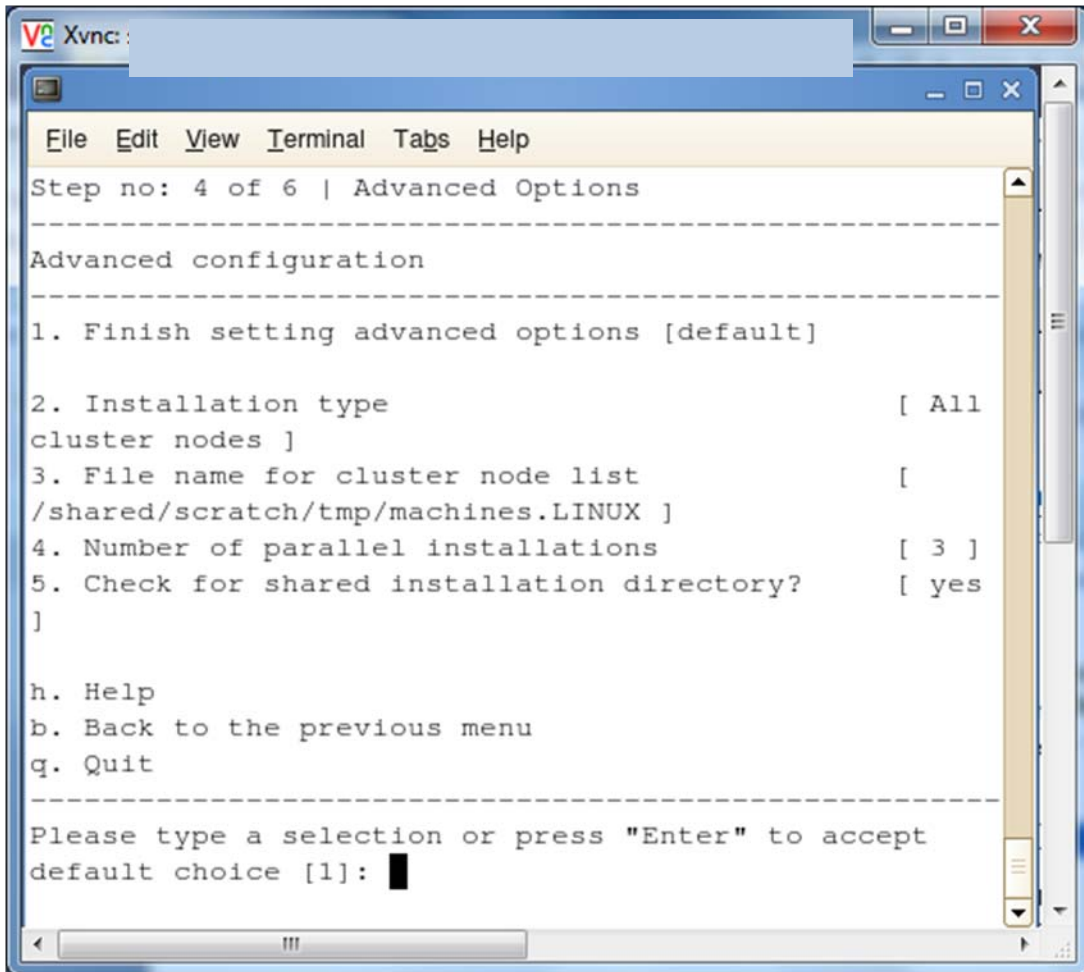


Figure 3.15 – Step 4 Continued – Select the default option of 1 as an indication that all advanced configuration options have been exercised

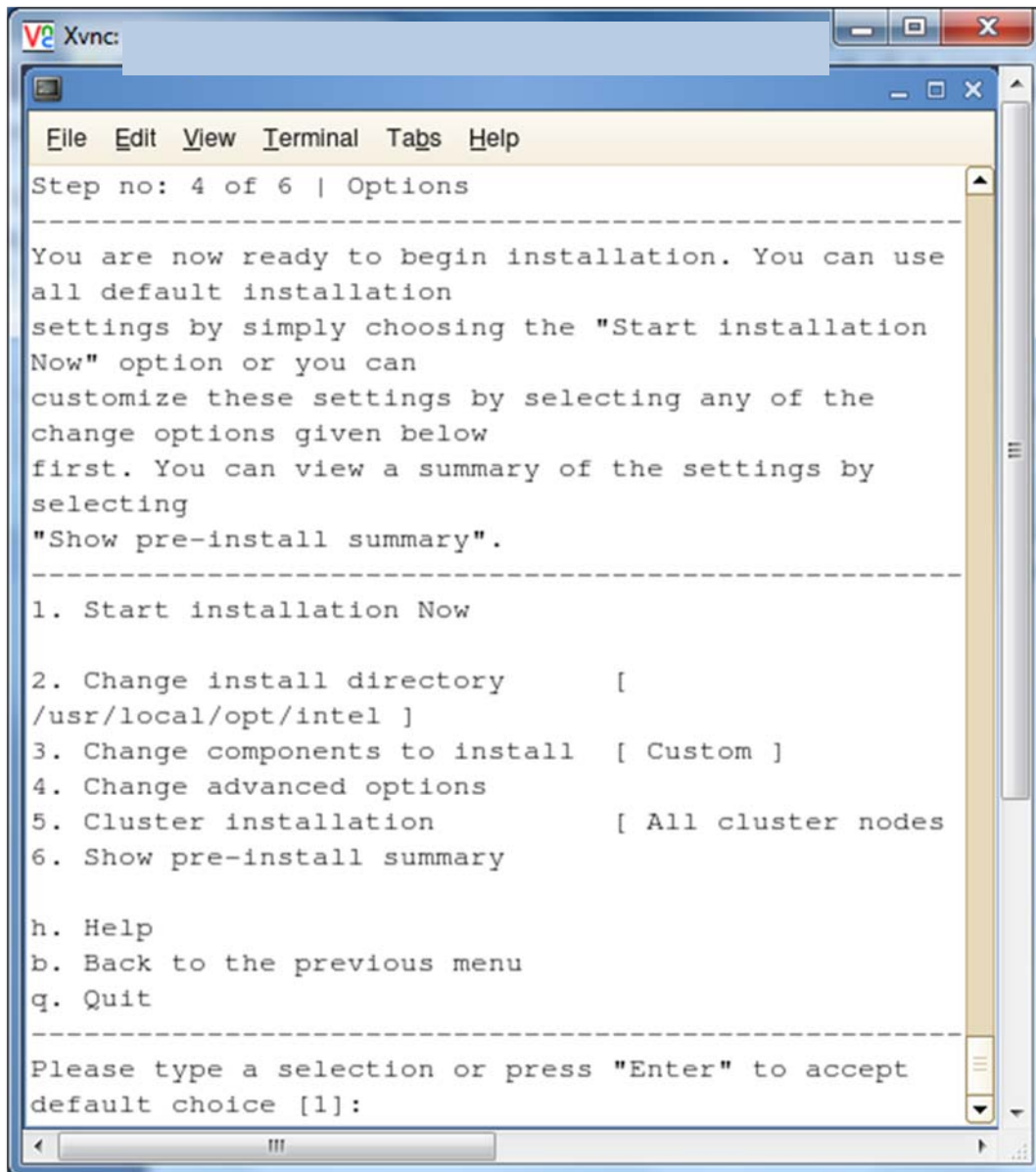


Figure 3.16 – Step 4 Continued – Select the default option of 1 as an indication that you ready to start the installation

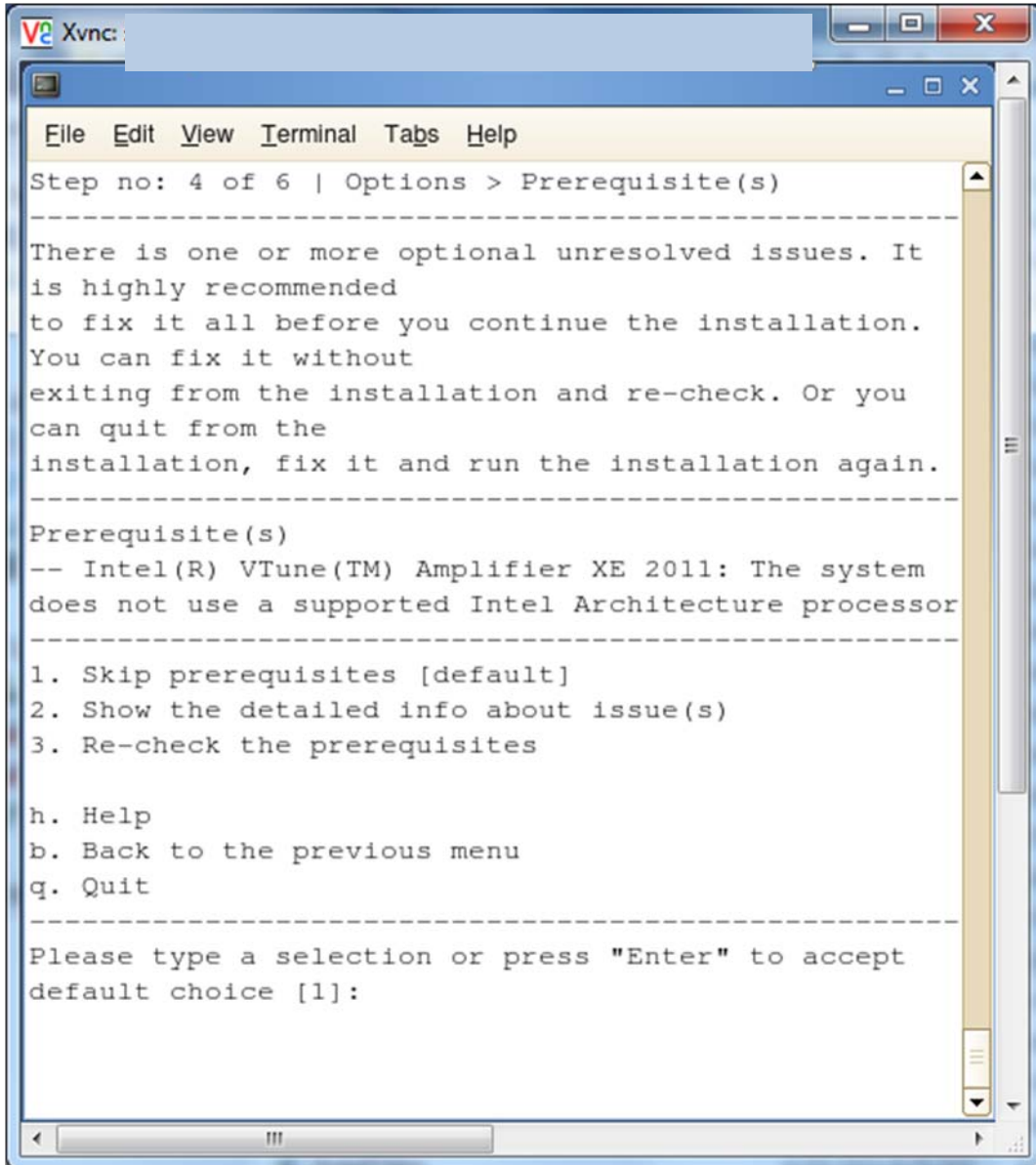


Figure 3.17 – Step 4 Continued – Let the install process proceed

Step 5 is the actual installation process. This is followed by step 6 which is the completion of the installation process.

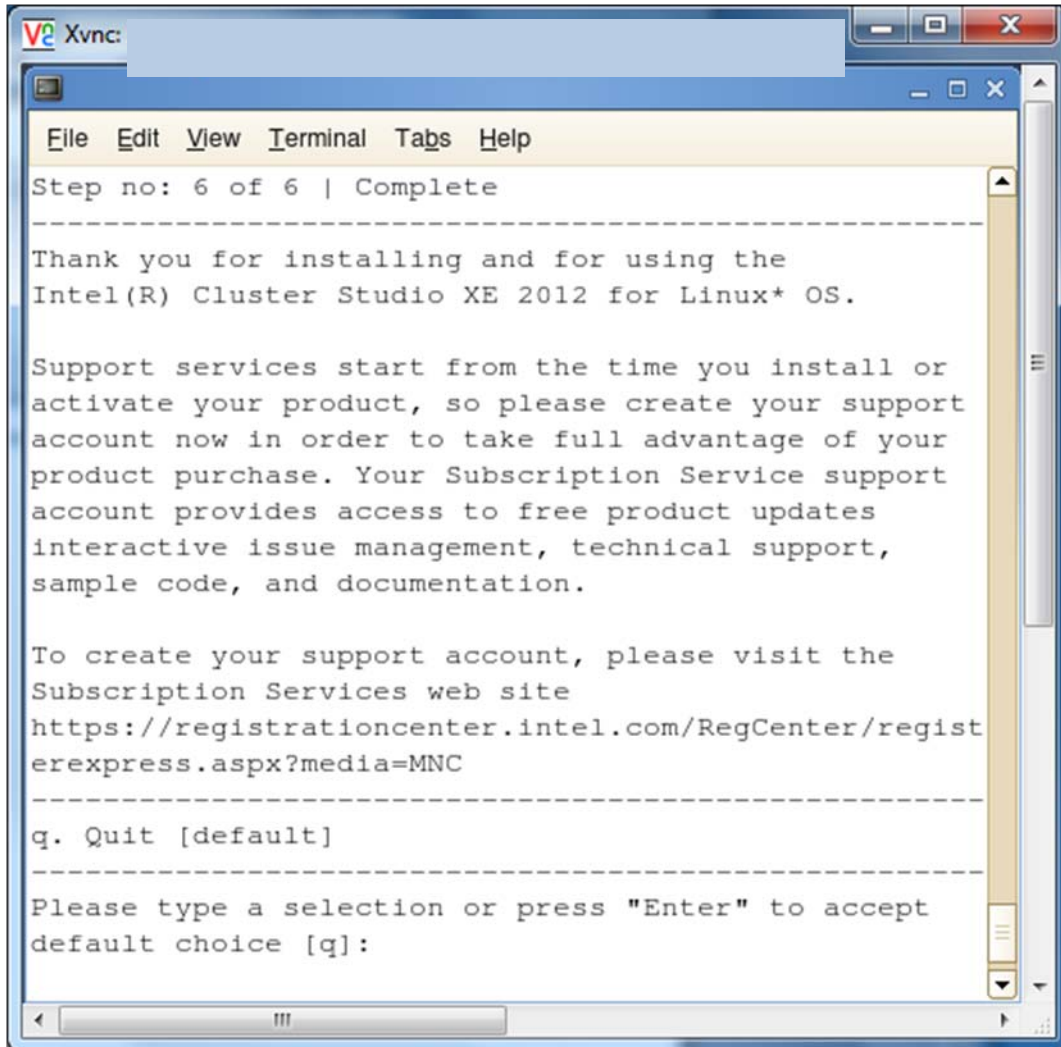


Figure 3.18 - Step 6 – The install process has completed and press the enter key to close the installer session

By default, the global root directory for the installation of the Intel Cluster Studio XE is:

```
/opt/intel/icsxe/<major>.<update>.<package_num>
```

where *<major>*, *<minor>*, *<update>*, and *<package_num>* are integers. An example would be 2012.0.037.

Within the folder path `/opt/intel/icsxe/<major>.<update>.<package_num>` you will find the text files:

```
ictvars.csh
```

```
ictvars.sh
```

and

```
icsxesupport.txt
```

If you are using Bourne Shell or Korn Shell for the login session, you should type:

```
.. ./ictvars.sh
```

and for a login session that uses C Shell, you should type:

```
source ./ictvars.csh
```

The file called:

```
icsxesupport.txt
```

contains the package ID and package contents information. Use the information in `icsxesupport.txt` when submitting customer support requests.

For the default installation path, an index file, an FAQ file, and the Getting Started Guide are located in the directory path:

```
/opt/intel/icsxe/<major>.<update>.<package_num>/doc
```

whereas mentioned above, `<major>`, `<update>`, and `<package_num>` are integers. A complete default folder path to the documentation directory might be:

```
/opt/intel/icsxe/2012.0.037/doc
```

The name of the index file is:

```
Doc_Index.htm
```

The index file can be used to navigate to the FAQ, the release notes, the Getting Started Guide, and an internet accessible [Intel Cluster Studio XE Tutorial](#). This web-based tutorial may have the latest information and instructions.

NOTE: For Beta programs involving the Intel Cluster Studio XE, there is no web based tutorial.

The documentation map file will also provide links to Intel® C++ Compiler XE documentation, Intel® Debugger Documentation, Intel® Fortran Compiler XE

documentation, Intel® Inspector XE documentation, Intel® Integrated Performance Primitives documentation, Intel® Math Kernel Library (MKL) documentation, Intel® MPI Library documentation, Intel® MPI Benchmarks documentation, Intel® Threading Building Blocks, Intel® Trace Analyzer and Collector documentation, and Intel® VTune™ Amplifier XE documentation. The content of the index file will look something like the following (Figure 3.19):

Document Name	File Name	Type of Information in the Document	Comment
Intel® Cluster Studio XE Getting Started Guide	Getting_Started.htm	Contains interoperability information about: <ul style="list-style-type: none"> • Intel® C++ Compiler XE 12.1 • Intel® Debugger 12.1 • Intel® Fortran Compiler XE 12.1 • Intel® Math Kernel Library (MKL) 10.3.6 • Intel® MPI Library 4.0 Update 3 • Intel® MPI Benchmarks 3.2.3 • Intel® Trace Analyzer and Collector 8.0 Update 3 • Intel® Inspector XE 2011 Update 6 • Intel® Integrated Performance Primitives 7.0 Update 5 • Intel® Threading Building Blocks 4.0 • Intel® VTune™ Amplifier XE 2011 Update 5 	For details about the Intel® Cluster Studio XE, please see the <i>Intel® Cluster Studio XE 2012 Getting Started Guide</i> .
Intel® Cluster Studio XE Release Notes	Release_Notes.htm	Contains information about this release and how to use the library with various compilers. Includes: <ul style="list-style-type: none"> • Overview • System Requirements • Installation • FAQ – Frequently Asked Questions • Technical Support and Feedback • Copyright and Legal Information 	For implementation-specific information about Intel® Cluster Studio XE, please see the <i>Intel® Cluster Studio XE 2012 Release Notes</i> .
Installer Guide for the Intel® Cluster Studio XE	Install.htm	Contains installer description for the Intel® Cluster Studio XE	This html page describes the process of doing an install of the Intel® Cluster Studio XE software.
Intel® Cluster HelpMe_FAQ	HelpMe_FAQ.htm	Contains a list of	This file may contain helpful

Figure 3.19 – A Rendering of the Intel Cluster Studio XE Documentation Index File display

The name of the FAQ file is:

HelpMe_FAQ.htm

The name of the Getting Started Guide file is:

Getting_Started.htm

By default, the local version of the release notes is located in the directory path:

`/opt/intel/icsxe/<major>.<update>.<package_num>/release_notes`

The name of the release notes file is:

Release_Notes.htm

[Back to Table of Contents](#)

4. Integrated Development Environments for Intel® Cluster Studio XE

For Linux* OS, there is an integrated development environment (IDE) by which you can develop software through Intel® Cluster Studio XE. This integrated development environment is Eclipse* for Intel® C++ Compiler XE.

If you are interested in using Eclipse*, install two software components that are not part of Intel® Cluster Studio XE. These two components are Eclipse*, and C/C++ Development Tooling* project (CDT*). CDT* provides an interface by which the Intel® C/C++ Compiler XE can be plugged into Eclipse*.

For further information about respectively downloading and installing Eclipse* and CDT* visit the URLs:

<http://www.eclipse.org/>

<http://www.eclipse.org/cdt/>

[Back to Table of Contents](#)

5. Getting Started with Intel® MPI Library

This chapter will provide some basic information about getting started with Intel® MPI Library. For complete documentation, see the Intel MPI Library documents *Intel MPI Library Getting Started Guide* located in `<directory-path-to-Intel-MPI-Library>/doc/Getting_Started.pdf` and *Intel MPI Library Reference Manual* located in `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf` on the system where Intel MPI Library is installed.

The software architecture for Intel MPI Library is described in Figure 5.1. With Intel MPI Library on Linux-based systems, you can choose the best interconnection fabric for running an application on a cluster that is based on IA-32, or Intel® 64 architecture. This is done at runtime by setting the `I_MPI_FABRICS` environment variable (See Section 5.4). Execution failure can be avoided even if interconnect selection fails. This feature helps avoid execution failures in batch computing. For such situations, the sockets interface will automatically be selected (Figure 5.1) as a backup.

Similarly using Intel MPI Library on Microsoft Windows CCS, you can choose the best interconnection fabric for running an application on a cluster that is based on Intel® 64 architecture.

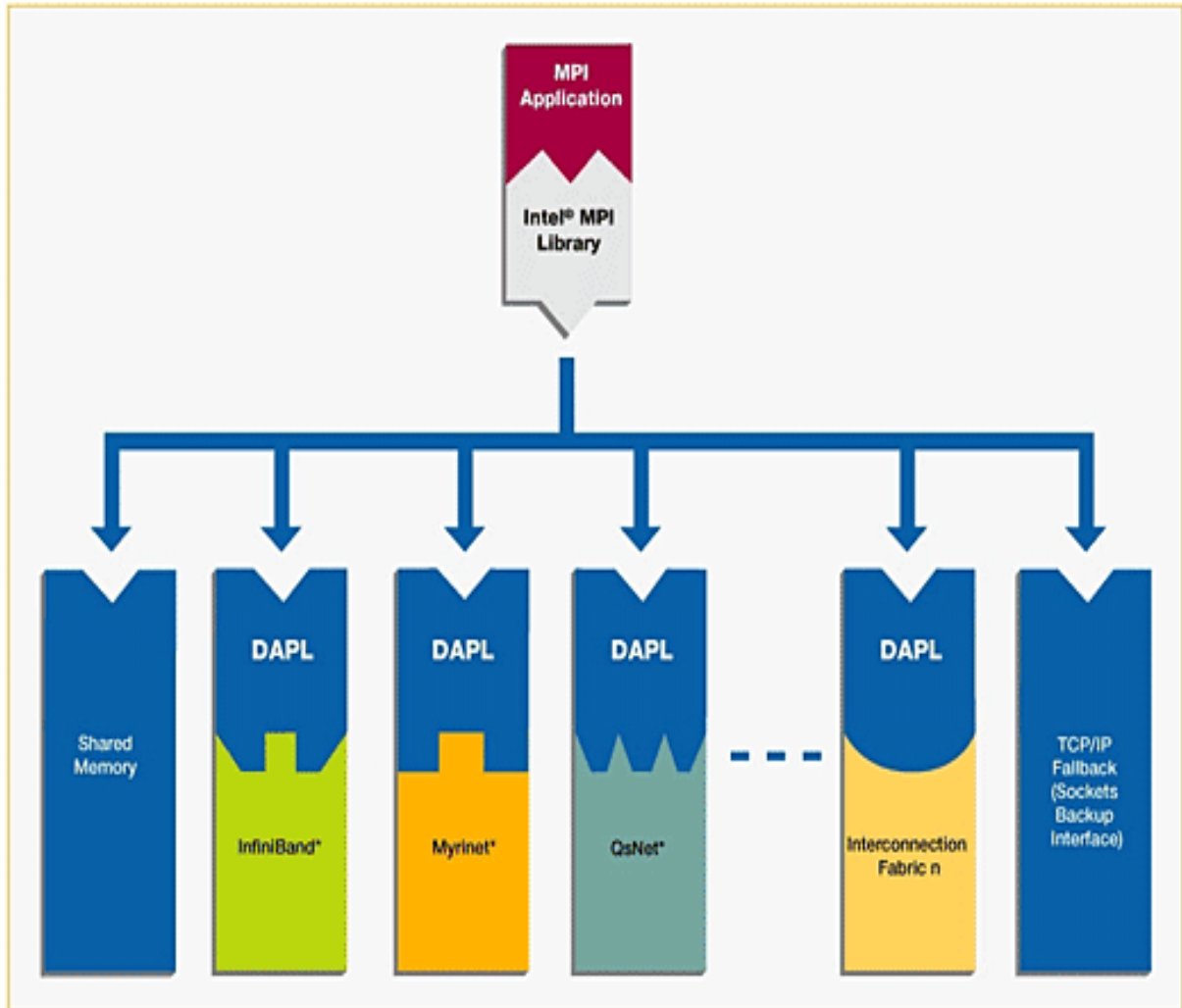


Figure 5.1 – Software architecture of the Intel® MPI Library Interface to Multiple Fast Interconnection Fabrics through shared memory, DAPL (Direct Access Programming Library), and the TCP/IP fallback

[Back to Table of Contents](#)

5.1 Launching MPD Daemons

The Intel MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. To run programs compiled with `mpicc` (or related) commands, you must first set up MPD daemons. It is strongly recommended that you start and maintain your own set of MPD daemons, as opposed to having the system administrator start up the MPD daemons once for use by all users on the system. This setup enhances system security and gives you greater flexibility in controlling your execution environment.

5.2 How to Set Up MPD Daemons on Linux* OS

1. Set up environment variables with appropriate values and directories, for example, in the `.cshrc` or `.bashrc` files. At a minimum, set the following environment variables. Ensure that the `PATH` variable includes the following:
 - The `<directory-path-to-Intel-MPI-Library>/bin` directory. For example, the `<directory-path-to-Intel-MPI-Library>/bin` directory path should be set.
 - Directory for Python* version 2.2 or greater.
 - If you are using Intel® C++ Compilers and/or Intel® Fortran Compilers, ensure that the `LD_LIBRARY_PATH` variable contains the directories for the compiler library. You can set this variable by using the `*vars.[c]sh` scripts included with the compiler. Set any additional environment variables your application uses.
2. Create a `$HOME/.mpd.conf` file that contains your MPD password. Your MPD password is not the same as any Linux login password, but rather is used for MPD only. It is an arbitrary password string that is used only to control access to the MPD daemons by various cluster users. To set up your MPD password:

```
secretword=<your mpd secretword>
```

Do not use any Linux login password for `<your mpd secretword>`. An arbitrary `<your mpd secretword>` string only controls access to the MPD daemons by various cluster users.

3. Set protection on the file so that you have read and write privileges, for example, and ensure that the `$HOME/.mpd.conf` file is visible on, or copied to, all the nodes in the cluster as follows:

```
chmod 600 $HOME/.mpd.conf
```

4. Verify that `PATH` settings and `.mpd.conf` contents can be observed through `ssh` on all nodes in the cluster. For example, use the following commands with each `<node>` in the cluster:

```
ssh <node> env
ssh <node> cat $HOME/.mpd.conf
```

5. Create an `mpd.hosts` text file that lists the nodes in the cluster, with one machine name per line, for use by `mpdboot`. Recall that the contents of the `machines.LINUX` file that was referenced previously can be used to construct an `mpd.hosts` file.
6. Start up the MPD daemons as follows:

```
mpdboot [ -d -v ] -n <#nodes> [-f <path/name of mpd.hosts file>]
```

For more information about the `mpdboot` command, see *Setting up MPD Daemons* in the `<directory-path-to-Intel-MPI-Library>/doc/Getting_Started.pdf` or the *mpdboot* section of `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf`.

7. Determine the status of the MPD daemons as follows:

```
mpdtrace
```

The output should be a list of nodes that are currently running MPD daemons.

Remarks

- *If required, shut down the MPD daemons as follows:*

```
mpdallexit
```

- *You as a user should start your own set of MPD daemons. It is not recommended to start MPD as root due to setup problems and security issues.*

[Back to Table of Contents](#)

5.3 The mpdboot Command for Linux* OS

Use the `mpdboot -f <hosts file>` option to select a specific hosts file to be used. The default is to use `${PWD}/mpd.hosts`. A valid host file must be accessible in order for `mpdboot` to succeed. As mentioned previously, you can also use the contents of the `machines.LINUX` file to construct an `mpd.hosts` file.

[Back to Table of Contents](#)

5.4 Compiling and Linking with Intel® MPI Library on Linux* OS

This section describes the basic steps required to compile and link an MPI program, when you use only the *Intel MPI Library Development Kit*. To compile and link an MPI program with the Intel MPI Library:

1. Ensure that the underlying compiler and related software appear in your `PATH`. If you are using Intel compilers, ensure that the compiler library directories appear in `LD_LIBRARY_PATH` environment variable. For example, regarding the Intel 12.1 compilers, the execution of the appropriate set-up scripts will do this automatically (the build number for the compilers might be something different than “`composer_xe_2011_sp1.6.061`” for your installation):

```
/opt/intel/composer_xe_2011_sp1.6.061/bin/iccvars.[c]sh
```

and

```
/opt/intel/composer_xe_2011_sp1.6.061/bin/ifortvars.[c]sh
```

2. Compile your MPI program through the appropriate `mpi` compiler command. For example, C code uses the `mpiicc` command as follows:

```
mpiicc <directory-path-to-Intel-MPI-Library>/test/test.c
```

Other supported compilers have an equivalent command that uses the prefix `mpi` on the standard compiler command. For example, the Intel MPI Library command for the Intel® Fortran Compiler (`ifort`) is `mpiifort`.

Supplier of Core Compiler	MPI Compilation Command	Core Compiler Compilation Command	Compiler Programming Language	Support Application Binary Interface (ABI)
GNU* Compilers	<code>mpicc</code>	<code>gcc, cc</code>	C	32/64 bit
	<code>mpicxx</code>	<code>g++ version 3.x</code> <code>g++ version 4.x</code>	C/C++	32/64 bit
	<code>mpif77</code>	<code>f77 or g77</code>	Fortran 77	32/64 bit
	<code>mpif90</code>	<code>gfortran</code>	Fortran 95	32/64 bit
Intel Compilers version 11.1, 12.0, or 12.1	<code>mpiicc</code>	<code>icc</code>	C	32/64 bit
	<code>mpiicpc</code>	<code>icpc</code>	C++	32/64 bit
	<code>mpiifort</code>	<code>ifort</code>	Fortran 77 and Fortran 95	32/64 bit

Remarks

The *Compiling and Linking* section of `<directory-path-to-Intel-MPI-Library>/doc/Getting_Started.pdf` or the *Compiler Commands* section of `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf` on the system where Intel MPI Library is installed *include additional details on `mpiicc` and other compiler commands, including commands for other compilers and languages.*

[Back to Table of Contents](#)

5.5 Selecting a Network Fabric

Intel MPI Library supports multiple, dynamically selectable network fabric device drivers to support different communication channels between MPI processes. The default communication method uses a built-in TCP (Ethernet, or sockets) device driver. Before the introduction of Intel® MPI Library 4.0, selection of alternative devices was done through the command line using the `I_MPI_DEVICE` environment variable. With Intel® MPI Library 4.0 and its successors, the `I_MPI_FABRICS` environment variable is to be used, and the environment variable `I_MPI_DEVICE` is considered a deprecated syntax. The following table lists the network fabric types for `I_MPI_FABRICS` that are supported by Intel MPI Library 4.0 and its successors:

Possible Interconnection-Device-Fabric Values for the I_MPI_FABRICS Environment Variable	Interconnection Device Fabric Meaning
shm	Shared-memory
dapl	DAPL-capable network fabrics, such as InfiniBand*, iWarp*, Dolphin*, and XPMEM* (through DAPL*)
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*)
tmi	Network fabrics with tag matching capabilities through the Tag Matching Interface (TMI), such as Qlogic* and Myrinet*
ofa	Network fabric, such as InfiniBand* (through OpenFabrics* Enterprise Distribution (OFED*) verbs) provided by the Open Fabrics Alliance* (OFA*)

The environment variable I_MPI_FABRICS has the following syntax:

```
I_MPI_FABRICS=<fabric> | <intra-node fabric>:<internodes-fabric>
```

where the *<fabric>* value meta-symbol can have the values shm, dapl, tcp, tmi, or ofa. The *<intra-node fabric>* value meta-symbol can have the values shm, dapl, tcp, tmi, or ofa. Finally, the *<inter-node fabric>* value meta-symbol can have the values dapl, tcp, tmi, or ofa.

The next section will provide some examples for using the I_MPI_FABRICS environment variable within the mpiexec command-line.

[Back to Table of Contents](#)

5.6 Running an MPI Program Using Intel® MPI Library on Linux* OS

Use the mpiexec command to launch programs linked with the Intel MPI Library example:

```
mpiexec -n <# of processes> ./myprog
```

The only required option for the mpiexec command is the -n option to set the number of processes. If your MPI application is using a network fabric other than the default fabric, use the -env option to specify a value to be assigned to the I_MPI_FABRICS variable. For example, to run an MPI program while using the shared

memory for intra-node communication and sockets for inter-node communication, use the following command:

```
mpiexec -n <# of processes> -env I_MPI_FABRICS shm:tcp ./myprog.exe
```

As an example of running an MPI application on a cluster system with a combined shared-memory and DAPL-enabled network fabric, the following `mpiexec` command-line might be used:

```
mpiexec -n <# of processes> -env I_MPI_FABRICS shm:dapl ./myprog.exe
```

See the section titled *Selecting a Network Fabric* in `<directory-path-to-Intel-MPI-Library>\doc\Getting_Started.pdf`, or the section titled *Fabrics Control* in `<directory-path-to-Intel-MPI-Library>\doc\Reference_Manual.pdf`.

[Back to Table of Contents](#)

5.7 Experimenting with Intel® MPI Library on Linux* OS

For the experiments that follow, it is assumed that a computing cluster has at least two nodes and there are two symmetric multi-processors (SMPs) per node. Start up the MPD daemons by issuing a command such as:

```
mpdboot -n 2 -r ssh -f ~/mpd.hosts
```

Type the command:

```
mpdtrace
```

to verify that there are MPD daemons running on the two nodes of the cluster. The response from issuing this command should be something like:

```
clusternode1  
clusternode2
```

assuming that the two nodes of the cluster are called `clusternode1` and `clusternode2`. The actual response will be a function of your cluster configuration.

In the `<directory-path-to-Intel-MPI-Library>/test` folder where Intel MPI Library resides, there are source files for four MPI test cases. In your local user area, you should create a test directory called:

```
test_intel_mpi/
```

From the installation directory of Intel MPI Library, copy the test files from `<directory-path-to-Intel-MPI-Library>/test` to the directory above. The contents of `test_intel_mpi` should now be:

```
test.c test.cpp test.f test.f90
```

Compile the test applications into executables using the following commands:

```
mpiifort test.f -o testf
mpiifort test.f90 -o testf90
mpiicc test.c -o testc
mpiicpc test.cpp -o testcpp
```

Issue the `mpiexec` commands:

```
mpiexec -n 2 ./testf
mpiexec -n 2 ./testf90
mpiexec -n 2 ./testc
mpiexec -n 2 ./testcpp
```

The output from `testcpp` should look something like:

```
Hello world: rank 0 of 2 running on clusternode1
Hello world: rank 1 of 2 running on clusternode2
```

If you have successfully run the above applications using Intel MPI Library, you can now run (without re-linking) the four executables on clusters that use Direct Access Program Library (DAPL) interfaces to alternative interconnection fabrics. If you encounter problems, please see the section titled *Troubleshooting* within the document *Intel MPI Library Getting Started Guide* located in `<directory-path-to-Intel-MPI-Library>/doc/Getting_Started.pdf` for possible solutions.

Assuming that you have a `dapl` device fabric installed on the cluster, you can issue the following commands for the four executables so as to access that device fabric:

```
mpiexec -env I_MPI_FABRICS dapl -n 2 ./testf
mpiexec -env I_MPI_FABRICS dapl -n 2 ./testf90
mpiexec -env I_MPI_FABRICS dapl -n 2 ./testc
mpiexec -env I_MPI_FABRICS dapl -n 2 ./testcpp
```

The output from `testf90` using the `dapl` device value for the `I_MPI_FABRICS` environment variable should look something like:

```
Hello world: rank          0  of          2  running on
  clusternode1

Hello world: rank          1  of          2  running on
  clusternode2
```

[Back to Table of Contents](#)

5.8 Controlling MPI Process Placement on Linux* OS

The `mpiexec` command controls how the ranks of the processes are allocated to the nodes in the cluster. By default, `mpiexec` uses round-robin assignment of ranks to the nodes. This placement algorithm may not be the best choice for your application, particularly for clusters with symmetric multi-processor (SMP) nodes.

Suppose that the geometry is `<#ranks> = 4` and `<#nodes> = 2`, where adjacent pairs of ranks are assigned to each node (for example, for 2-way SMP nodes). Issue the command:

```
cat ~/mpd.hosts
```

The results should be something like:

```
clusternode1
clusternode2
```

Since each node of the cluster is a 2-way SMP, and four processes are to be used for the application, the next experiment will distribute the four processes such that two of the processes will execute on `clusternode1` and two processes will execute on `clusternode2`. For example, you might issue the following commands:

```
mpiexec -n 2 -host clusternode1 ./testf : -n 2 -host clusternode2 ./testf
mpiexec -n 2 -host clusternode1 ./testf90 : -n 2 -host clusternode2 ./testf90
mpiexec -n 2 -host clusternode1 ./testc : -n 2 -host clusternode2 ./testc
mpiexec -n 2 -host clusternode1 ./testcpp : -n 2 -host clusternode2 ./testcpp
```

The following output should be produced for the executable `testc`:

```
Hello world: rank 0 of 4 running on clusternode1
Hello world: rank 1 of 4 running on clusternode1
Hello world: rank 2 of 4 running on clusternode2
Hello world: rank 3 of 4 running on clusternode2
```

In general, if there are i nodes in the cluster and each node is j -way SMP system, the `mpiexec` command-line syntax for distributing the i by j processes amongst the i by j processors within the cluster is:

```
mpiexec -n j -host <nodename-1> ./mpi_example : \  
        -n j -host <nodename-2> ./mpi_example : \  
        -n j -host <nodename-3> ./mpi_example : \  
        ...  
        -n j -host <nodename-i> ./mpi_example
```

NOTE: Fill in appropriate host names for `<nodename-1>` through `<nodename-i>` with respect to your cluster system. For a complete discussion on how to control

process placement through the `mpiexec` command, see the *Local Options* section of the *Intel MPI Library Reference Manual* located in `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf`.

[Back to Table of Contents](#)

5.9 Using the Automatic Tuning Utility Called *mpitune*

The `mpitune` utility was first introduced with Intel® MPI Library 3.2. It can be used to find optimal settings of Intel® MPI Library in regards to the cluster configuration or a user's application for that cluster.

As an example, the executables `testc`, `testcpp`, `testf`, and `testf90` in the directory `test_intel_mpi` could be used. The command invocation for `mpitune` might look something like the following:

```
mpitune --host-file machines.LINUX --output-file testc.conf --
        application \"mpiexec -n 4 testc\"
```

where the options above are just a subset of the following complete command-line switches:

Command-line Option	Semantic Meaning
<code>-a \"<app_cmd_line>\" --application \"<app_cmd_line>\"</code>	Switch on the application tuning mode. Quote the full command line as shown
<code>-cm --cluster-mode {exclusive full}</code>	Set the cluster usage mode exclusive – only one task will executed on the cluster at a time full – maximum number of tasks will be execute. This is the default mode
<code>-d --debug</code>	Print debug information
<code>-dl [d1[,d2...[,dN]]] --device-list [d1[,d2...[,dN]]]</code>	Select the device(s) you want to tune. By default use all of the devices mentioned in the <code><installdir>/<arch>/etc/devices.xml</code> file
<code>-er --existing-ring</code>	Try to use an existing MPD ring. By default, create a new MPD ring
<code>-fl [f1[,f2...[,fN]]] --fabric-list [f1[,f2...[,fN]]]</code>	Select the fabric(s) you want to tune. By default use all of the fabrics mentioned in the <code><installdir>/<arch>/etc/fabrics.xml</code> file
<code>-h --help</code>	Display a help message
<code>-hf <hostsfile> --host-file <hostsfile></code>	Specify an alternative host file name. By default, use the <code>\$PWD/mpd.hosts</code>

<code>-hr --host-range {min:max min: :max}</code>	Set the range of hosts used for testing. The default minimum value is 1. The default maximum value is the number of hosts defined by the <code>mpd.hosts</code> or the existing MPD ring. The <code>min:</code> or <code>:max</code> format will use the default values as appropriate
<code>-i <count> --iterations <count></code>	Define how many times to run each tuning step. Higher iteration counts increase the tuning time, but may also increase the accuracy of the results. The default value is 3
<code>-mh --master-host</code>	Dedicate a single host to <code>mpitune</code>
<code>--message-range {min:max min: :max}</code>	Set the message size range. The default minimum value is 0. The default maximum value is 4194304 (4mb). By default, the values are given in bytes. They can also be given in the following format: 16kb, 8mb, or 2gb. The <code>min:</code> or <code>:max</code> format will use the default values as appropriate
<code>-of <file-name> --output-file <file-name></code>	Specify the application configuration file to be generated in the application-specific mode. By default, use the <code>\$PWD/app.conf</code>
<code>-od <outputdir> --output-directory <outputdir></code>	Specify the directory name for all output files. By default, use the current directory. The directory should be accessible from all hosts
<code>-pr {min:max min: :max} / --ppn-range {min:max min: :max} / --perhost-range {min:max min: :max}</code>	Set the maximum number of processes per host. The default minimum value is 1. The default maximum value is the number of cores of the processor. The <code>min:</code> or <code>:max</code> format will use the default values as appropriate
<code>-sf [file-path] --session-file [file-path]</code>	Continue the tuning process starting from the state saved in the <code>file-path</code> session file
<code>-s --silent</code>	Suppress all diagnostic output
<code>-td <dir-path> --temp-directory <dir-path></code>	Specify a directory name for the temporary data. By default, use the <code>\$PWD/mpitunertemp</code> . This directory should be accessible from all hosts
<code>-t \"<test_cmd_line>\" --test \"<test_cmd_line>\"</code>	Replace the default Intel® MPI Benchmarks by the indicated benchmarking program in the cluster-specific mode. Quote the full command line as shown

<code>-tl <minutes> --time-limit <minutes></code>	Set <code>mpitune</code> execution time limit in minutes. The default value is 0, which means no limitations
<code>-V --version</code>	Print out the version information

Details on optimizing the settings for Intel® MPI Library with regards to the cluster configuration or a user's application for that cluster are described in the next two subsections.

[Back to Table of Contents](#)

5.9.1 Cluster Specific Tuning

Once you have installed the Intel® Cluster Tools on your system, you may want to use the `mpitune` utility to generate a configuration file that is targeted at optimizing the Intel® MPI Library with regards to the cluster configuration. For example, the `mpitune` command:

```
mpitune -hf machines.LINUX -of testc.conf --test \"testc\"
```

could be used, where `machines.LINUX` contains a list of the nodes in the cluster. Completion of this command may take some time. The `mpitune` utility will generate a configuration file that might have a name such as `app.conf`. You can then run the `mpiexec` command on an application using the `-tune` option. For example, the `mpiexec` command-line syntax for the `testc` executable might look something like the following:

```
mpiexec -tune -n 4 testc
```

[Back to Table of Contents](#)

5.9.2 MPI Application-Specific Tuning

The `mpitune` invocation:

```
mpitune -hf machines.Linux -of testf90.conf --application \"mpiexec -n 4 testf90\"
```

will generate a file called `app.config` that is base on the application `testf90`. Completion of this command may take some time also. This configuration file can be used in the following manner:

```
mpiexec -tune testf90.conf -n 4 testf90
```

where the `mpiexec` command will load the configuration options recorded in `testf90.conf`.

If you want to use `mpitune` utility on each of the test applications `testc`, `testcpp`, `testf`, and `testf90`, see the complete discussion on how to use the `mpitune` utility

in the *Intel MPI Library for Linux* OS Reference Manual* located in `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf`.

[Back to Table of Contents](#)

5.10 Extended File I/O System Support on Linux* OS

Intel® MPI Library provides loadable shared library modules to provide native support for the following file I/O systems:

- Panasas* ActiveScale* File System (PanFS)
- Parallel Virtual File System*, Version 2 (Pvfs2)

Set the `I_MPI_EXTRA_FILESYSTEM` environment variable to `on` to enable parallel file system support. Set the `I_MPI_EXTRA_FILESYSTEM_LIST` environment variable to request native support for the specific file system. For example, to request the native support for the Panasas* ActiveScale* File System, do the following:

```
mpiexec -env I_MPI_EXTRA_FILESYSTEM on -env I_MPI_EXTRA_FILESYSTEM_LIST panfs -n 4 ./myprog
```

[Back to Table of Contents](#)

5.10.1 How to Use the Environment Variables `I_MPI_EXTRA_FILESYSTEM` and `I_MPI_EXTRA_FILESYSTEM_LIST`

The environment variable `I_MPI_EXTRA_FILESYSTEM` is used to enable parallel I/O file system support. The general syntax for this environment variable is:

```
I_MPI_EXTRA_FILESYSTEM=<value>
```

where `<value>` can be:

Value	Meaning
enable or yes or on or 1	Turn on native support for a parallel file I/O system
disable or no or off or 0	Turn off native support for a parallel file I/O system. This is the default setting.

In conjunction with the `I_MPI_EXTRA_FILESYSTEM` environment variable, the environment variable `I_MPI_EXTRA_FILESYSTEM_LIST` will control which file I/O system or systems are used. In general, the syntax for the `I_MPI_EXTRA_FILESYSTEM_LIST` environment variable is:

I_MPI_EXTRA_FILESYSTEM_LIST=<file-system₁>[, <file-system₂>, <file-system₃>, ... , <file-system_n>]

where <file-system_i> can be:

File I/O System <file-system _i >	Meaning
panfs	Panasas* ActiveScale* File system
Pvfs2	Parallel Virtual File System, Version 2

The `mpiexec` and `mpirun` commands associated with Intel® MPI Library will load the shared I/O libraries associated with the `I_MPI_EXTRA_FILESYSTEM_LIST` environment variable. As mentioned previously, you must use the environment variables `I_MPI_EXTRA_FILESYSTEM` and `I_MPI_EXTRA_FILESYSTEM_LIST` together.

For a complete discussion on how to use the environment variables `I_MPI_EXTRA_FILESYSTEM` and `I_MPI_EXTRA_FILESYSTEM_LIST`, see the *Extended File System Support* section of the *Intel MPI Library for Linux* OS Reference Manual* located in <directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf.

To make inquiries about Intel MPI Library, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

6. Interoperability of Intel® MPI Library with the Intel® Debugger (IDB)

As mentioned previously (for example, Figure 2.1), components of the Intel Cluster Studio XE will now work with the Intel® Debugger. The Intel Debugger is a parallel debugger with the following software architecture (Figure 6.1):

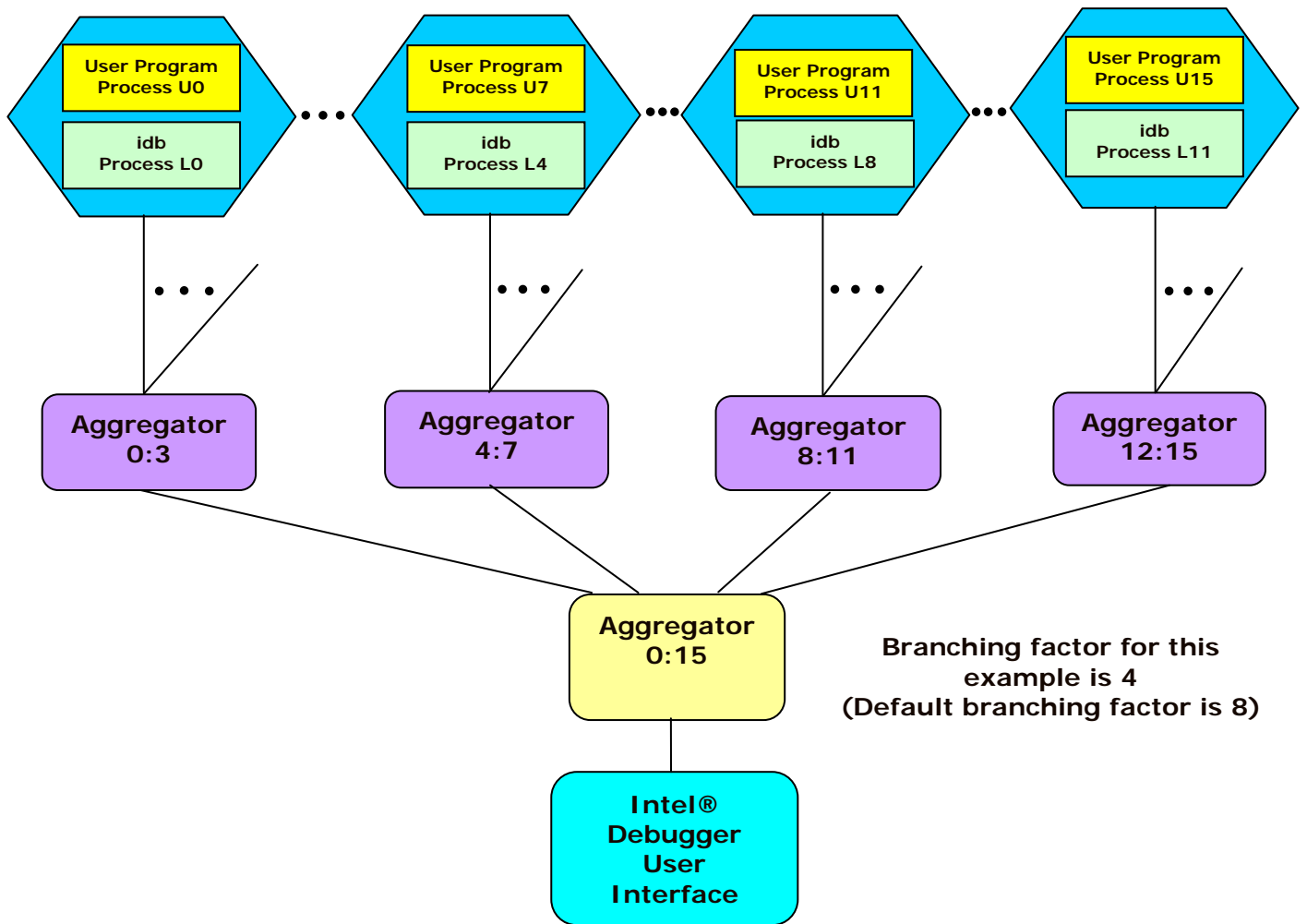


Figure 6.1 – The Software Architecture of the Intel Debugger

With respect to Figure 6.1, there is a user interface to a root debugger. This is demonstrated at the bottom of Figure 6.1. The root debugger communicates with a tree of parallel debuggers. These are the leaf nodes at the top of the illustration. There are aggregation capabilities for consolidating debug information. This is done through the aggregators in Figure 6.1.

All processes with the same output are aggregated into a single and final output message. For example, the following message represents 42 MPI processes:

```
[0-41] Linux Application Debugger for Xeon(R)-based applications,  
Version XX
```

Diagnostics which have different hexadecimal digits, but are otherwise identical, are condensed by aggregating the differing digits into a range. As an example:

```
[0-41]>2 0x120006d6c in  
feedback(myid=[0;41],np=42,name=0x11fffe018="mytest") "mytest.c":41
```

[Back to Table of Contents](#)

6.1 Login Session Preparations for Using Intel® Debugger on Linux* OS

The debugger executable for the Intel Debugger is called `idb`. In the 11.1 version of the Intel® Debugger, the `idb` command invokes the GUI. Alternatively for the 11.1 version of Intel® Debugger, to get the command-line interface, use `idbc`. You should follow three steps in preparing your login session to use the Intel Debugger.

1. The Intel® IDB Debugger graphical environment is a Java* application and requires a Java* Runtime Environment* (JRE*) to execute. The debugger will run with a version 5.0 (also called 1.5) JRE.

Install the JRE according to the JRE provider's instructions.

Finally you need to export the path to the JRE as follows:

```
export PATH=<path_to_JRE_bin_DIR>:$PATH export
```

2. Set the environment variable `IDB_HOME` to the folder path where the Intel Debugger executable, `idb`, resides. Also, you will need to source either `idbvars.sh` or `idbvars.csh` through `ifortvars.[c]sh` or `iccvars.[c]sh` depending on which command-line shell you are using. For example in augmenting your `.bashrc` file for the Bourne* Shell or the Korn* Shell, you can source the Intel® C++ Compiler XE file called `iccvars.sh` or the Intel® Fortran Compiler XE file `ifortvars.sh` which are located within the `bin` directory of the

Intel® Compiler XE installation directory on your system. Regarding your `.bashrc` file, the Bourne Shell or the Korn Shell sourcing syntax might look something like the following for Intel® 64 architecture:

```
. /opt/intel/composer_xe_2011_sp1.6.061/bin/iccvars.sh intel64
export IDB_HOME=/opt/intel/composer_xe_2011_sp1.6.061/bin/intel64
```

or

```
. /opt/intel/composer_xe_2011_sp1.6.061/bin/ifortvars.sh intel64
export IDB_HOME=/opt/intel/composer_xe_2011_sp1.6.061/bin/intel64
```

For augmenting your `.cshrc` file, the C Shell syntax should be something like:

```
source /opt/intel/composer_xe_2011_sp1.6.061/bin/iccvars.csh intel64
setenv IDB_HOME /opt/intel/composer_xe_2011_sp1.6.061/bin/intel64
```

or

```
source /opt/intel/composer_xe_2011_sp1.6.061/bin/ifortvars.csh intel64
setenv IDB_HOME /opt/intel/composer_xe_2011_sp1.6.061/bin/intel64
```

Depending on the Intel® architecture, the argument to `iccvars.[c]sh` and `ifortvars.[c]sh` can be `ia32`, or `intel64`. Sourcing `iccvars.[c]sh` or `ifortvars.[c]sh` will update the `PATH` and `MANPATH` environment variables also.

3. Edit the `~/.rhosts` file in your home directory so that it contains the list of nodes that comprise the cluster. Recall the contents of a file called `machines.LINUX`, where a contrived cluster consisting of eight nodes might be:

```
clusternode1
clusternode2
clusternode3
clusternode4
clusternode5
clusternode6
clusternode7
clusternode8
```

For example, assuming that the names listed above make up your cluster, they could be added to your `~/.rhosts` file with the following general syntax:

```
<hostname as echoed by the shell command hostname> <your username>
```

For the list of nodes above and assuming that your login name is `user01`, the contents of your `~/.rhosts` file might be:

```
clusternode1 user01
clusternode2 user01
clusternode3 user01
```

```
clusternode4 user01
clusternode5 user01
clusternode6 user01
clusternode7 user01
clusternode8 user01
```

The permission bit settings of `~/.rhosts` should be set to 600 using the `chmod` command. The shell command for doing this might be:

```
chmod 600 ~/.rhosts
```

Once you complete the three steps above, you are ready to use the Intel Debugger. The general syntax for using the Intel Debugger with Intel MPI Library is as follows:

```
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n <number of processes> [other
Intel MPI options] <executable> [arguments to the executable]
```

The environment variable `MPIEXEC_DEBUG` needs to be referenced so that MPI processes will suspend their execution to wait for the debuggers to attach to them. For the command-line example above, the `-genv` command-line option sets the environment variable `MPIEXEC_DEBUG` for *all* MPI processes. In general, the global environment variable command line switch `-genv` has the syntax:

```
-genv <environment variable> <value>
```

where *<environment variable>* is a meta-symbol that is a stand-in for a relevant environment variable, and *<value>* is a stand-in for setting an appropriate value for the preceding environment variable name.

For the contents of the directory `test_intel_mpi` that was described in Chapter 5, there should be the four source files:

```
test.c test.cpp test.f test.f90
```

Compile the test applications into executables using the following commands:

```
mpiifort -g test.f -o testf
mpiifort -g test.f90 -o testf90
mpiicc -g test.c -o testc
mpiicpc -g test.cpp -o testcpp
```

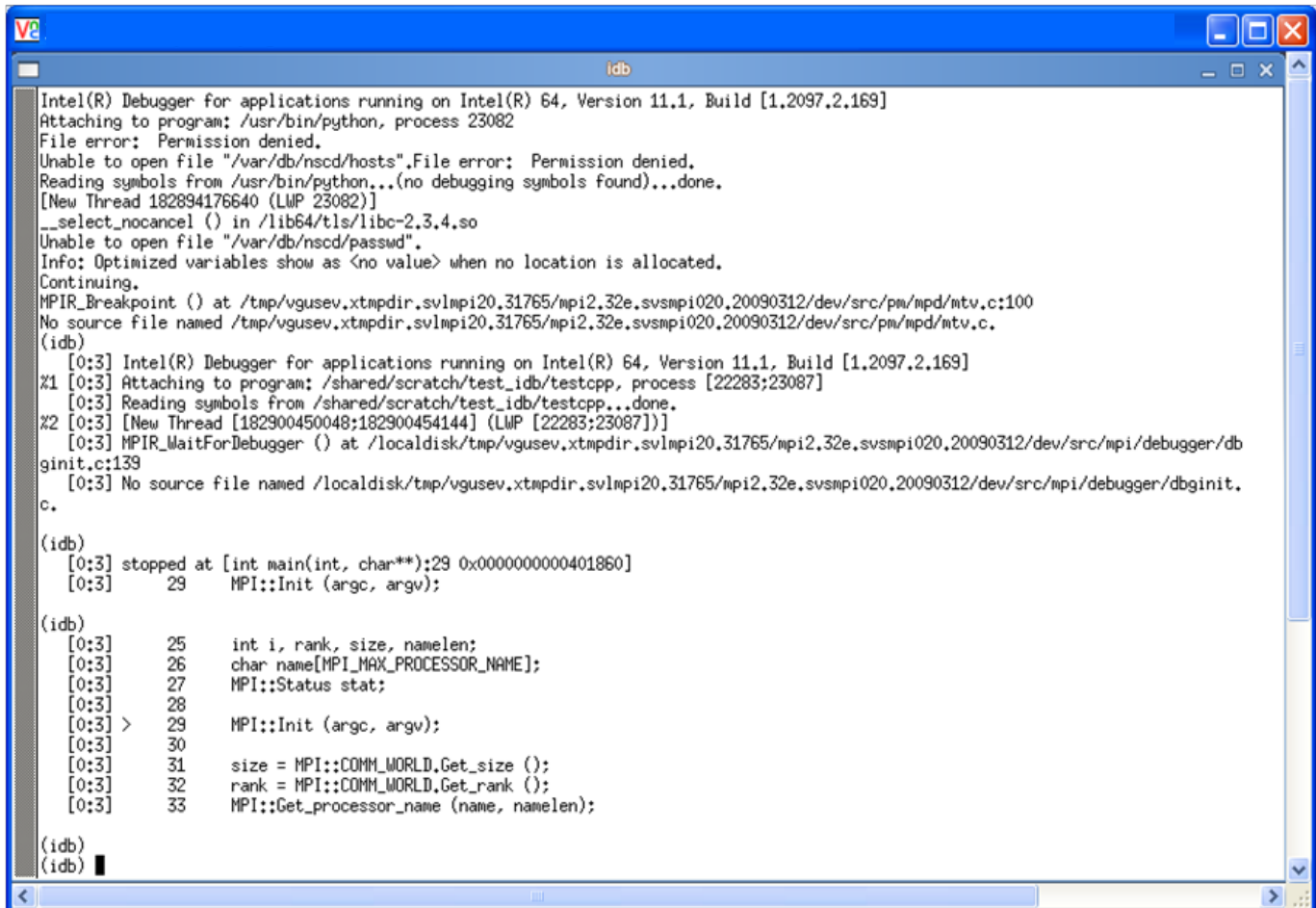
You can issue `mpiexec` commands that might look something like the following:

```
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n 4 ./testf
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n 4 ./testf90
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n 4 ./testc
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n 4 ./testcpp
```

The commands above are using four MPI processes. Figure 6.2 shows what the debug session might look like after issuing the shell command:

```
mpiexec -idb -genv MPIEXEC_DEBUG 1 -n 4 ./testcpp
```

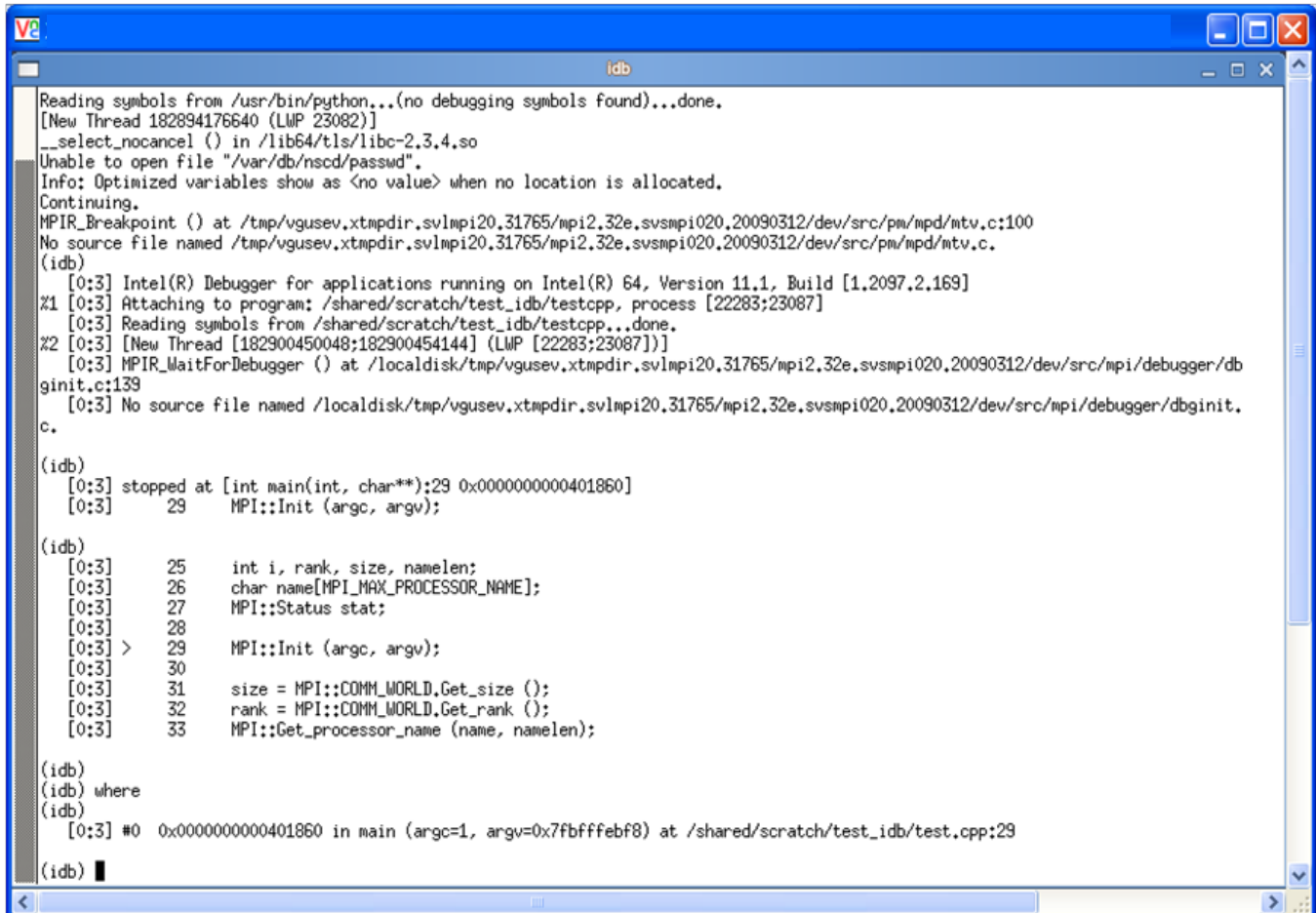
In Figure 6.2, the debugger stops the `testcpp` application at the C++ method `MPI::Init(argc, argv)`.



```
Intel(R) Debugger for applications running on Intel(R) 64, Version 11.1, Build [1.2097.2.169]
Attaching to program: /usr/bin/python, process 23082
File error: Permission denied.
Unable to open file "/var/db/nscd/hosts".File error: Permission denied.
Reading symbols from /usr/bin/python...(no debugging symbols found)...done.
[New Thread 182894176640 (LWP 23082)]
__select_nocancel () in /lib64/tls/libc-2.3.4.so
Unable to open file "/var/db/nscd/passwd".
Info: Optimized variables show as <no value> when no location is allocated.
Continuing.
MPIR_Breakpoint () at /tmp/vgusev.xtmpdir.svlmpi20,31765/mpi2,32e.svsmpi020,20090312/dev/src/pm/mpd/mtv.c:100
No source file named /tmp/vgusev.xtmpdir.svlmpi20,31765/mpi2,32e.svsmpi020,20090312/dev/src/pm/mpd/mtv.c.
(idb)
[0:3] Intel(R) Debugger for applications running on Intel(R) 64, Version 11.1, Build [1.2097.2.169]
x1 [0:3] Attaching to program: /shared/scratch/test_idb/testcpp, process [22283;23087]
[0:3] Reading symbols from /shared/scratch/test_idb/testcpp...done.
x2 [0:3] [New Thread [182900450048;182900454144] (LWP [22283;23087])]
[0:3] MPIR_WaitForDebugger () at /localdisk/tmp/vgusev.xtmpdir.svlmpi20,31765/mpi2,32e.svsmpi020,20090312/dev/src/mpi/debugger/db
ginit.c:139
[0:3] No source file named /localdisk/tmp/vgusev.xtmpdir.svlmpi20,31765/mpi2,32e.svsmpi020,20090312/dev/src/mpi/debugger/dbginit.
c.
(idb)
[0:3] stopped at [int main(int, char**):29 0x0000000000401860]
[0:3] 29 MPI::Init (argc, argv);
(idb)
[0:3] 25 int i, rank, size, namelen;
[0:3] 26 char name[MPI_MAX_PROCESSOR_NAME];
[0:3] 27 MPI::Status stat;
[0:3] 28
[0:3] > 29 MPI::Init (argc, argv);
[0:3] 30
[0:3] 31 size = MPI::COMM_WORLD.Get_size ();
[0:3] 32 rank = MPI::COMM_WORLD.Get_rank ();
[0:3] 33 MPI::Get_processor_name (name, namelen);
(idb)
(idb)
```

Figure 6.2 – idb session for the executable called testc

NOTE: The user interface for `idb` is `gdb*`-compatible by default. To see where the MPI application is with respect to execution, you can type the IDB command called `where` after the prompt (`idb`) in Figure 6.2. This will produce a call stack something like what is shown in Figure 6.3.



```
Reading symbols from /usr/bin/python...(no debugging symbols found)...done.
[New Thread 182894176640 (LWP 23082)]
__select_nocancel () in /lib64/tls/libc-2.3.4.so
Unable to open file "/var/db/nscd/passwd".
Info: Optimized variables show as <no value> when no location is allocated.
Continuing.
MPIR_Breakpoint () at /tmp/vgusev.xtmdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/pm/mpd/mtv.c:100
No source file named /tmp/vgusev.xtmdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/pm/mpd/mtv.c.
(idb)
[0:3] Intel(R) Debugger for applications running on Intel(R) 64, Version 11.1, Build [1.2097.2.169]
x1 [0:3] Attaching to program: /shared/scratch/test_idb/testcpp, process [22283;23087]
[0:3] Reading symbols from /shared/scratch/test_idb/testcpp...done.
x2 [0:3] [New Thread [182900450048;182900454144] (LWP [22283;23087])]
[0:3] MPIR_WaitForDebugger () at /localdisk/tmp/vgusev.xtmdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c:139
[0:3] No source file named /localdisk/tmp/vgusev.xtmdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c.
(idb)
[0:3] stopped at [int main(int, char**):29 0x000000000401860]
[0:3] 29 MPI::Init (argc, argv);
(idb)
[0:3] 25 int i, rank, size, namelen;
[0:3] 26 char name[MPI_MAX_PROCESSOR_NAME];
[0:3] 27 MPI::Status stat;
[0:3] 28
[0:3] > 29 MPI::Init (argc, argv);
[0:3] 30
[0:3] 31 size = MPI::COMM_WORLD.Get_size ();
[0:3] 32 rank = MPI::COMM_WORLD.Get_rank ();
[0:3] 33 MPI::Get_processor_name (name, namelen);
(idb)
(idb) where
(idb)
[0:3] #0 0x000000000401860 in main (argc=1, argv=0x7fbfffeb8) at /shared/scratch/test_idb/test.cpp:29
(idb) █
```

Figure 6.3 – The application call stack after typing the IDB command where

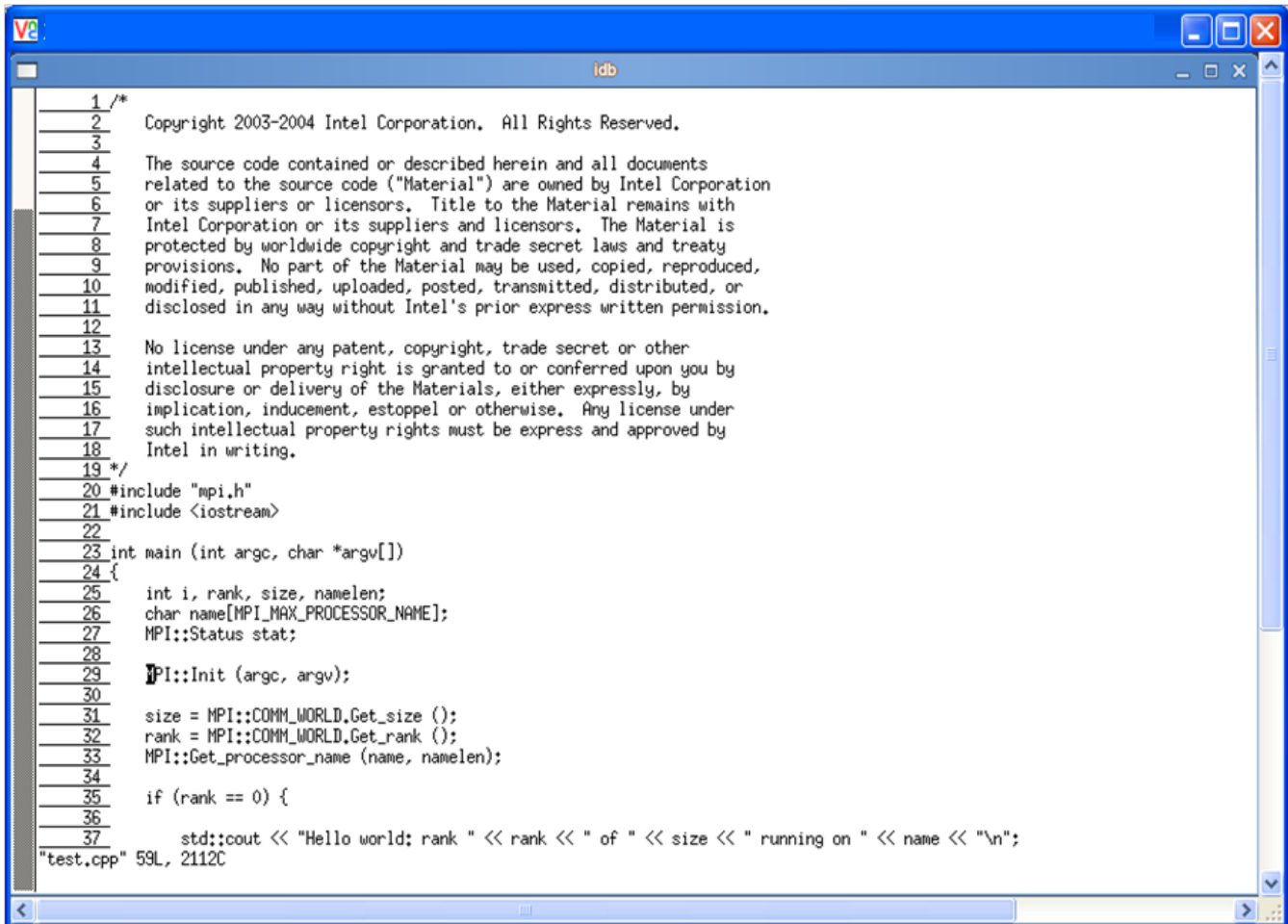
The C++ application has the source file name `test.cpp` and according to the IDB debugger stack trace, the line referenced in `test.cpp` is line 29. If you would like to use a text editor to look at `test.cpp`, you can modify the debugging user interface from the default which is `gdb*` to that of `idb` by typing the debug command:

```
set $cmdset = "idb"
```

You can then type the command:

```
edit +29 test.cpp
```

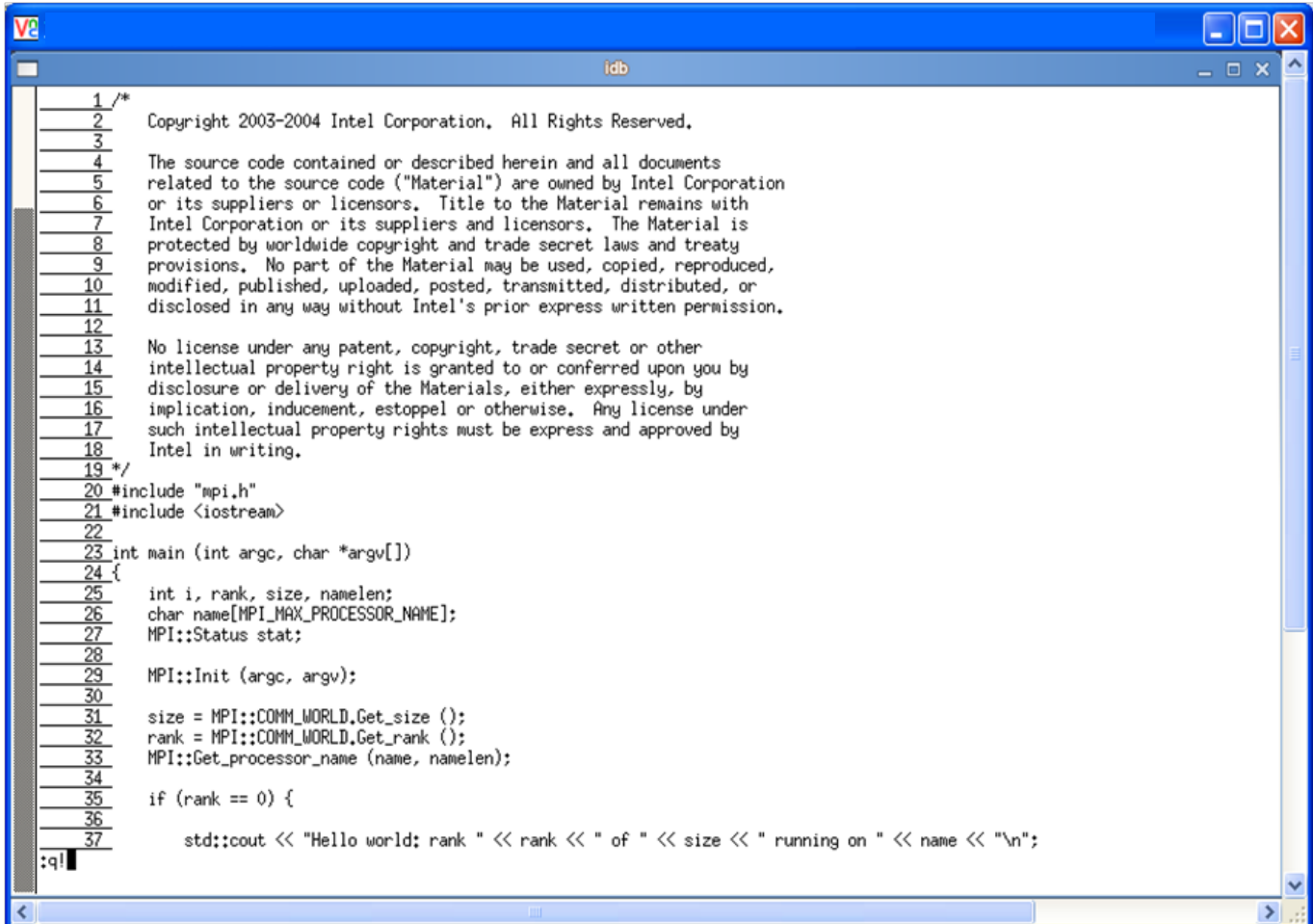
in Figure 6.3 and the result will be something like that shown in Figure 6.4. Line 29 of `test.cpp` is the MPI library call to `Init`. The edit session in Figure 6.4 is using the `vi` editor. In general, the editor that is invoked is a function of the `EDITOR` environment variable.



```
1 /*
2 Copyright 2003-2004 Intel Corporation. All Rights Reserved.
3
4 The source code contained or described herein and all documents
5 related to the source code ("Material") are owned by Intel Corporation
6 or its suppliers or licensors. Title to the Material remains with
7 Intel Corporation or its suppliers and licensors. The Material is
8 protected by worldwide copyright and trade secret laws and treaty
9 provisions. No part of the Material may be used, copied, reproduced,
10 modified, published, uploaded, posted, transmitted, distributed, or
11 disclosed in any way without Intel's prior express written permission.
12
13 No license under any patent, copyright, trade secret or other
14 intellectual property right is granted to or conferred upon you by
15 disclosure or delivery of the Materials, either expressly, by
16 implication, inducement, estoppel or otherwise. Any license under
17 such intellectual property rights must be express and approved by
18 Intel in writing.
19 */
20 #include "mpi.h"
21 #include <iostream>
22
23 int main (int argc, char *argv[])
24 {
25     int i, rank, size, namelen;
26     char name[MPI_MAX_PROCESSOR_NAME];
27     MPI::Status stat;
28
29     MPI::Init (argc, argv);
30
31     size = MPI::COMM_WORLD.Get_size ();
32     rank = MPI::COMM_WORLD.Get_rank ();
33     MPI::Get_processor_name (name, namelen);
34
35     if (rank == 0) {
36
37         std::cout << "Hello world: rank " << rank << " of " << size << " running on " << name << "\n";
"test.cpp" 59L, 2112C
```

Figure 6.4 – Launching of an edit session from the Intel Debugger

You can use the command `:q!` to close the `vi` edit session. This is demonstrated in Figure 5.5.

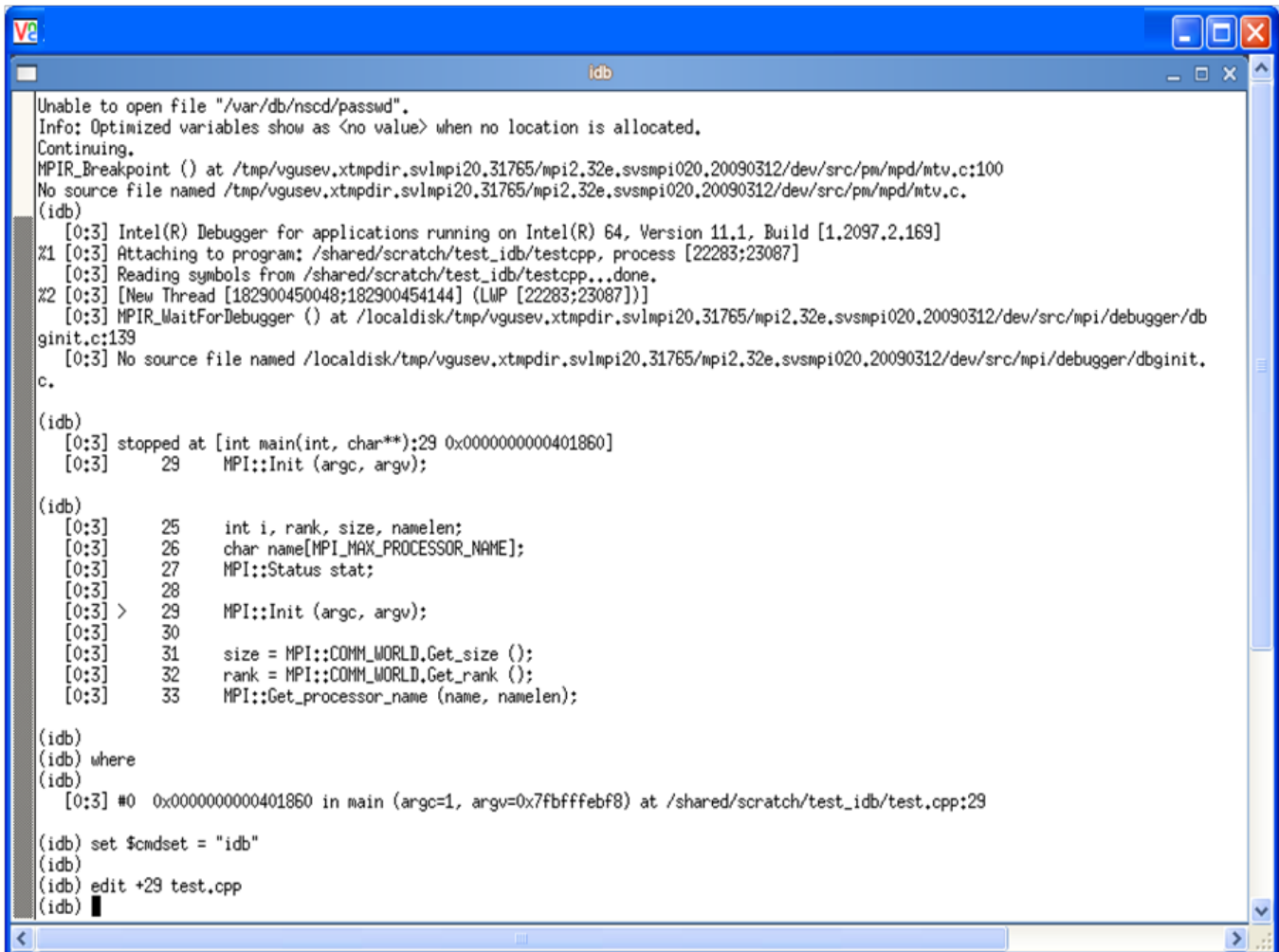


```
1 /*
2 Copyright 2003-2004 Intel Corporation. All Rights Reserved.
3
4 The source code contained or described herein and all documents
5 related to the source code ("Material") are owned by Intel Corporation
6 or its suppliers or licensors. Title to the Material remains with
7 Intel Corporation or its suppliers and licensors. The Material is
8 protected by worldwide copyright and trade secret laws and treaty
9 provisions. No part of the Material may be used, copied, reproduced,
10 modified, published, uploaded, posted, transmitted, distributed, or
11 disclosed in any way without Intel's prior express written permission.
12
13 No license under any patent, copyright, trade secret or other
14 intellectual property right is granted to or conferred upon you by
15 disclosure or delivery of the Materials, either expressly, by
16 implication, inducement, estoppel or otherwise. Any license under
17 such intellectual property rights must be express and approved by
18 Intel in writing.
19 */
20 #include "mpi.h"
21 #include <iostream>
22
23 int main (int argc, char *argv[])
24 {
25     int i, rank, size, namelen;
26     char name[MPI_MAX_PROCESSOR_NAME];
27     MPI::Status stat;
28
29     MPI::Init (argc, argv);
30
31     size = MPI::COMM_WORLD.Get_size ();
32     rank = MPI::COMM_WORLD.Get_rank ();
33     MPI::Get_processor_name (name, namelen);
34
35     if (rank == 0) {
36
37         std::cout << "Hello world: rank " << rank << " of " << size << " running on " << name << "\n";
38
39     }
40 }
```

:q!

Figure 6.5 – Terminating the vi editing session using the command :q!

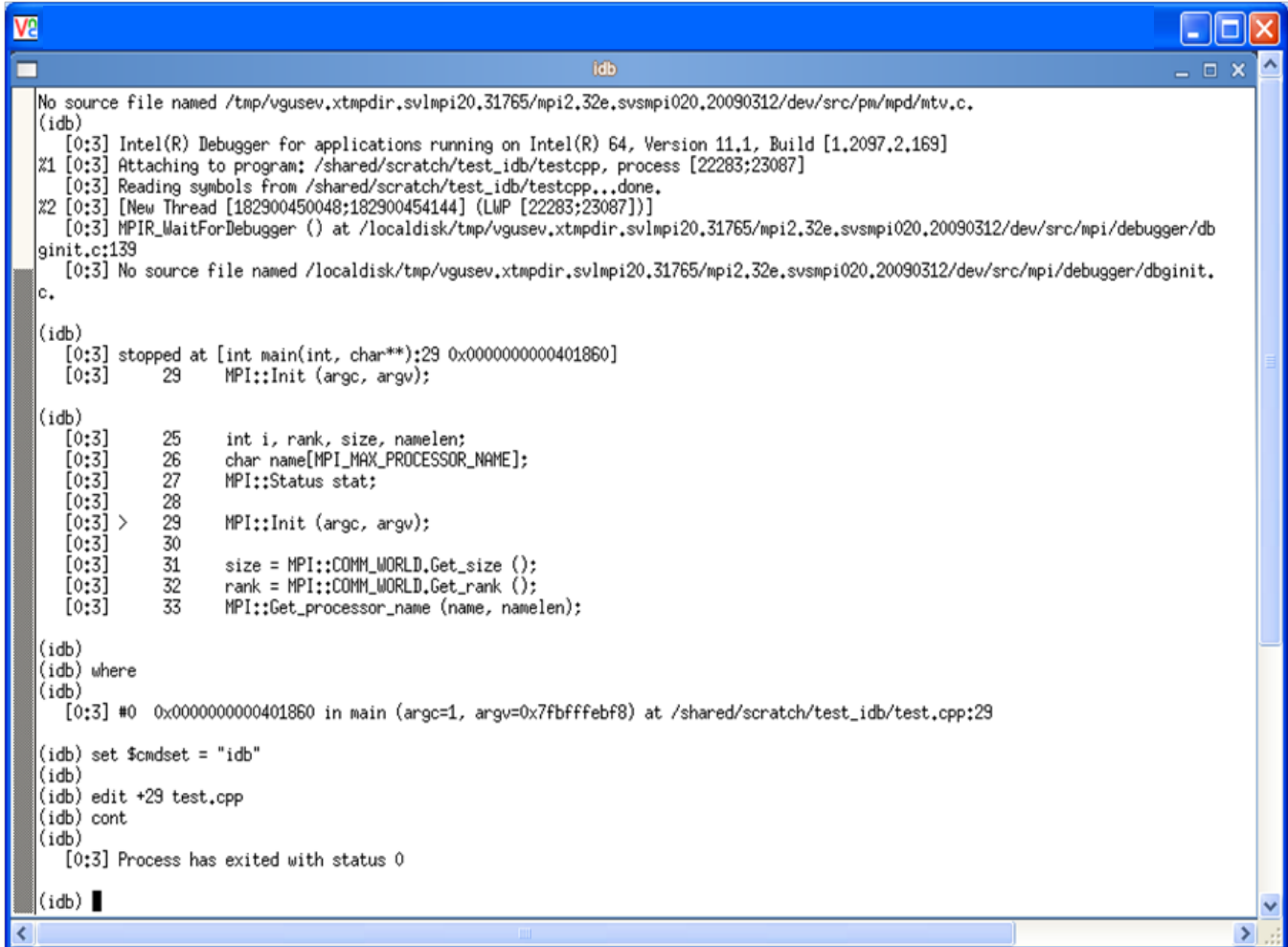
The "run" command is disabled in MPI debugging. To continue the execution of the MPI application, use "cont". If you proceed to type the word cont after the (ldb) prompt shown at the bottom of Figure 6.6, then debugging session results that might look something like that shown in Figure 6.7 will appear. Also, "Hello world" messages will appear in the login session where the mpiexec command was issued.



```
Unable to open file "/var/db/nscd/passwd".
Info: Optimized variables show as <no value> when no location is allocated.
Continuing.
MPIR_Breakpoint () at /tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/pm/mpd/mtv.c:100
No source file named /tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/pm/mpd/mtv.c.
(ldb)
[0:3] Intel(R) Debugger for applications running on Intel(R) 64, Version 11.1, Build [1.2097.2.169]
%1 [0:3] Attaching to program: /shared/scratch/test_idb/testcpp, process [22283;23087]
[0:3] Reading symbols from /shared/scratch/test_idb/testcpp...done.
%2 [0:3] [New Thread [182900450048;182900454144] (LWP [22283;23087])]
[0:3] MPIR_WaitForDebugger () at /localdisk/tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c:139
[0:3] No source file named /localdisk/tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c.
(ldb)
[0:3] stopped at [int main(int, char**):29 0x000000000401860]
[0:3] 29 MPI::Init (argc, argv);
(ldb)
[0:3] 25 int i, rank, size, namelen;
[0:3] 26 char name[MPI_MAX_PROCESSOR_NAME];
[0:3] 27 MPI::Status stat;
[0:3] 28
[0:3] > 29 MPI::Init (argc, argv);
[0:3] 30
[0:3] 31 size = MPI::COMM_WORLD.Get_size ();
[0:3] 32 rank = MPI::COMM_WORLD.Get_rank ();
[0:3] 33 MPI::Get_processor_name (name, namelen);
(ldb)
(ldb) where
(ldb)
[0:3] #0 0x000000000401860 in main (argc=1, argv=0x7fbfffeb8) at /shared/scratch/test_idb/test.cpp:29
(ldb) set $cmdset = "ldb"
(ldb)
(ldb) edit +29 test.cpp
(ldb) █
```

Figure 6.6 – Returning control back to IDB after terminating the editing session

The four MPI processes for the example in Figure 6.7 are labeled 0 to 3.



```
(idb) No source file named /tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/pm/mpd/mtv.c.
(idb) [0:3] Intel(R) Debugger for applications running on Intel(R) 64, Version 11.1, Build [1.2097.2.169]
%1 [0:3] Attaching to program: /shared/scratch/test_idb/testcpp, process [22283;23087]
[0:3] Reading symbols from /shared/scratch/test_idb/testcpp...done.
%2 [0:3] [New Thread [182900450048;182900454144] (LMP [22283;23087])]
[0:3] MPIR_WaitForDebugger () at /localdisk/tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c:139
[0:3] No source file named /localdisk/tmp/vgusev.xtmpdir.svlmpi20.31765/mpi2.32e.svsmpi020.20090312/dev/src/mpi/debugger/dbginit.c.
(idb) [0:3] stopped at [int main(int, char**):29 0x000000000401860]
[0:3] 29 MPI::Init (argc, argv);
(idb) [0:3] 25 int i, rank, size, namelen;
[0:3] 26 char name[MPI_MAX_PROCESSOR_NAME];
[0:3] 27 MPI::Status stat;
[0:3] 28
[0:3] > 29 MPI::Init (argc, argv);
[0:3] 30
[0:3] 31 size = MPI::COMM_WORLD.Get_size ();
[0:3] 32 rank = MPI::COMM_WORLD.Get_rank ();
[0:3] 33 MPI::Get_processor_name (name, namelen);
(idb) (idb) where
(idb) [0:3] #0 0x000000000401860 in main (argc=1, argv=0x7fbffffebf8) at /shared/scratch/test_idb/test.cpp:29
(idb) set $cmdset = "idb"
(idb) (idb) edit +29 test.cpp
(idb) (idb) cont
(idb) [0:3] Process has exited with status 0
(idb) █
```


**Figure 6.7 – State of the IDB session as a result of issuing the IDB command
cont**

You can type the word `quit` to end the IDB debug session, and therefore close the display shown in Figure 6.7.

The `rerun` command is not supported within IDB. To rerun MPI application with the IDB debugger, quit IDB and then re-enter the `mpirexec` command.

For a complete discussion on how to use the Intel Debugger (9.1.x or greater), see the contents of the *Intel Debugger (IDB) Manual* located in `<directory-path-to-Intel-composerxe>/Documentation/en_US/debugger/debugger_documentation.htm` on your computing system.

To make inquiries about the Intel Debugger, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

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

In the folder path where Intel® Trace Analyzer and Collector reside, there is a folder called `examples`. The folder path where the examples directory resides might be something like:

```
/opt/intel/icsxe/2012.0.037/itac/examples
```

If you copy the `examples` folder into a work area which is accessible by all of the nodes of the cluster, you might try the following sequence of commands:

```
gmake distclean
gmake all
```

This set of commands will respectively clean up the folder content and compile and execute the following C and Fortran executables:

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

If you select the executable `vtjacobic` and run it with the following environment variable setting:

```
setenv VT_LOGFILE_PREFIX vtjacobic_inst
```

where the `mpiexec` command uses four processes as shown:

```
mpiexec -n 4 ./vtjacobic
```

then the trace data will be placed into the folder `vtjacobic_inst`. The contents of `vtjacobic_inst` will look something like the following:

```
.          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
```

when the command:

```
ls -aC --width=80 vtjacobic_inst
```

is used. If you run the Intel Trace Analyzer with the command:

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

the following display panel will appear (Figure 7.1):

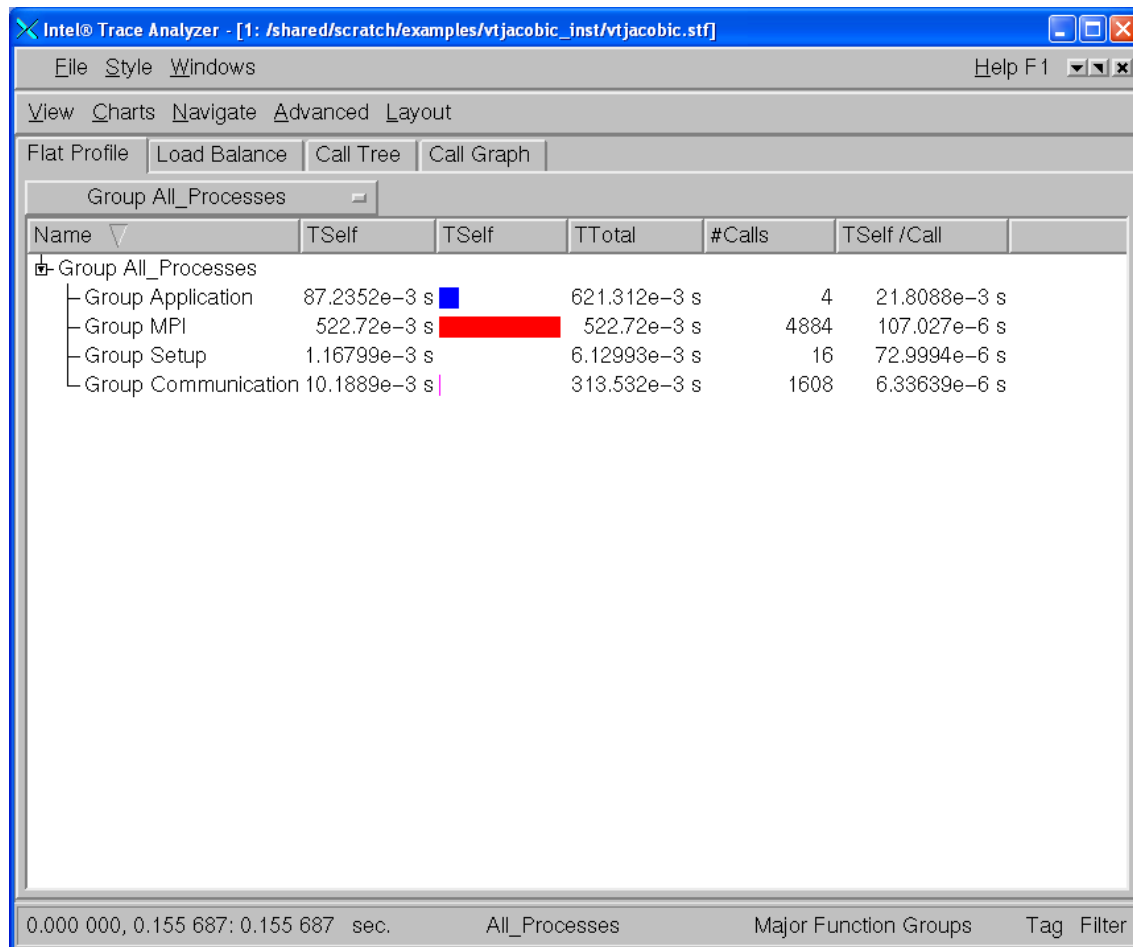


Figure 7.1 - Intel Trace Analyzer Display for vtjacobic.stf

Figure 7.2 shows the Event Timeline display which results when following the menu path **Charts->Event Timeline** within Figure 7.1.

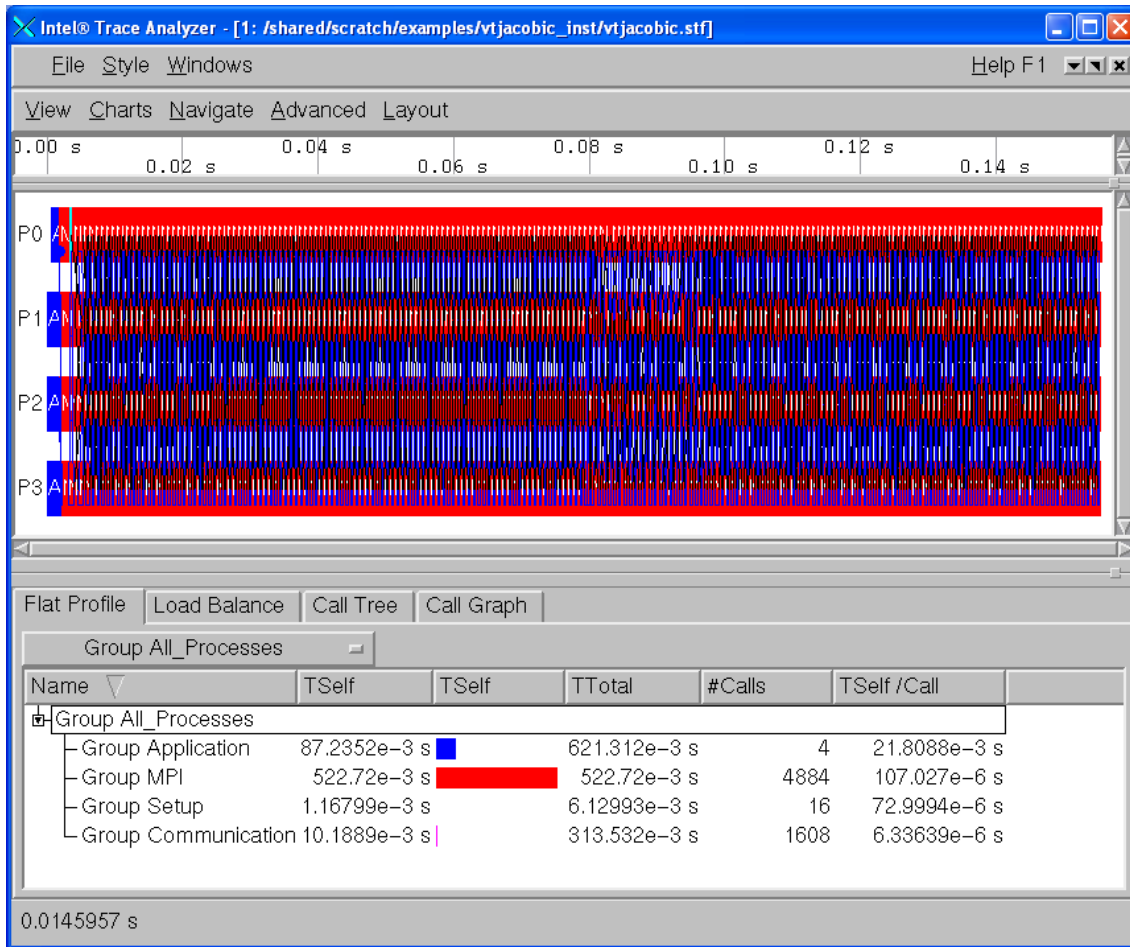


Figure 7.2 - Intel Trace Analyzer Display for vtjacobic.stf using Charts->Event Timeline

You can use the trace analyzer to view the contents of the other *.stf files in this working directory on your cluster system.

[Back to Table of Contents](#)

7.1 Experimenting with Intel® Trace Analyzer and Collector in a Fail-Safe Mode

There may be situations where an application will end prematurely; thus trace data could be lost. The Intel Trace Collector has a trace library that works in a fail-safe mode. An example shell command-line syntax for linking such a library is:

```
mpicc test.c -o testc_fs -L${VT_LIB_DIR} -lVTfs ${VT_ADD_LIBS}
```

where the special Intel Trace Collector Library for fail-safe (acronym `fs`) tracing is -
`lvtfs`.

In case of execution failure by the application, the fail-safe library freezes all MPI
processes and then writes out the trace file. Figure 7.3 shows an Intel Trace Analyzer
display for `test.c`.

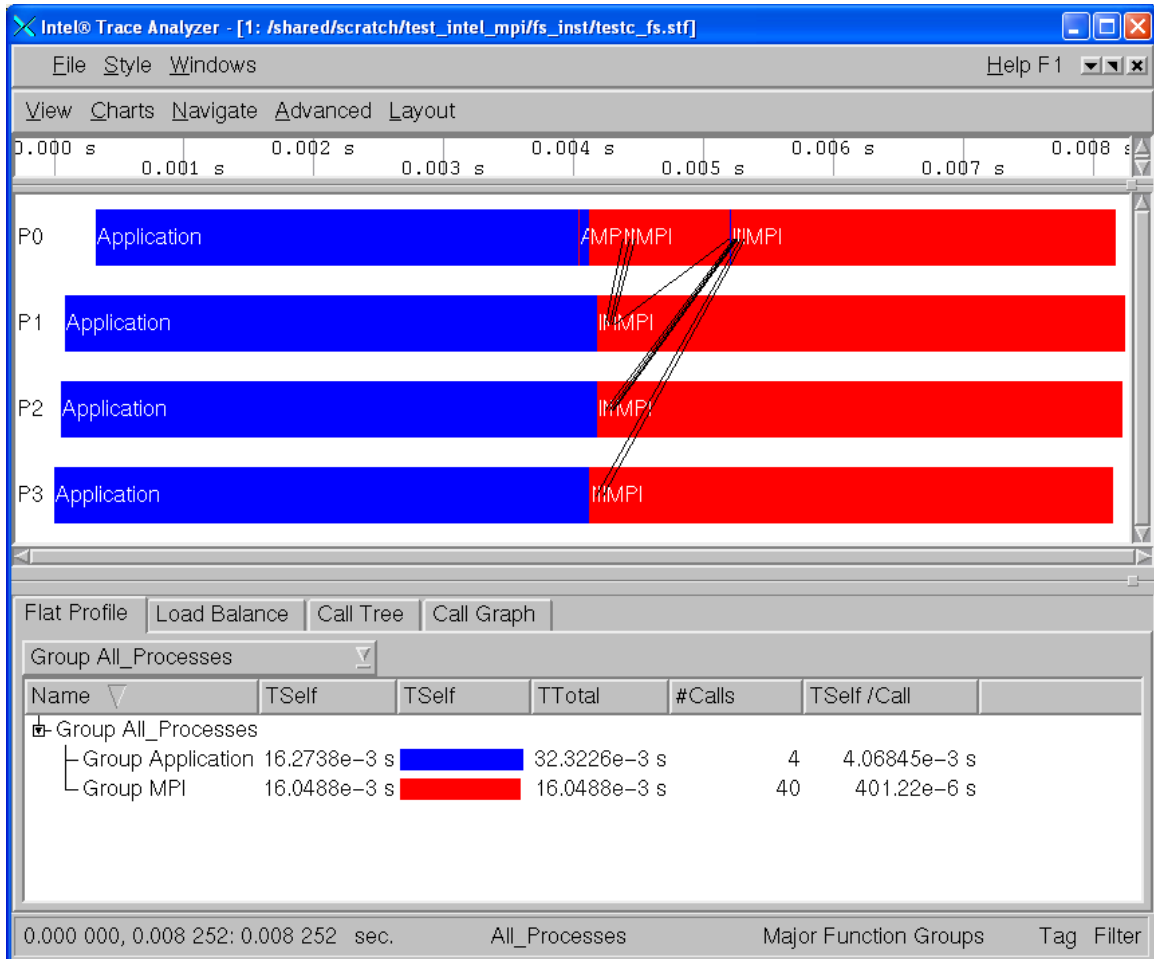


Figure 7.3 – Intel Trace Analyzer display of Fail-Safe Trace Collection by Intel Trace Collector

Regarding `-lvtfs` library, see the Intel Trace Collector user documentation by
viewing the file:

`<directory-path-to-ITAC>/doc/ITC_Reference_Guide.pdf`

on the system where the Intel Trace Collector is installed. You can use `vtf` as a
search phrase within the documentation.

[Back to Table of Contents](#)

7.2 Using itcpin to Instrument an Application

The `itcpin` utility is a binary instrumentation tool that comes with Intel Trace Analyzer and Collector. The Intel® architectures must be IA-32, or Intel® 64.

The basic syntax for instrumenting a binary executable with the `itcpin` utility is as follows:

```
itcpin [<ITC options>] -- <application command line>
```

where `--` is a delimiter between Intel® Trace Collector (ITC) options and the application command-line.

The *<ITC options>* that will be used are:

`--run (off)`

`itcpin` only runs the given executable if this option is used. Otherwise it just analyzes the executable and prints configurable information about it.

`--insert`

Intel Trace Collector has several libraries that can be used to do different kinds of tracing. An example library value could be `VT` which is the Intel Trace Collector Library. This is the default instrumentation library.

To obtain a list of all of the options, type:

```
itcpin --help
```

To demonstrate the use of `itcpin`, you can compile a C programming language example for calculating the value of `pi` where the application uses the MPI parallel programming paradigm. You can download the C source from the URL:

<http://www.nccs.gov/wp-content/training/mpi-examples/C/pical.c>

For the `pi.c` example, the following shell commands will allow you to instrument the binary called `pi.exe` with Intel Trace Collector instrumentation. The shell commands before and after the invocation of `itcpin` should be thought of as prolog and epilog code to aid in the use of the `itcpin` utility.

```
mpiicc -o pi.exe pi.c
setenv VT_LOGFILE_FORMAT STF
setenv VT_PCTRACE 5
setenv VT_LOGFILE_PREFIX ${PWD}/itcpin_inst
setenv VT_PROCESS "0:N ON"
```

```

rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
mpiexec -n 4 itcpin --run -- pi.exe

```

The shell commands above could be packaged into a C Shell script. An explanation for the *instrumentation* environment variables can be found in the Intel Trace Collector Users' Guide under the search topic "ITC Configuration".

Figure 7.4 shows the timeline and function panel displays that are generated from the instrumentation data that is stored into the directory `${PWD}/itcpin_inst` as indicated by the environment variable `VT_LOGFILE_PREFIX`. The command that initiated the Intel Trace Analyzer with respect to the directory `${PWD}` is:

```

traceanalyzer itcpin_inst/pi.exe.stf &

```

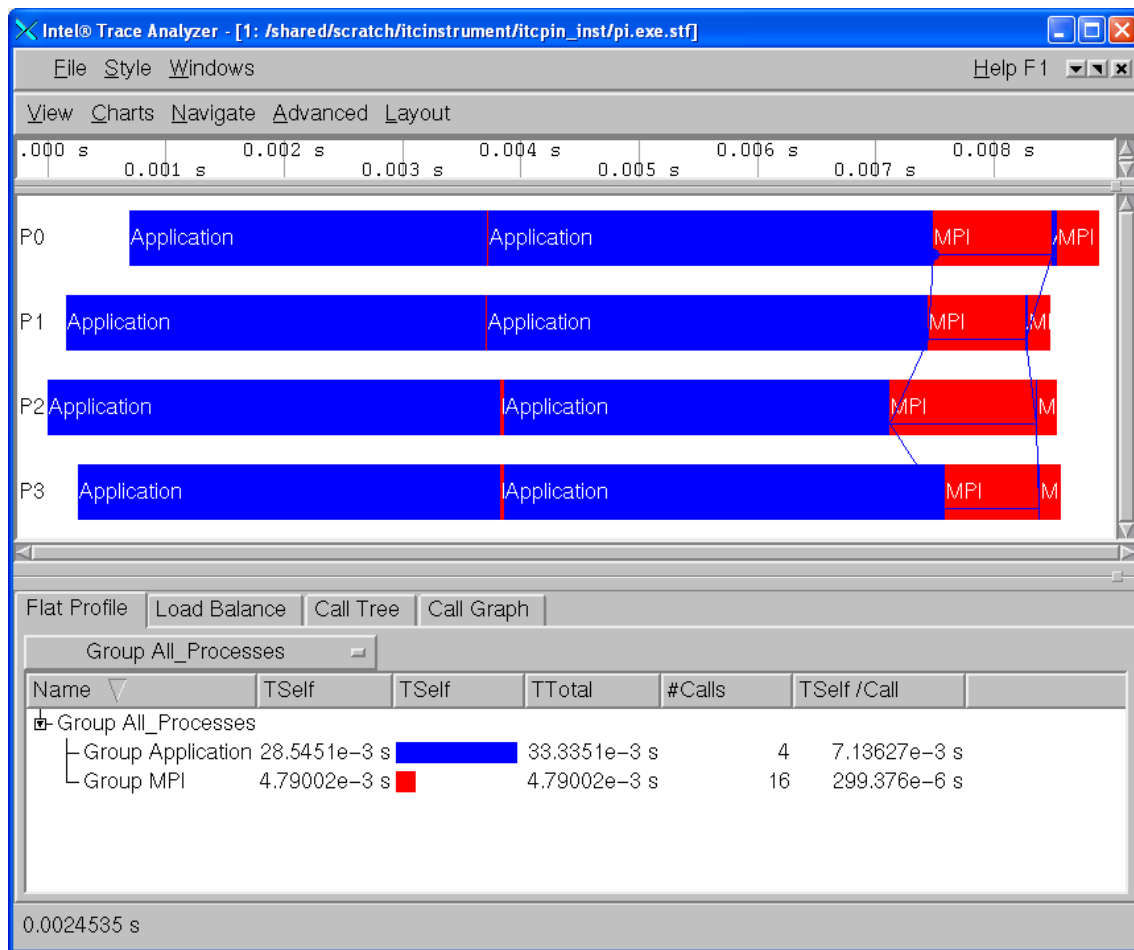


Figure 7.4 – Intel Trace Analyzer display of the “pi” integration application that has been binary instrumented with `itcpin`

Complete user documentation regarding the `itcpin` utility for the Intel Trace Collector can be found within the file:

```
<directory-path-to-ITAC>/doc/ITC_Reference_Guide.pdf
```

on the system where the Intel Trace Collector is installed. You can use `itcpin` as a search phrase within the documentation. To make inquiries about the Intel Trace Analyzer, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

7.3 Experimenting with Intel® Trace Analyzer and Collector in Conjunction with the LD_PRELOAD Environment Variable

There is an environment variable called `LD_PRELOAD` which can be initialized to reference instrumentation libraries. `LD_PRELOAD` instructs the operating system loader to load additional libraries into a program, beyond what was specified when it was initially compiled. In general, this environment variable allows users to add or replace functionality such as inserting performance tuning instrumentation. For Bourne* Shell or Korn* Shell, the syntax for setting the `LD_PRELOAD` environment variable to instrument with Intel Trace Collector might be:

```
export LD_PRELOAD="libVT.so:libddl.so"
```

For C Shell, the syntax might be:

```
setenv LD_PRELOAD "libVT.so:libddl.so"
```

For the `pi.c` example, the following shell commands will allow you to use the `LD_PRELOAD` environment variable to instrument a binary with Intel Trace Collector instrumentation.

```
mpiicc -o pi.exe pi.c
setenv VT_PCTRACE 5
setenv VT_LOGFILE_PREFIX ${PWD}/ld_preload_inst
setenv VT_PROCESS "0:N ON"
setenv LD_PRELOAD "libVT.so:libddl.so"
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
mpiexec -n 4 ./pi.exe 1000000
```

As mentioned previously, the shell commands above could be packaged into a C Shell script. The `mpiexec` command uses four MPI processes and the value of 1,000,000 indicates the number of intervals that will be used in the calculation of "pi". Figure 7.5 shows the timeline and function panel displays that are generated from the instrumentation data that was stored in the directory

`/${PWD}/ld_preload_inst` as indicated by the environment variable `VT_LOGFILE_PREFIX`. The command that initiated the Intel Trace Analyzer with respect to the directory `/${PWD}` is:

```
traceanalyzer ld_preload_inst/pi.exe.instr.stf &
```

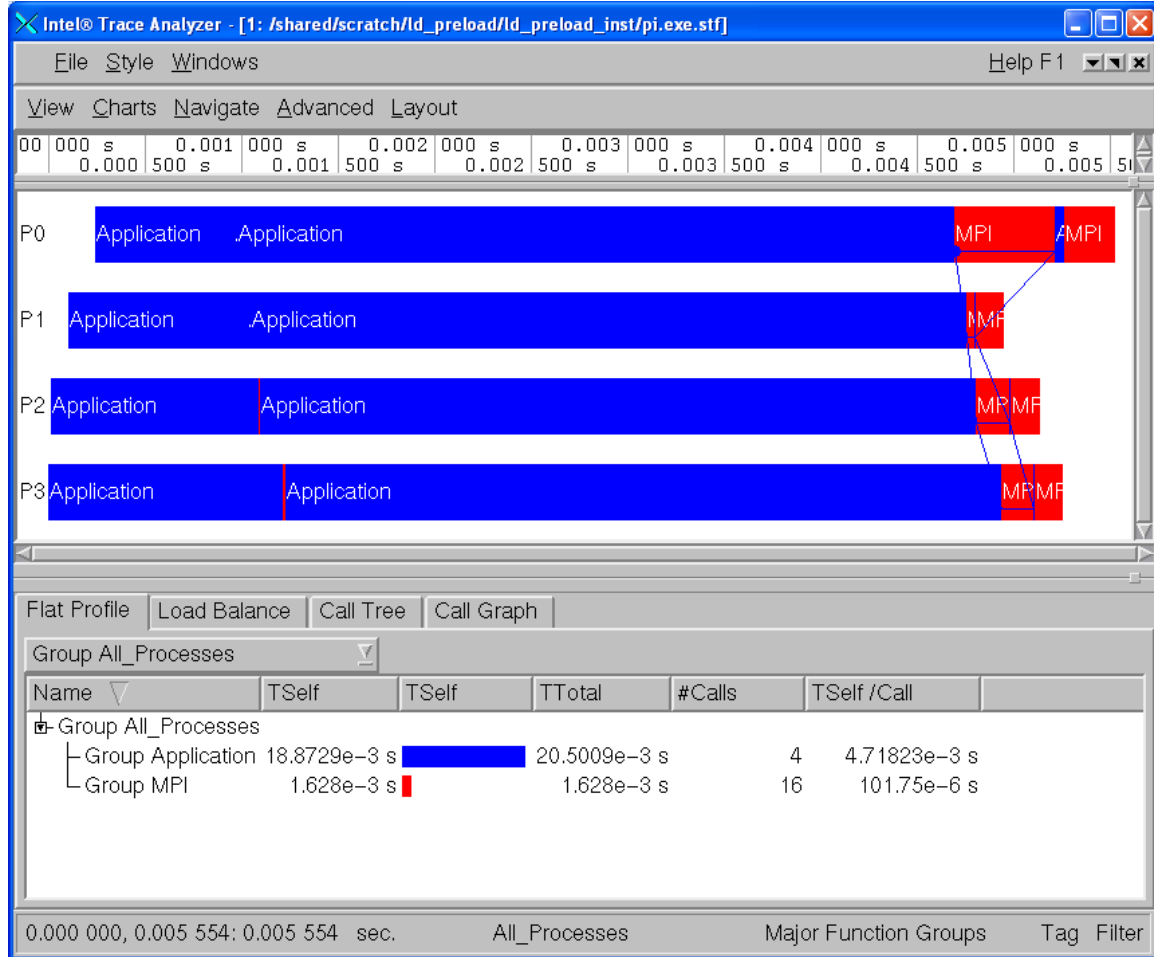


Figure 7.5 – Intel Trace Analyzer display of the “pi” integration application that has been instrumented through the `LD_PRELOAD` environment variable

Complete user documentation regarding the `LD_PRELOAD` environment variable for the Intel Trace Collector can be found within the file:

`<directory-path-to-ITAC>/doc/ITC_Reference_Guide.pdf`

on the system where the Intel Trace Collector is installed. You can use `LD_PRELOAD` as a search phrase within the documentation. To make inquiries about `LD_PRELOAD` in conjunction with Intel Trace Analyzer and Collector, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

7.4 Experimenting with Intel® Trace Analyzer and Collector in Conjunction with PAPI* Counters

The counter analysis discussion that follows assumes that a PAPI* library is installed on the cluster system. PAPI is an acronym for Performance API and it serves to gather information regarding performance counter hardware. Details can be found at the URL:

<http://icl.cs.utk.edu/papi/>

This discussion assumes that the PAPI library is installed in a directory path such as `/usr/local/papi`. In the examples directory for Intel Trace Analyzer and Collector, there is a subfolder called `poisson`. Using root privileges, the library called `libVTsample.a` needs to be configured in the `lib` directory of Intel Trace Analyzer and Collector so that PAPI instrumentation can be captured through the Intel Trace Analyzer and Collector. The library path for the Intel Trace Analyzer and Collector might be something like:

```
${VT_ROOT}/lib
```

In this directory, a system administrator can use the following `gmake` command to create the `libVTsample.a` library:

```
export PAPI_ROOT=/usr/local
gmake all
```

The environment variable `PAPI_ROOT` is used by the makefile to formulate the path to `${PAPI_ROOT}/include` which is a directory that contains PAPI header files. When the `libVTsample.a` library is built, the Poisson example can be linked with PAPI instrumentation as follows:

```
gmake MPI_HOME=${I_MPI_ROOT} make_dir=./ LIB_PATH="" LIBS="-
L${VT_ROOT}/lib -lVTsample -lVT -L${PAPI_ROOT}/papi/lib -lpapi
${VT_ADD_LIBS}"
```

The shell commands for running the `poisson` application might be the following:

```
rm -rf ${PWD}/papi_inst
mkdir ${PWD}/papi_inst
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${PAPI_ROOT}/papi/lib
setenv VT_LOGFILE_PREFIX ${PWD}/papi_inst
setenv VT_CONFIG ${PWD}/vtconfig
mpiexec -n 16 ./poisson
```

The Intel Trace Collector configuration file which is called `vtconfig` for the above example contains the following PAPI counter selection:

```
COUNTER PAPI_L1_DCM ON
```

This PAPI counter directive is for L1 data cache misses. The general syntax for counter directives is:

```
COUNTER <name of counter> ON
```

The value of ON indicates that this particular hardware counter is to be monitored by Intel Trace Collector. The names of the PAPI hardware counters can be found in the folder path `${PAPI_ROOT}/include/papiStdEventDefs.h` on the system where the PAPI library is installed.

Figure 7.6 illustrates a maximized view for the Counter Timeline Chart and the Function Profile Chart that were generated from the instrumentation data that was stored in the directory `${PWD}/papi_inst` as indicated by the environment variable `VT_LOGFILE_PREFIX`. The command that initiated the Intel Trace Analyzer with respect to the directory `${PWD}` was:

```
traceanalyzer papi_inst/poisson.stf &
```

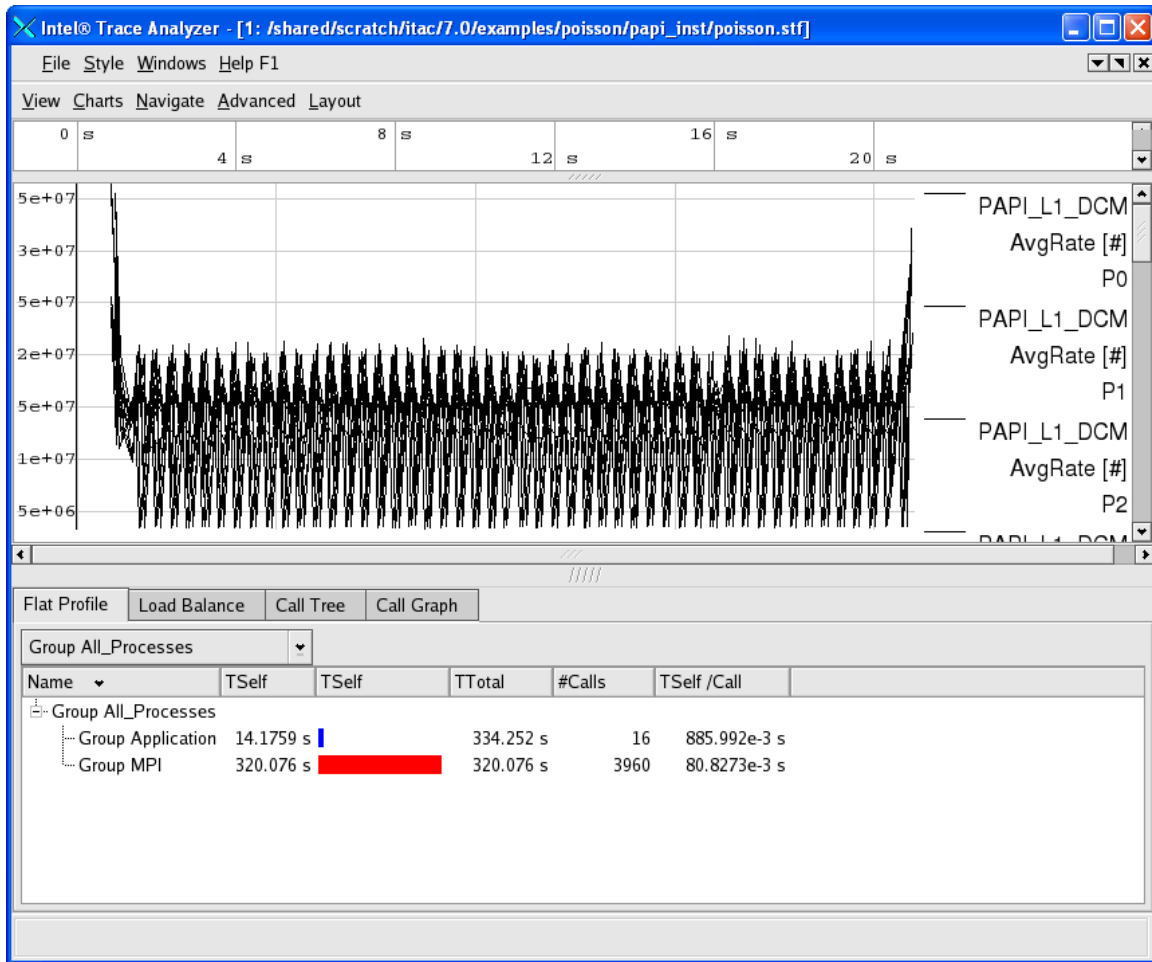


Figure 7.6 – A maximized view for the Counter Timeline Chart and the Function Profile Chart

NOTE: In the Counter Timeline Chart in Figure 7.6 that the PAPI counter PAPI_L1_DCM appears as a label in the right margin.

In general, the shell syntax for compiling the Intel MPI Library test files called test.c, test.cpp, test.f, and test.f90 with the PAPI interface involves the link options that look something like:

```
-L${VT_LIB_DIR} -lVTsample -lVT -L${PAPI_ROOT}/papi/lib -lpapi
${VT_ADD_LIBS}
```

The compilation commands are:

```
mpiicc test.c -o testc -L${VT_LIB_DIR} -lVTsample -lVT -
L${PAPI_ROOT}/papi/lib -lpapi ${VT_ADD_LIBS}
```

```
mpiicpc test.cpp -o testcpp -L${VT_LIB_DIR} -lVTsample -lVT -  
L${PAPI_ROOT}/papi/lib -lpapi ${VT_ADD_LIBS}
```

```
mpiifort test.f -o testf -L${VT_LIB_DIR} -lVTsample -lVT -  
L${PAPI_ROOT}/papi/lib -lpapi ${VT_ADD_LIBS}
```

```
mpiifort test.f90 -o testf90 -L${VT_LIB_DIR} -lVTsample -lVT -  
L${PAPI_ROOT}/papi/lib -lpapi ${VT_ADD_LIBS}
```

On Linux OS, complete user documentation regarding PAPI hardware counters for the Intel Trace Collector can be found within the file:

<directory-path-to-ITAC>/doc/ITC_Reference_Guide.pdf

on the system where the Intel Trace Collector is installed. You can use PAPI as a search phrase within the documentation. To make inquiries about PAPI in conjunction with the Intel Trace Analyzer and Collector, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

7.5 Experimenting with the Message Checking Component of Intel® Trace Collector

Intel Trace Collector environment variables which should be useful for message checking are:

`VT_DEADLOCK_TIMEOUT <delay>`, where *<delay>* is a time value. The default value is 1 minute and the notation for the meta-symbol *<delay>* could be 1m. This controls the same mechanism to detect deadlocks as in `libVTfs` which is the fail-safe library. For interactive use it is recommended to set it to a small value like "10s" to detect deadlocks quickly without having to wait long for the timeout.

`VT_DEADLOCK_WARNING <delay>` where *<delay>* is a time value. The default value is 5 minutes and the notation for the meta-symbol *<delay>* could be 5m. If on average the MPI processes are stuck in their last MPI call for more than this threshold, then a GLOBAL:DEADLOCK:NO PROGRESS warning is generated. This is a sign of a load imbalance or a deadlock which cannot be detected because at least one process polls for progress instead of blocking inside an MPI call.

`VT_CHECK_TRACING <on | off>`. By default, during correctness checking with `libVTmc` no events are recorded and no trace file is written. This option enables recording of all events also supported by the normal `libVT` and the writing of a trace file. The trace file will also contain the errors found during the run.

On Linux OS, complete user documentation regarding the message checking feature for the Intel Trace Collector can be found within the file:

`<directory-path-to-ITAC>/doc/ITC_Reference_Guide.pdf`

The chapter title is called "Correctness Checking".

An MPI application can be instrumented in four ways with the message checking library.

- 1) Compile the application with a static version of the message checking library:

```
mpiicc deadlock.c -o deadlock_static.exe -g -L ${VT_LIB_DIR} -lVTmc  
${VT_ADD_LIBS}
```

```
mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv  
VT_DEADLOCK_WARNING 25s -n 2 ./deadlock_static.exe 0 80000
```

- 2) Compile the application with a shared object version of the message checking library:

```
mpiicc deadlock.c -o deadlock_shared.exe -g -L ${VT_SLIB_DIR} -lVTmc  
${VT_ADD_LIBS}
```

```
mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv  
VT_DEADLOCK_WARNING 25s -n 2 ./deadlock_shared.exe 0 80000
```

NOTE: The library path for the Intel® C++ Compiler will vary from version to version.

- 3) Use the `itcpin` command:

```
mpiicc deadlock.c -o deadlock.exe -g
```

```
mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv  
VT_DEADLOCK_WARNING 25s -n 2 itcpin --insert libVTmc.so --run --  
./deadlock.exe 0 80000
```

- 4) Use the `LD_PRELOAD` environment variable with the `mpiexec` command. An example might be:

```
mpiicc deadlock.c -o deadlock.exe -g
```

```
mpiexec -genv VT_CHECK_TRACING on -genv LD_PRELOAD libVTmc.so -genv  
VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -n 2  
./deadlock.exe 0 80000
```

There is a sub-directory of the examples directory called `checking`. The `checking` directory has the following contents:

```
global/  GNUmakefile  local/  misc/
```

The GNUmakefile has targets `all`, `clean`, `print`, and `run`, where `all` is the default. After typing `gmake`, one can type the command:

```
gmake run
```

The output error diagnostics for the command above will be sent to `stderr`. If you wish to retain the output into a file, the results for `stderr` can be directed to a file.

Each leaf sub-folder contains a source file and an `*.ref.out` file which can be used as a point of reference for the expected diagnostics that the message checking component of the Intel® Trace Collector should capture. For example, if you search the global sub-directory, you will find a folder path of the following form:

```
global/collective/datatype_mismatch/
```

The contents of the leaf directory consist of:

```
MPI_Bcast.c  MPI_Bcast.ref.out
```

The file `MPI_Bcast.ref.out` has diagnostic information that looks something like the following:

```

                                ...
[0] INFO: initialization completed successfully

[0] ERROR: GLOBAL:COLLECTIVE:DATATYPE:MISMATCH: error
[0] ERROR:   Mismatch found in local rank [1] (global rank [1]),
[0] ERROR:   other processes may also be affected.
[0] ERROR:   No problem found in local rank [0] (same as global rank):
[0] ERROR:     MPI_Bcast(*buffer=0x7fbffffe9f0, count=1, datatype=MPI_INT,
root=0, comm=MPI_COMM_WORLD)
[0] ERROR:     main (global/collective/datatype_mismatch/MPI_Bcast.c:50)
[0] ERROR:     1 elements transferred by peer but 4 expected by
[0] ERROR:     the 3 processes with local ranks [1:3] (same as global ranks):
[0] ERROR:     MPI_Bcast(*buffer=0x7fbffffe9f4, count=4, datatype=MPI_CHAR,
root=0, comm=MPI_COMM_WORLD)
[0] ERROR:     main (global/collective/datatype_mismatch/MPI_Bcast.c:53)

[0] INFO: GLOBAL:COLLECTIVE:DATATYPE:MISMATCH: found 1 time (1 error + 0
warnings), 0 reports were suppressed
[0] INFO: Found 1 problem (1 error + 0 warnings), 0 reports were suppressed.
```

For the text above, there are error messages of the form:

```
[0] ERROR:     main (global/collective/datatype_mismatch/MPI_Bcast.c:50)
```

and

```
[0] ERROR:     main (global/collective/datatype_mismatch/MPI_Bcast.c:53)
```

These error messages refer to the line number 50 and 53 respectively in the source file `MPI_Bcast.c`:

```

                                     ...
39 int main (int argc, char **argv)
40 {
41     int rank, size;
42
43     MPI_Init( &argc, &argv );
44     MPI_Comm_size( MPI_COMM_WORLD, &size );
45     MPI_Comm_rank( MPI_COMM_WORLD, &rank );
46
47     /* error: types do not match */
48     if( !rank ) {
49         int send = 0;
50         MPI_Bcast( &send, 1, MPI_INT, 0, MPI_COMM_WORLD );
51     } else {
52         char recv[4];
53         MPI_Bcast( &recv, 4, MPI_CHAR, 0, MPI_COMM_WORLD );
54     }
55
56     MPI_Finalize( );
57
58     return 0;
59 }
```

At lines 52 and 53, adjustments can be made to the source which would look something like the following:

```

52         int recv[4];
53         MPI_Bcast( &recv, 1, MPI_INT, 0, MPI_COMM_WORLD );
```

The modifications are to change the data-type definition for the object `recv` at line 52 from `char` to `int`, and at line 53, the third argument which is the MPI data-type is modified from `MPI_CHAR` to `MPI_INT`.

Upon doing this and following a process of recompiling and re-running the application will generate the following:

```

                                     ...
[0 Thu Mar 26 19:53:34 2009] INFO: Error checking completed without
finding any problems.
                                     ...
```

This indicates the message checking errors that were originally encountered have been eliminated for this example.

At the URL:

<http://www.shodor.org/refdesk/Resources/Tutorials/BasicMPI/deadlock.c>

one can obtain the source to an MPI example using C bindings that demonstrates deadlock.

When issuing the `mpiexec` command with the `LD_PRELOAD` environment variable:

```
mpiexec -genv VT_CHECK_TRACING on -genv VT_LOGFILE_PREFIX
/shared/scratch/test_correctness_checking/inst -genv LD_PRELOAD
libVTmc.so -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s
-n 2 ./deadlock.exe 0 80000
```

diagnostic messages that look something like the following are generated.

```

                                ...
0/2: receiving 80000

1/2: receiving 80000
[0] ERROR: no progress observed in any process for over 0:29 minutes,
aborting application
[0] WARNING: starting premature shutdown

[0] ERROR: GLOBAL:DEADLOCK:HARD: fatal error
[0] ERROR:      Application aborted because no progress was observed for
over 0:29 minutes,
[0] ERROR:      check for real deadlock (cycle of processes waiting for
data) or
[0] ERROR:      potential deadlock (processes sending data to each other
and getting blocked
[0] ERROR:      because the MPI might wait for the corresponding
receive).
[0] ERROR:      [0] no progress observed for over 0:29 minutes, process
is currently in MPI call:
[0] ERROR:      MPI_Recv(*buf=0x7fbf9e4740, count=800000,
datatype=MPI_INT, source=1, tag=999, comm=MPI_COMM_WORLD,
*status=0x7fbffffef40)
[0] ERROR:      main
(/shared/scratch/test_correctness_checking/deadlock.c:49)
[0] ERROR:      (/lib64/tls/libc-2.3.4.so)
[0] ERROR:
(/shared/scratch/test_correctness_checking/deadlock.exe)
[0] ERROR:      [1] no progress observed for over 0:29 minutes, process
is currently in MPI call:
[0] ERROR:      MPI_Recv(*buf=0x7fbf9e4740, count=800000,
datatype=MPI_INT, source=0, tag=999, comm=MPI_COMM_WORLD,
*status=0x7fbffffef40)
```

```
12 [0] ERROR:      main
   (/shared/scratch/test_correctness_checking/deadlock.c:49)

13 [0] ERROR:      (/lib64/tls/libc-2.3.4.so)

14 [0] ERROR:
   (/shared/scratch/test_correctness_checking/deadlock.exe)

15

16 [0] INFO: GLOBAL:DEADLOCK:HARD: found 1 time (1 error + 0
   warnings), 0 reports were suppressed

17 [0] INFO: Found 1 problem (1 error + 0 warnings), 0 reports were
   suppressed.
```

The compiler option `-g` inserts debug information that allows one to map from the executable back to the source code. Because the environment variable `VT_CHECK_TRACING` was set for the `mpiexec` command, trace information was placed into the directory referenced by `VT_LOGFILE_PREFIX` which for the example command-line:

```
mpiexec -genv VT_CHECK_TRACING on -genv VT_LOGFILE_PREFIX
/shared/scratch/test_correctness_checking/inst -genv LD_PRELOAD
libVTmc.so -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s
-n 2 ./deadlock.exe 0 80000
```

is `/shared/scratch/test_correctness_checking/inst`.

You can use the Intel® Trace Analyzer to view the deadlock problem that was reported in the output listing above. Here is what the trace information might look like (Figure 7.7):

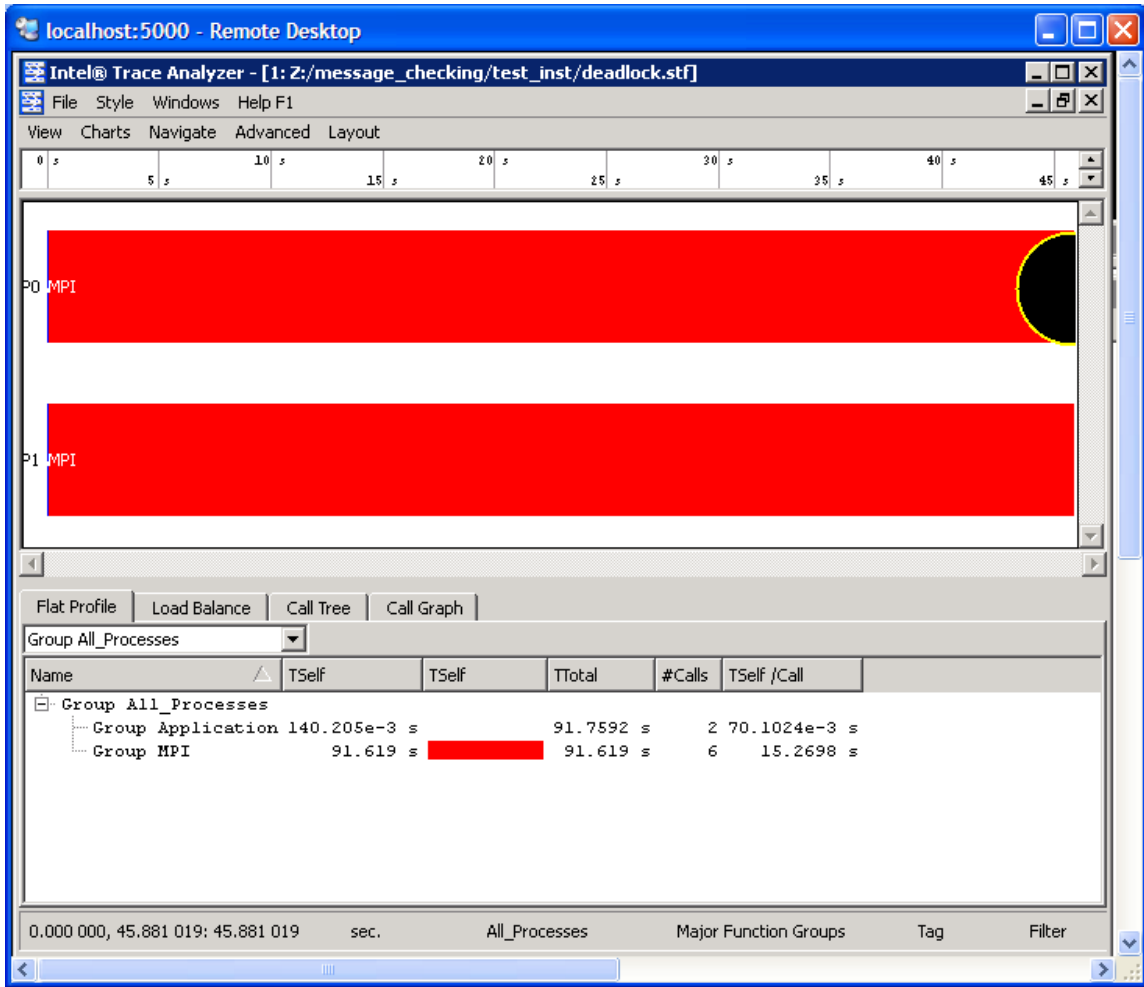


Figure 7.7 – Event Timeline illustrating an error as signified by the black circle

For the event timeline chart, errors and warnings are represented by yellow-bordered circles (Figure 7.7). The color of each circle depends on the type of the particular diagnostic. If there is an error the circle will be filled in with a black coloring. If there is a warning, the circle will be filled in with a gray coloring.

For Figure 7.7, error messages and warnings can be suppressed by using a context menu. A context menu will appear if you right click the mouse as shown in Figure 7.8 and follow the path **Show->Issues**. If you uncheck the **Issues** item, the black and gray circles will clear.

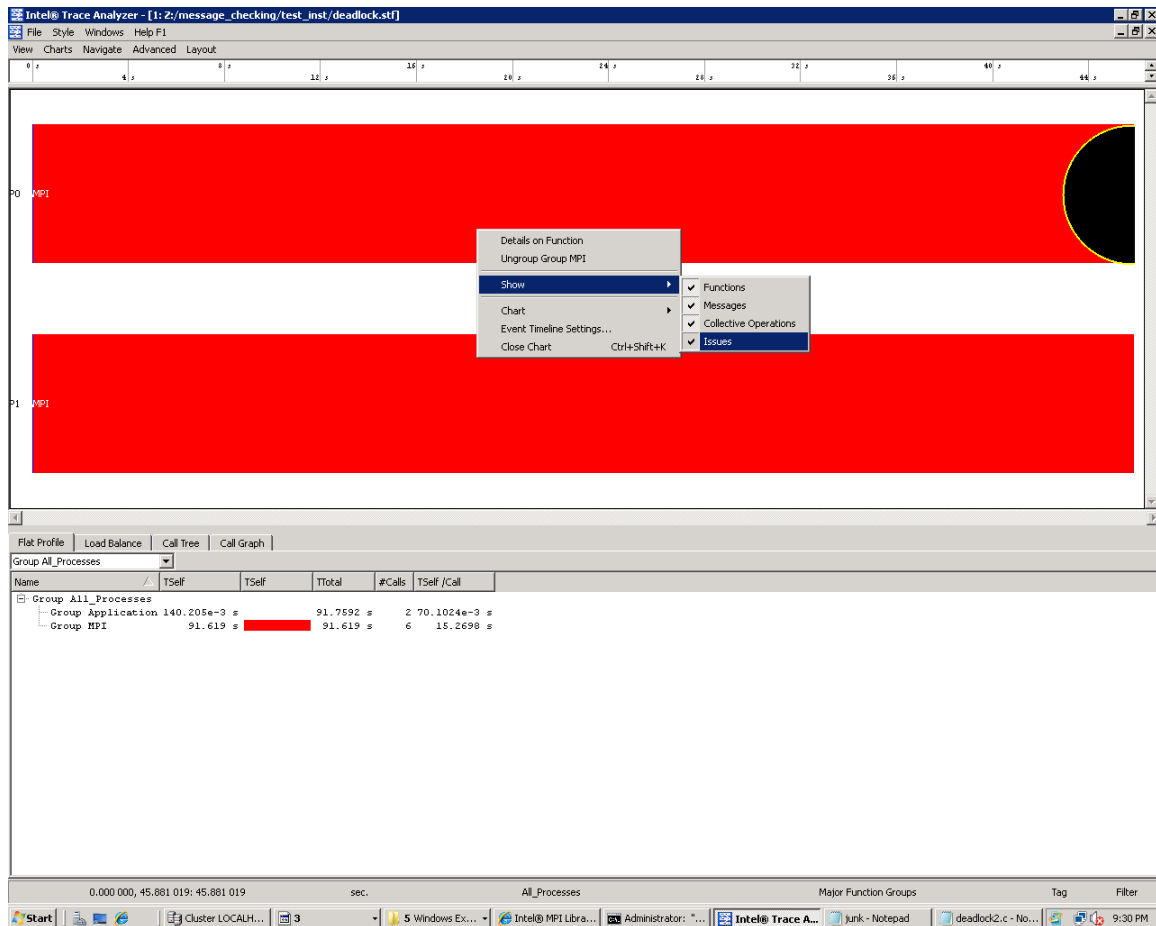


Figure 7.8 – Context menu that can be used to suppress “Issues”. This is done by un-checking the “Issues” item

You can determine what source line is associated with an error message by using the context menu and selecting Details on Function. This will generate the following Details on Function panel (Figure 7.9):

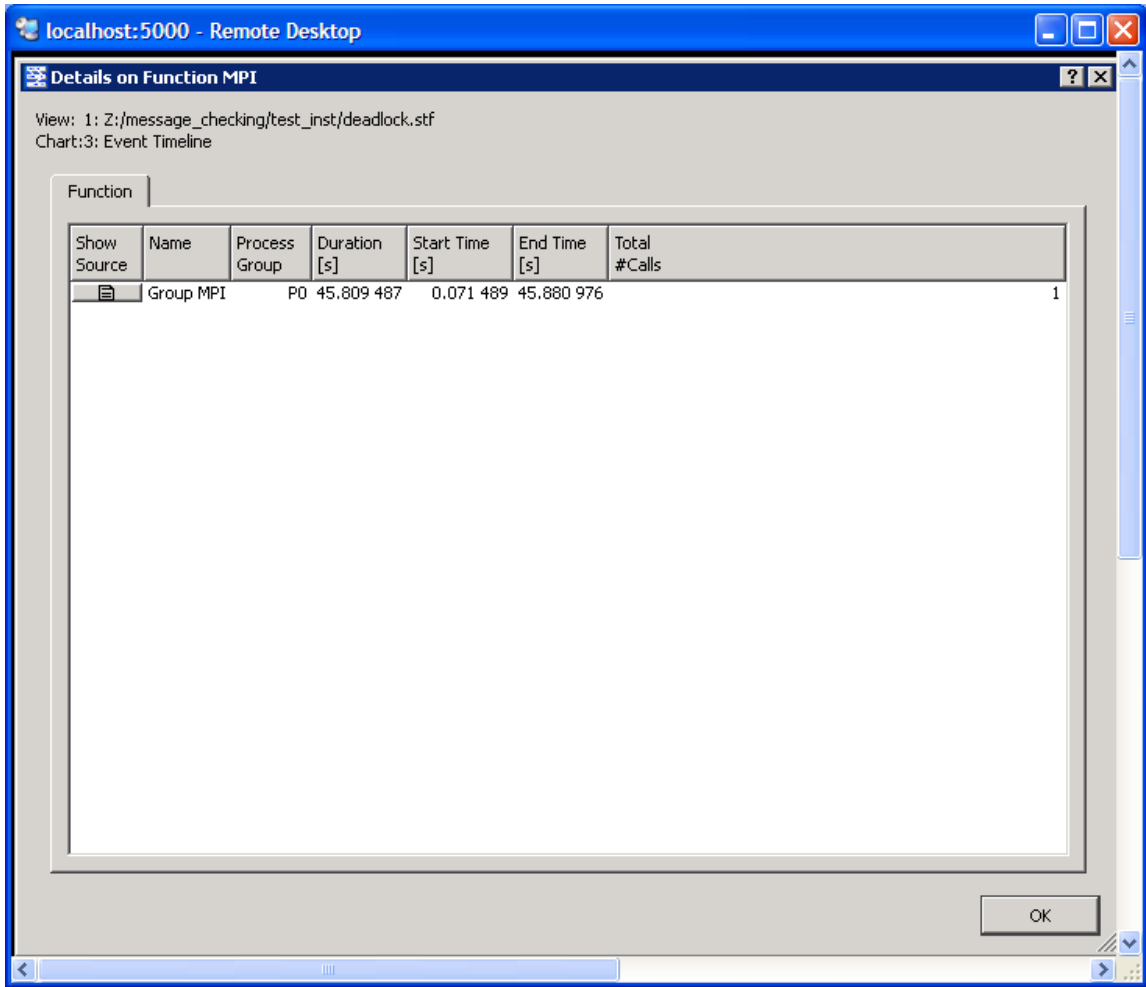


Figure 7.9 – Illustration of the Detail on Function panel. The Show Source tab is the first item on the left

If you click on the **Show Source** tab in Figure 7.9, you will ultimately reach a source file panel such as what is demonstrated in Figure 7.10.

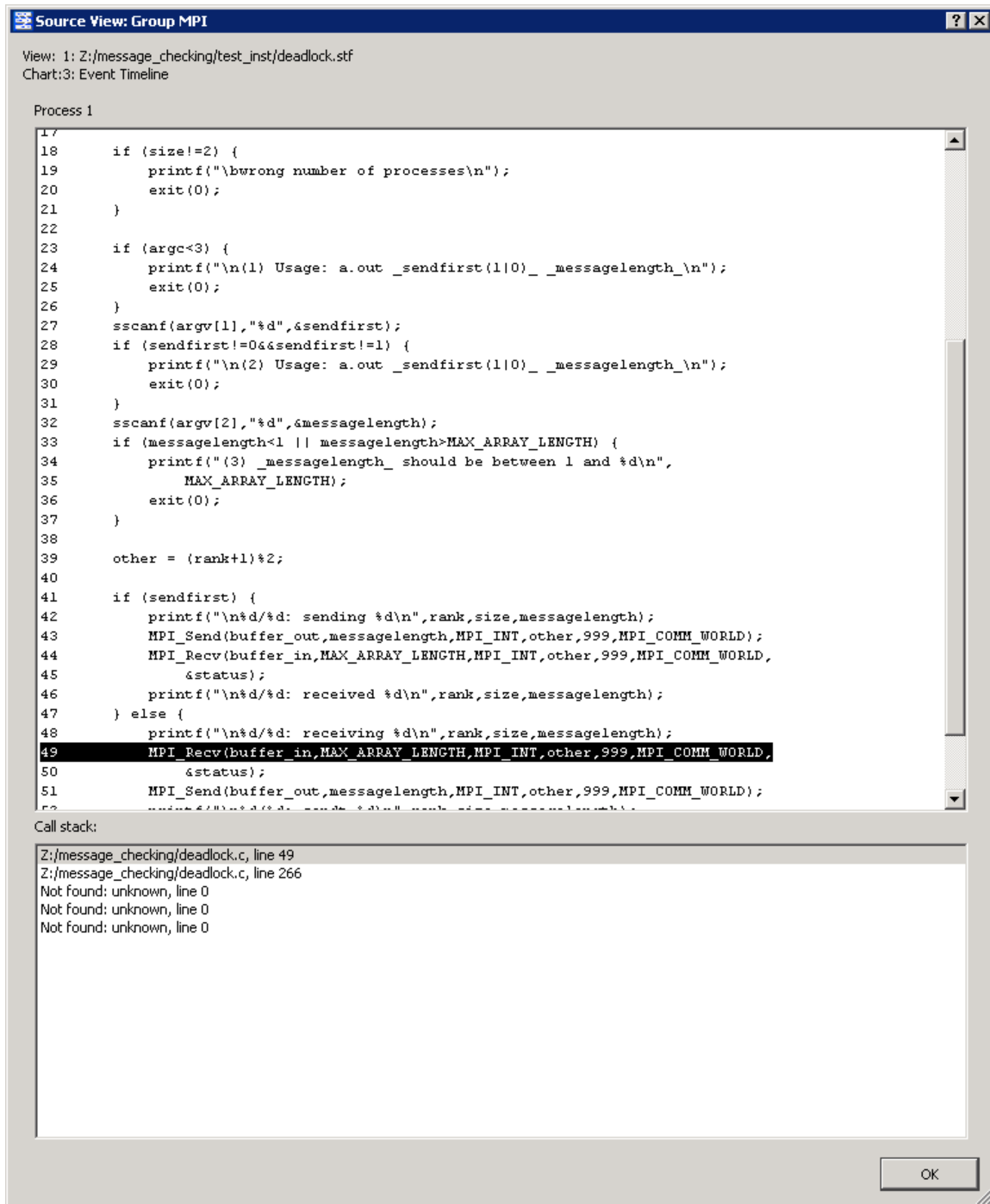


Figure 7.10 – The source panel display which shows the line in the user’s source where deadlock has taken place.

The diagnostic text messages and the illustration in Figure 7.10 reference line 49 of deadlock.c looks something like the following:

```

...
49     MPI_Recv (buffer_in, MAX_ARRAY_LENGTH, MPI_INT, other,
999,
50             MPI_COMM_WORLD, &status);
51     MPI_Send (buffer_out, messagelength, MPI_INT, other, 999,
52             MPI_COMM_WORLD);
...

```

This is illustrated in Figure 7.11. To avoid deadlock situations, one might be able to resort to the following solutions:

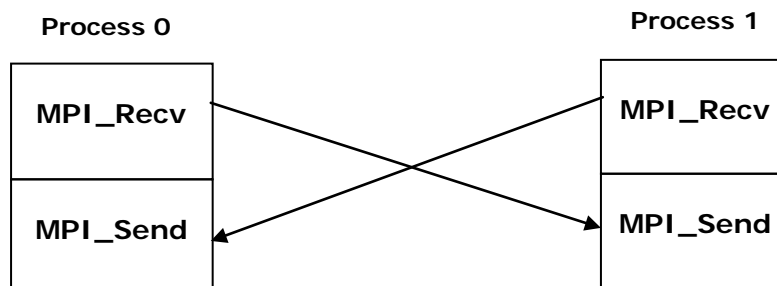


Figure 7.11 – Cycle illustration for processes 0 and 1 when executing source lines 49 and 43 within application deadlock.c

1. Use a different ordering of MPI communication calls between processes
2. Use non-blocking calls
3. Use MPI_Sendrecv or MPI_Sendrecv_replace
4. Buffered mode

The if-structure for the original program looks something like the following:

```

...
41  if (sendfirst) {
42      printf ("\n%d/%d: sending %d\n", rank, size, messagelength);
43      MPI_Send (buffer_out, messagelength, MPI_INT, other, 999, MPI_COMM_WORLD);
44      MPI_Recv (buffer_in, MAX_ARRAY_LENGTH, MPI_INT, other, 999,
45              MPI_COMM_WORLD, &status);
46      printf ("\n%d/%d: received %d\n", rank, size, messagelength);
47  } else {
48      printf ("\n%d/%d: receiving %d\n", rank, size, messagelength);
49      MPI_Recv (buffer_in, MAX_ARRAY_LENGTH, MPI_INT, other, 999,
50              MPI_COMM_WORLD, &status);
51      MPI_Send (buffer_out, messagelength, MPI_INT, other, 999,
52              MPI_COMM_WORLD);
33      printf ("\n%d/%d: sendt %d\n", rank, size, messagelength);
54  }
...

```

If you replace lines 43 to 44 and lines 49 to 52 with calls to MPI_Sendrecv so that they look something like:

```
MPI_Sendrecv (buffer_out, messagelength, MPI_INT, other, 999,
buffer_in, MAX_ARRAY_LENGTH, MPI_INT, other, 999, MPI_COMM_WORLD,
&status);
```

and save this information into a file called deadlock2.c, and proceed to compile the modified application. The result of running the mpiexec command:

```
mpiexec -genv VT_CHECK_TRACING on -genv LD_PRELOAD libVTmc.so -genv
VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -n 2
./deadlock2.exe 0 80000
```

is the following:

```

                                ...
0/2: receiving 80000
1/2: receiving 80000
0/2: sent 80000
1/2: sent 80000
[0] INFO: Error checking completed without finding any problems.
```

This indicates the deadlock errors that were originally encountered have been eliminated for this example. Using the Intel® Trace Analyzer to view the instrumentation results, you can see that the deadlock issues have been resolved (Figure 7.12).

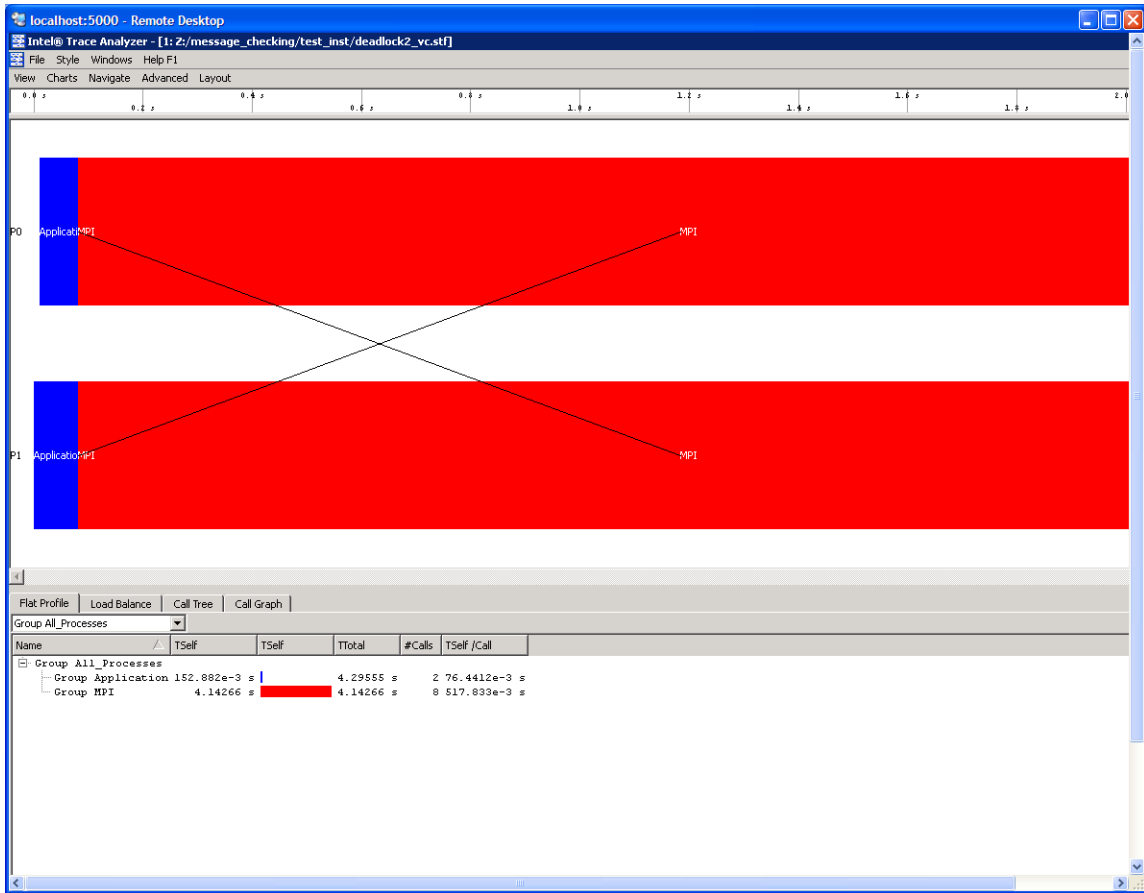


Figure 7.12 – Illustration of deadlock removal by using MPI_Sendrecv in the original source file called deadlock.c

[Back to Table of Contents](#)

7.6 Saving a Working Environment through a Project File

There may be situations where you are in the middle of an inspection with Intel® Trace Analyzer and you need to be away. For example, suppose you initially typed the command:

```
traceanalyzer test_inst/testcpp.stf
```

and you need to temporarily stop the analysis, and you are looking at the following panel:

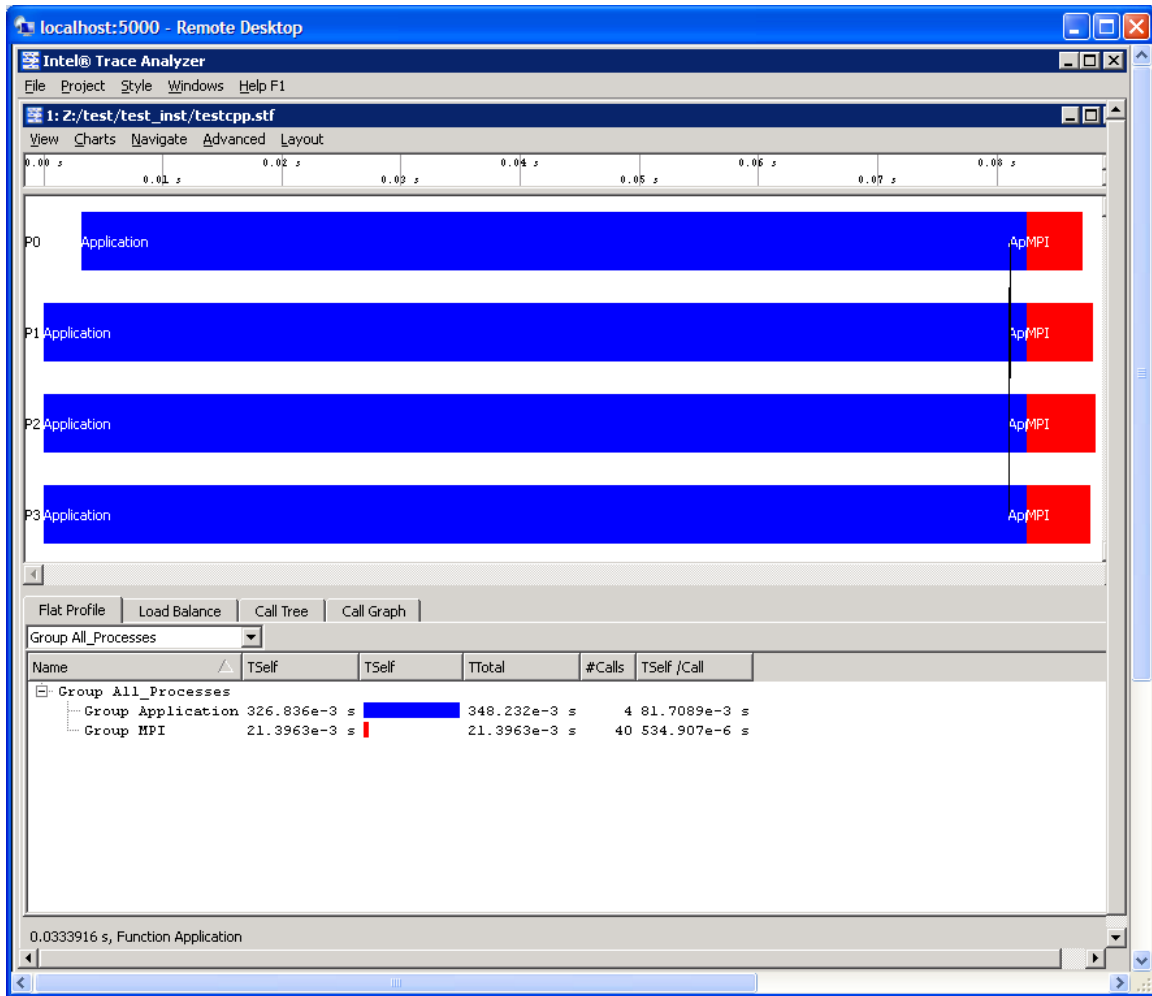


Figure 7.13 – Event timeline for running 4 MPI processes for the executable generated from test.cpp

For the panel rendering above, if you selection **Project->Save Project** or **Project->Save Project As...**, you will generate a subpanel that allows you to save the state of your session. This is project file has a suffix of ".itapr", which is an acronym for Intel® Trace Analyzer project. Figure 7.14 shows the process of saving the state of your session through a project file.

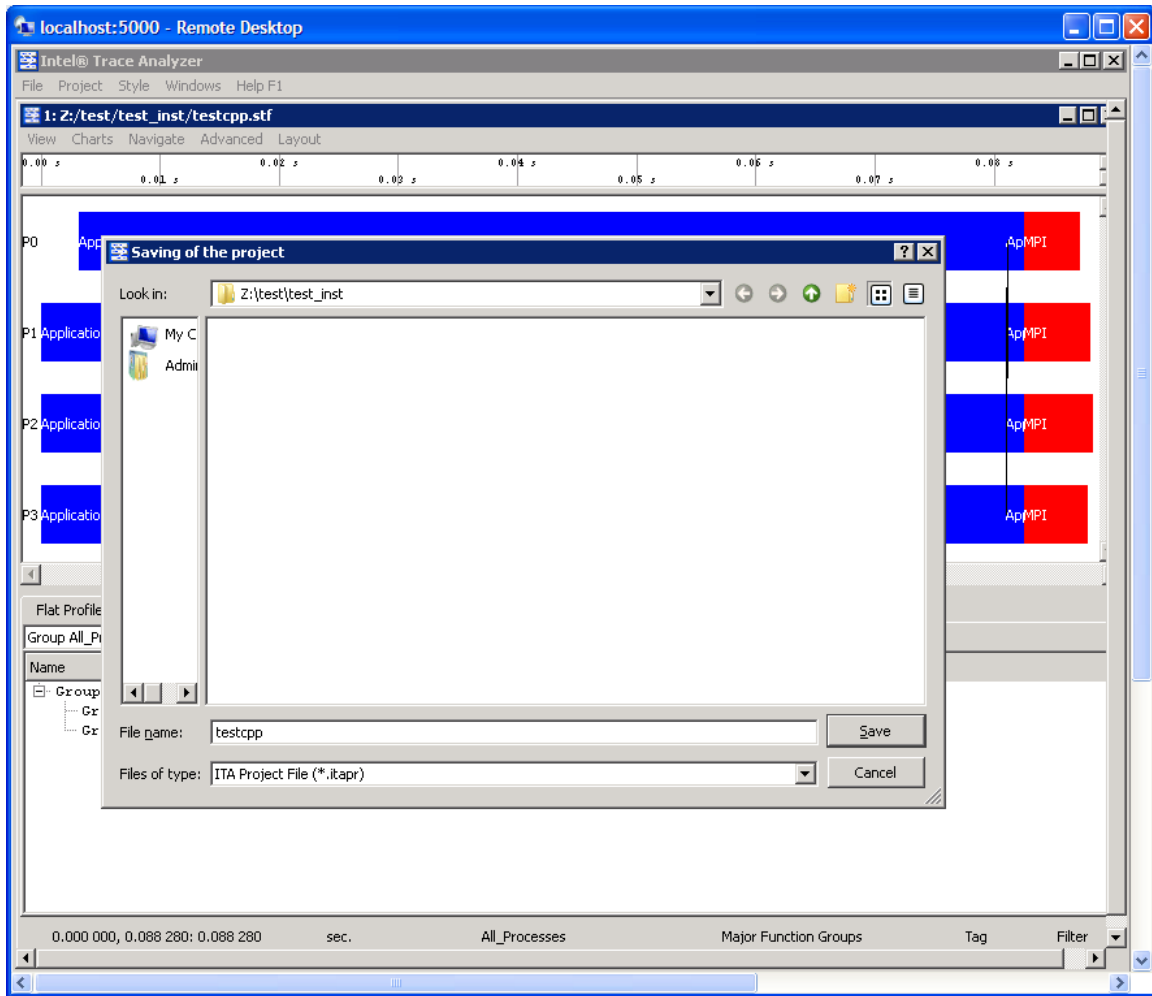


Figure 7.14 – Saving a Project File called testcpp.itapr

Suppose at a later time you wish to continue the analysis with Intel® Trace Analyzer. You can type the command:

```
traceanalyzer
```

You can then select **Project->Load Project...** and the following subpanel will appear (Figure 7.15):

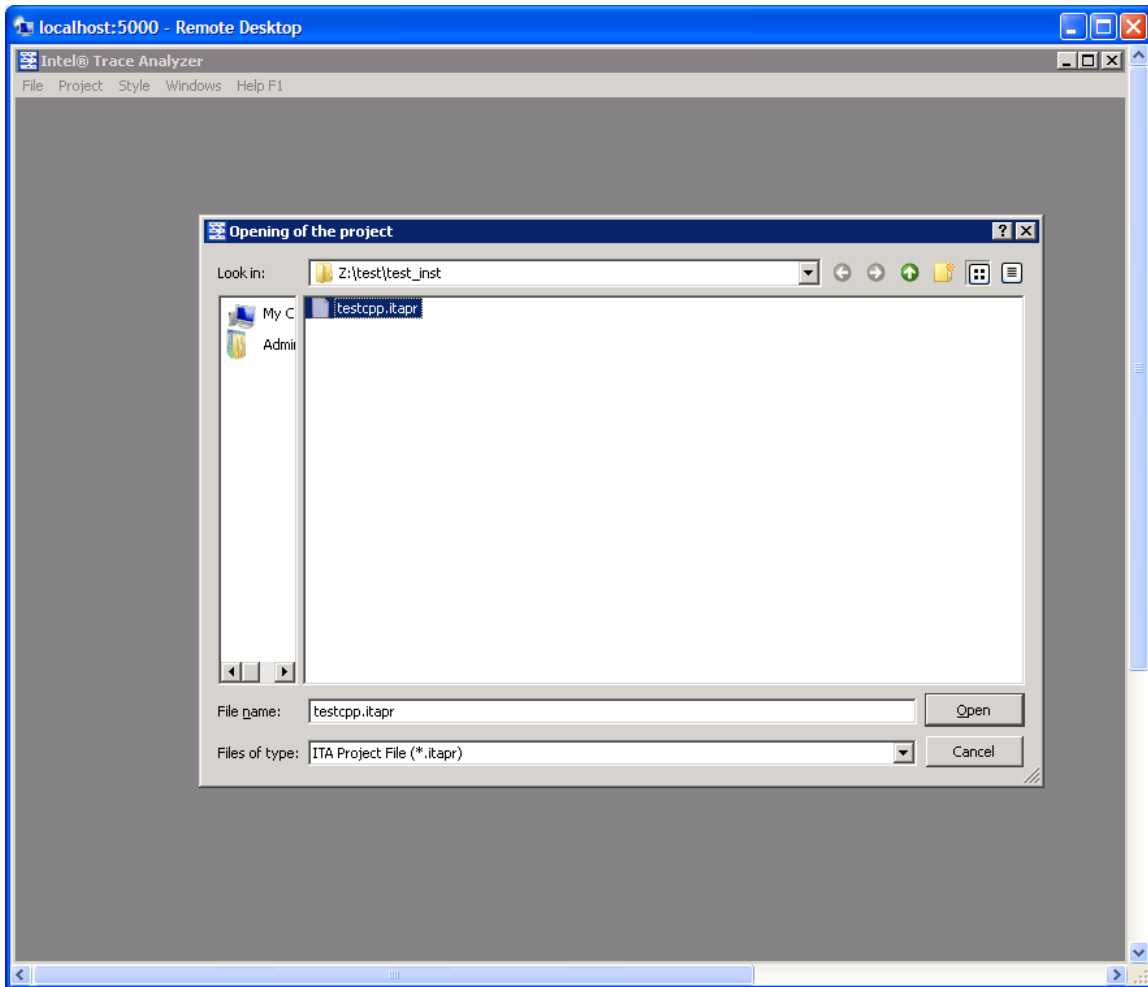


Figure 7.15 – Loading a Project File called testcpp.itapr

With regards to Figure 7.15, click on the **Open** button and you will immediately go back to point where you last left off (Figure 7.13). For complete details on saving and loading a project file, please see Section 2.2 of the Intel® Trace Analyzer Reference Guide, which is titled “Project Menu”. The path to this file is:

<directory-path-to-ITAC>/doc/ITA_Reference_Guide.pdf

on the system where the Intel® Trace Analyzer and Collector is installed.

[Back to Table of Contents](#)

7.7 Analysis of Application Imbalance

With respect to Figure 6.13, you may want to know a summary of process imbalance for the executable. You can do this by selecting the menu path **Advanced-**

>**Application Imbalance Diagram.** Figure 7.16 shows the result of making this selection.

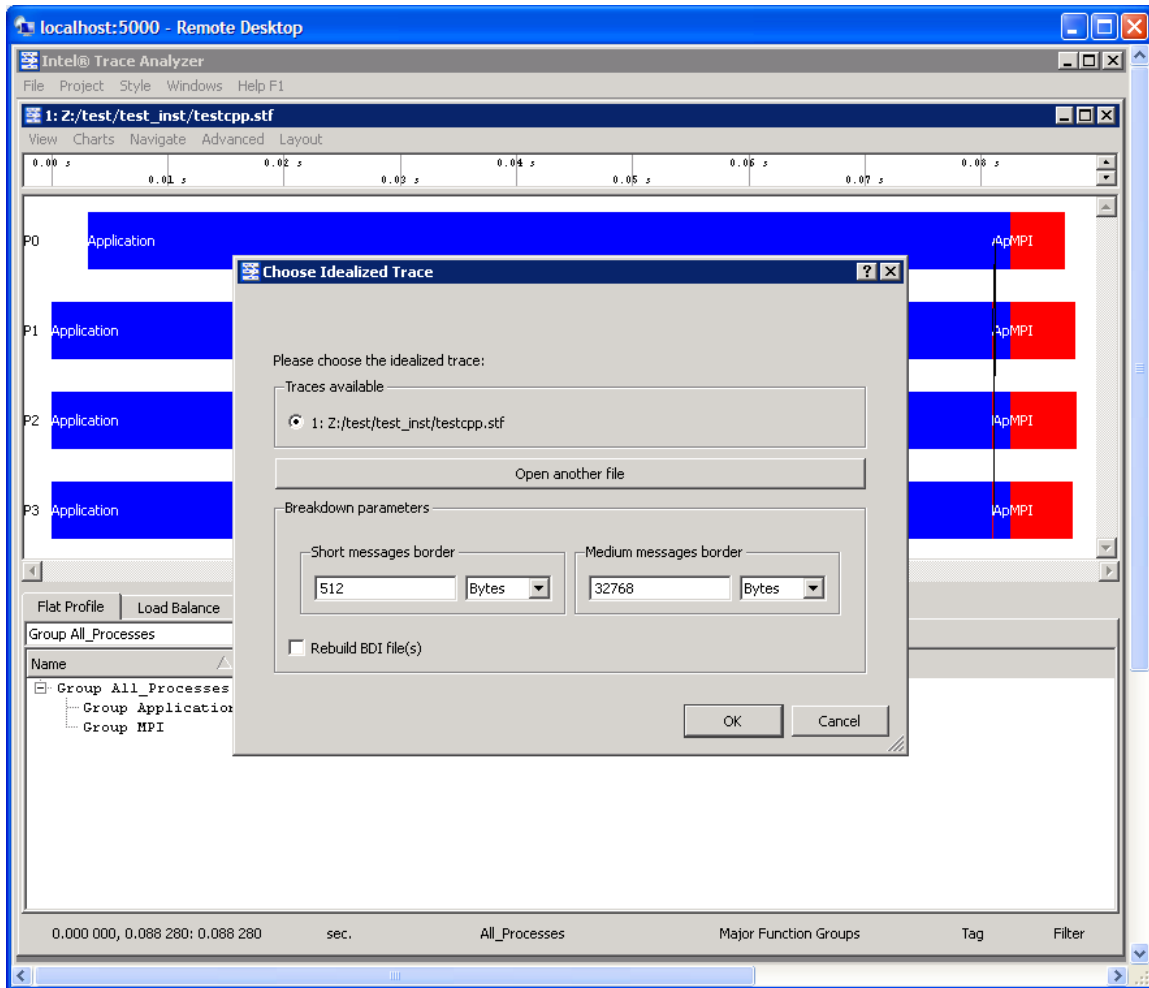


Figure 7.16 – Selecting Application Imbalance for the menu selection Advanced->Application Imbalance Diagram

Click on the **OK** button in the subpanel will generate the following (Figure 7.17). You can verify the meaning of the histogram subcomponents by clicking on the **Colors...** button in Figure 7.17. This will generate the panel shown in Figure 7.18.

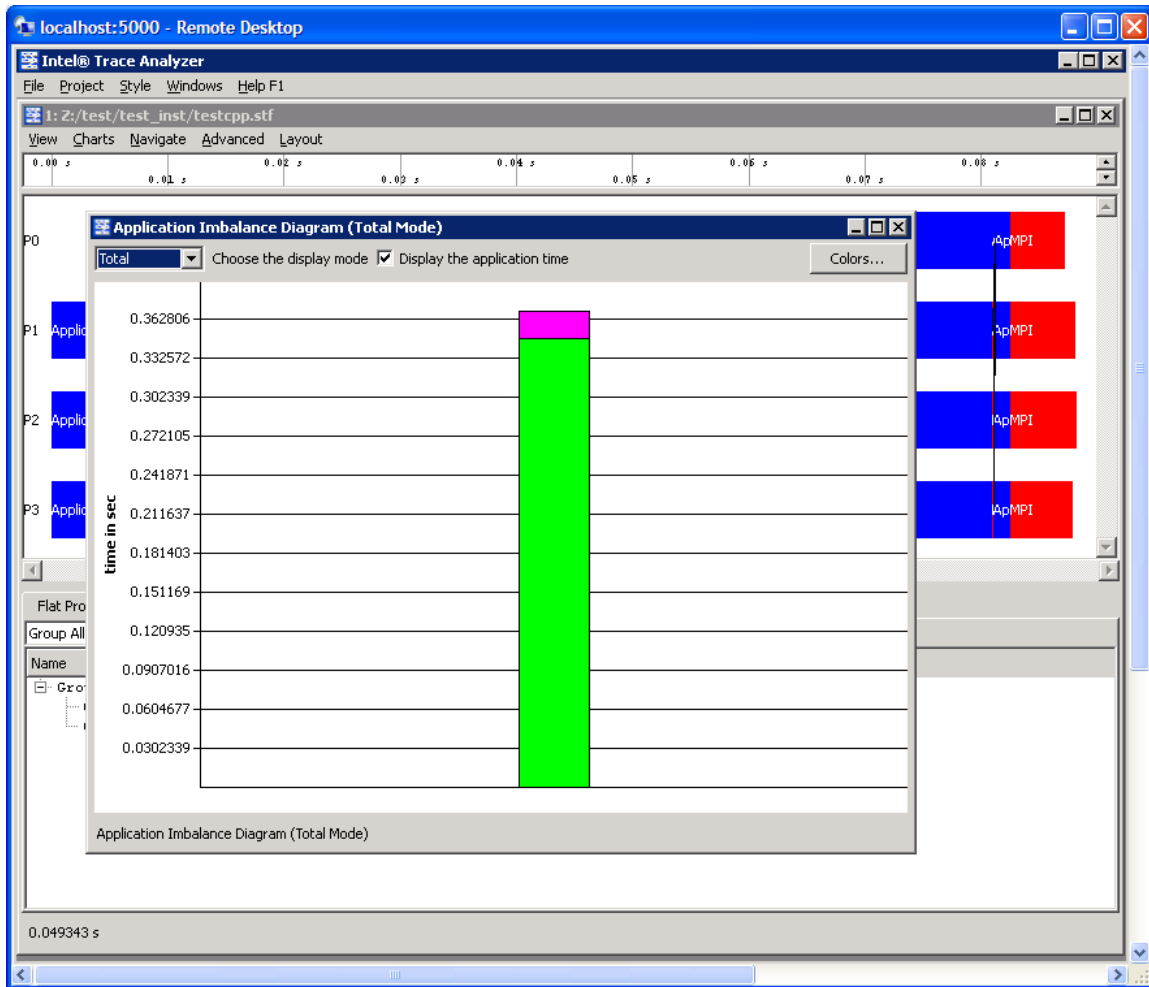


Figure 7.17 – Histogram subpanel as a result of pressing the OK button shown in Figure 7.16

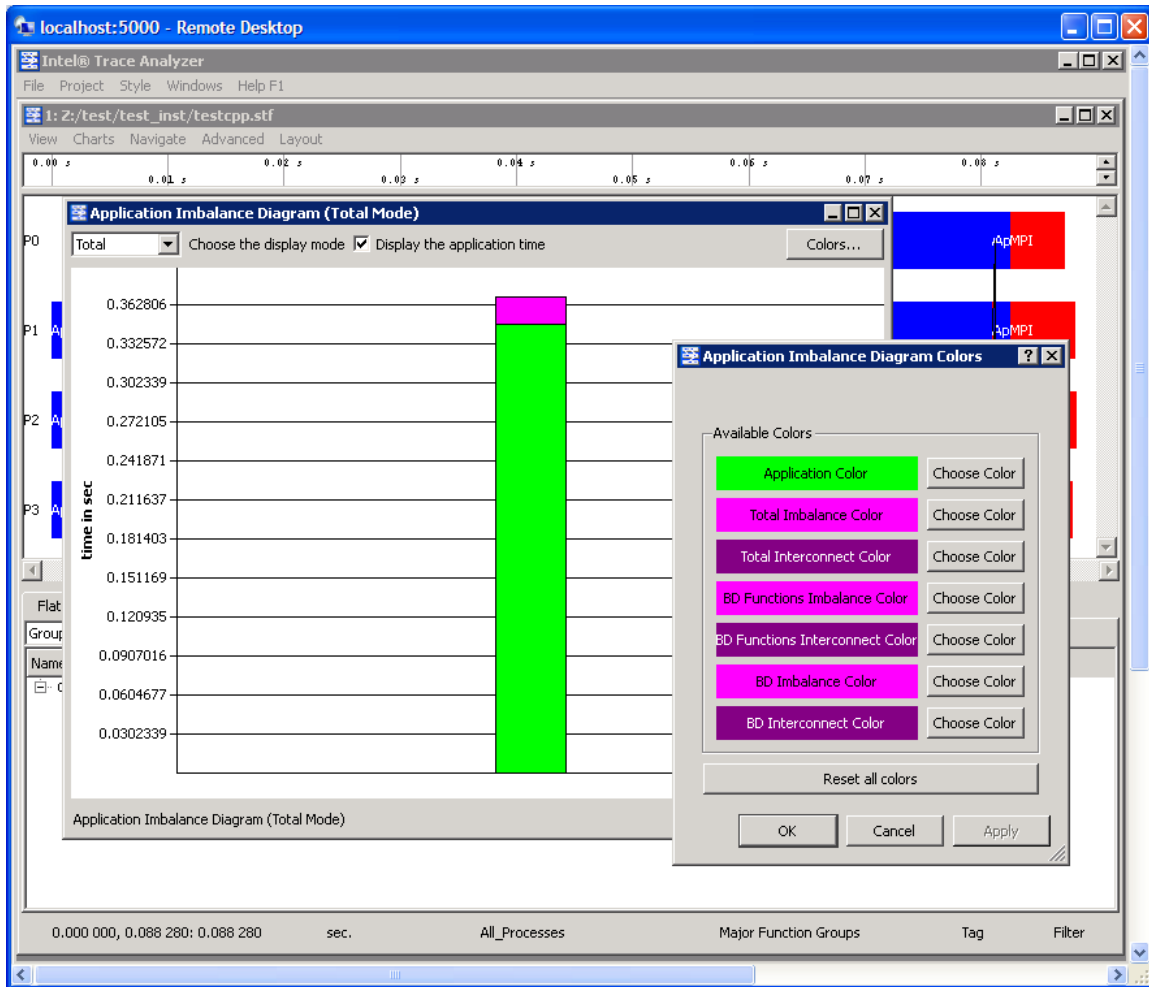


Figure 7.18 – Legend for interpreting the histogram contributions for the Application Imbalance Diagram

For complete details on application imbalance, please see Section 5.4 of the Intel® Trace Analyzer Reference Guide, which is titled “Application Imbalance Diagram Dialog Box”. The path to this file is:

<directory-path-to-ITAC>/doc/ITA_Reference_Guide.pdf

on the system where the Intel® Trace Analyzer and Collector is installed.

[Back to Table of Contents](#)

7.8 Analysis with the Ideal Interconnect Simulator

In analyzing the performance of your executable, you can compare your instrumentation trace with an ideal trace for the executable. To do this, make the

menu selection **Advanced->Idealization**. As a result, a dialog subpanel will appear which will allow you to create an idealized trace of execution (Figure 7.19):

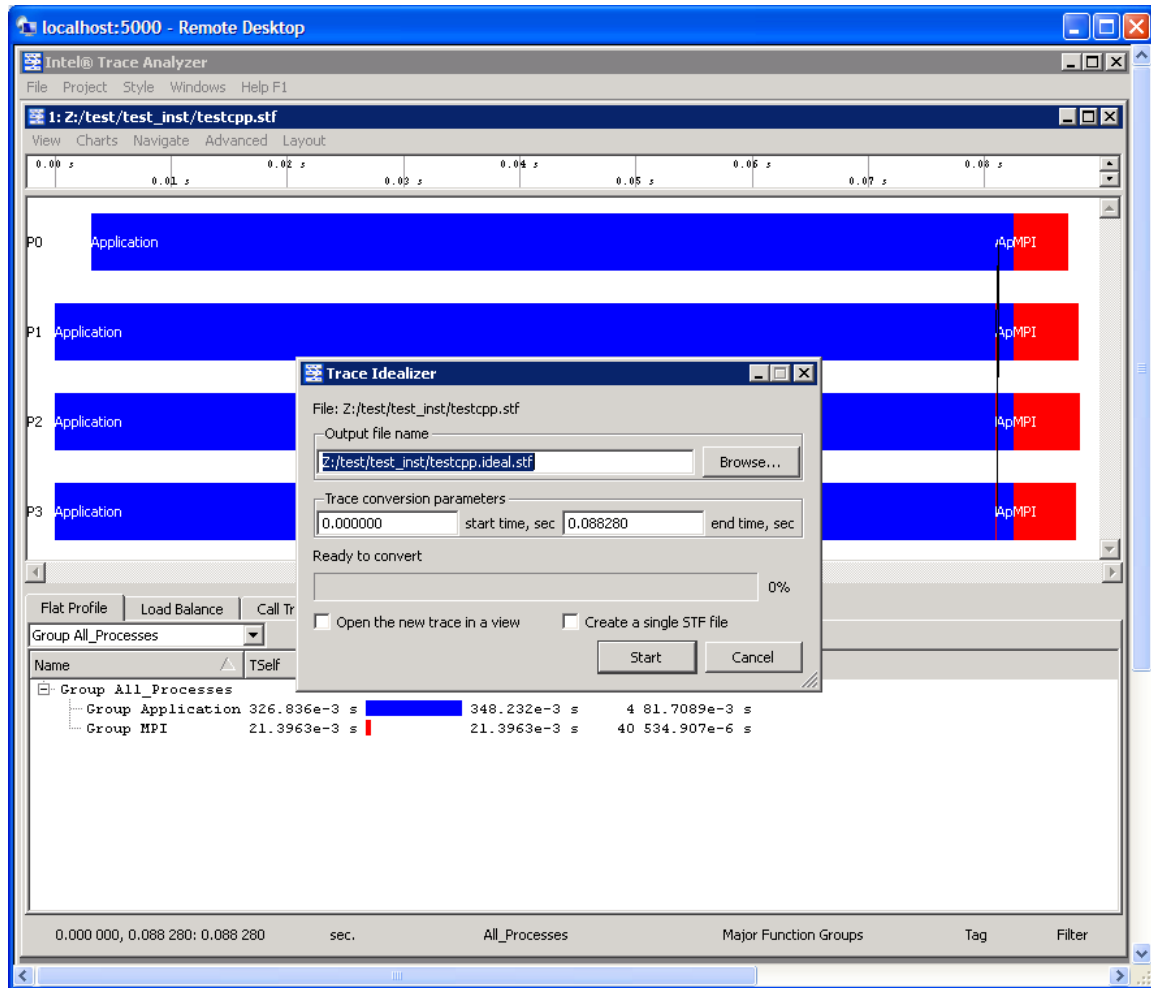


Figure 7.19 – Trace Idealizer dialog box generated as a result of the menu selection Advanced->Idealization

By clicking on the Start button in the dialog panel for Figure 7.19, a trace file will be generated called "testcpp.ideal.stf". After creating this file, you can then make the menu selection **File->Open** for the given Intel® Trace Analyzer panel and open the trace file "testcpp.ideal.stf" for comparative analysis. Figure 7.20 shows the side-by-side results of the actual execution trace and the ideal trace for the application "test.cpp".

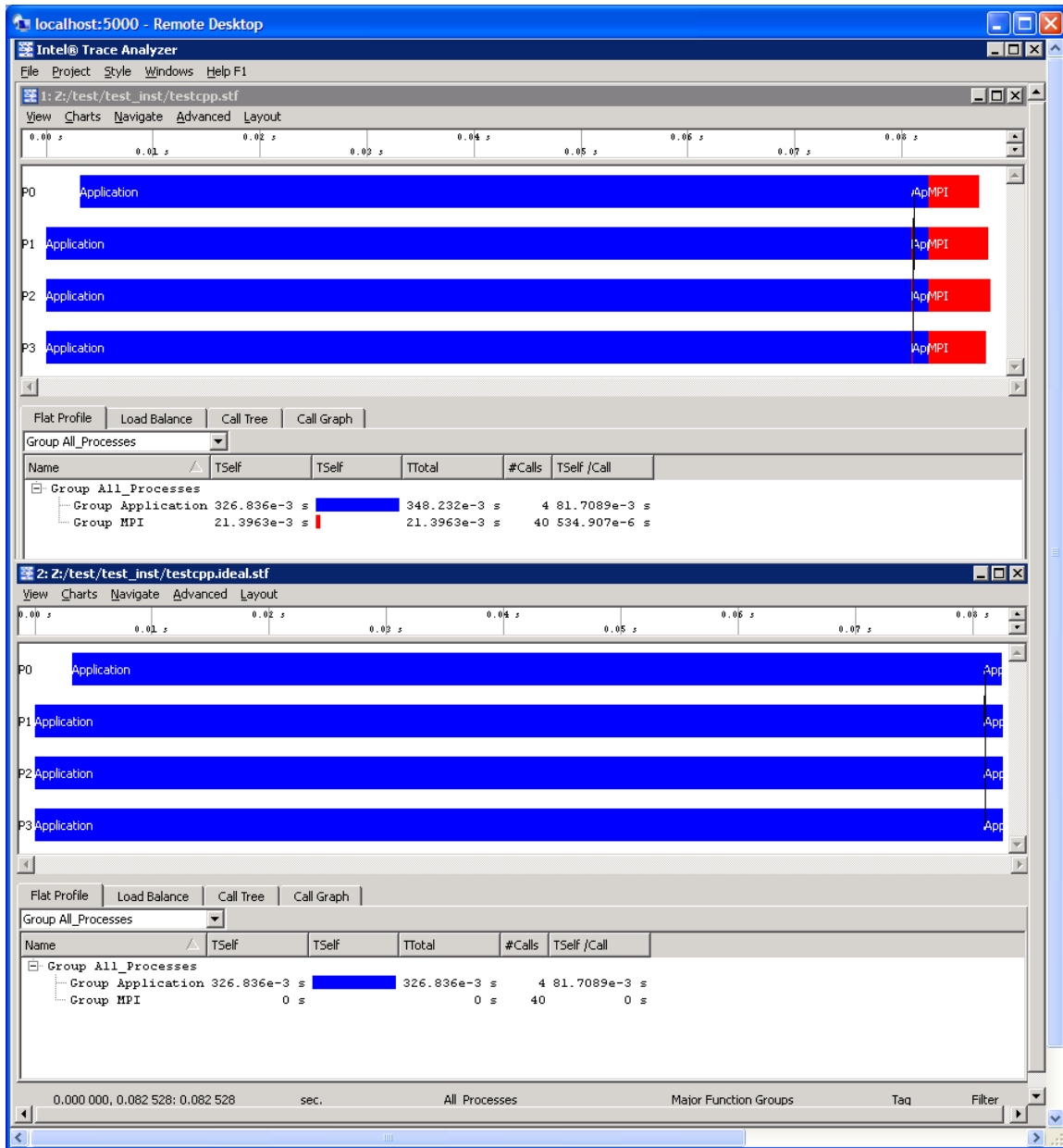


Figure 7.20 – Comparison of the actual execution trace versus the idealized trace for the application `test.cpp`

NOTE: In Figure 7.20, the cost of doing message passing in the ideal case is negligible. You can use the data from the ideal case to help gauge the type of tuning performance that should be pursued.

For complete details on application imbalance, please see Section 5.3 of the Intel® Trace Analyzer Reference Guide, which is titled “Trace Idealizer Dialog Box”. The path to this file is:

`<directory-path-to-ITAC>/doc/ITA_Reference_Guide.pdf`

on the system where the Intel® Trace Analyzer and Collector is installed.

[Back to Table of Contents](#)

7.9 Building a Simulator with the Custom Plug-in Framework

The Intel® Trace Analyzer and Collector provides you with a custom plug-in API that allows you to write your own simulator. You can find the simulator API in the folder path:

`<directory-path-to-ITAC>/examples/icpf/`

on the system where the Intel® Trace Analyzer and Collector is installed. The API source file within the subfolder `icpf` is called `h_devsim.cpp`. For background on building a customer simulator for trace files, please see Chapter 9 of the Intel® Trace Analyzer Reference Guide, which is titled “Custom Plug-in Framework”. The path to this file is:

`<directory-path-to-ITAC>/doc/ITA_Reference_Guide.pdf`

[Back to Table of Contents](#)

8. Getting Started in Using the Intel® Math Kernel Library (Intel® MKL)

On Linux-based platforms, the installation process for Intel MKL on the cluster system will produce a sub-directory that looks something like `.../mkl` where the build number `037` may vary. The default directory path for the library installation process is:

```
/opt/intel/icsxe/2012.0.037/mkl
```

The contents of the `.../mkl` sub-directory should be:

```
benchmarks/  
bin/  
examples/  
include/  
interfaces/  
lib/  
tests/  
tools/
```

Complete user documentation for Intel Math Kernel Library 10.3 Update 6 can be found within the directory path:

```
<directory-path-to-mkl>/doc
```

where `<directory-path-to-mkl>` is the absolute directory path to where the Intel MKL files and sub-directories are installed on the cluster system.

To experiment with the ScaLAPACK test suite, recursively copy the contents of the directory path:

```
<directory-path-to-mkl>/tests/scalapack
```

to a scratch directory area which is sharable by all of the nodes of the cluster. In the scratch directory, issue the command:

```
cd scalapack
```

You can type the command:

```
gmake libem64t mpi=intelmpi LIBdir=<directory-path-to-mkl>/lib/intel64
```

NOTE: The `gmake` command above is applicable to Intel® 64 processor-based systems. This makefile creates and runs executables for the ScaLAPACK* (SCALable LAPACK) examples.

```
<directory-path-to-mkl>/tests/scalapack/source/TESTING
```

Finally, for IA-32 architectures, the `gmake` command might be:

```
gmake libia32 mpi=intelmpi LIBdir=<directory-path-to-mkl>/lib/ia32
```

In the `scalapack` working directory where the `gmake` command was issued, the ScaLAPACK executables can be found in `source/TESTING`, and the results of the computation will be placed into a sub-directory called `_results`. The `_results` directory will be created in same directory from which the `gmake` command was launched. Within this folder is another sub-folder which has a naming convention that uses the following makefile variable configuration:

```
_${arch}_${mpi}_${comp}_${opt}_${ADD_IFACE}
```

For example, on Intel® 64 architecture, using Intel MPI Library 4.0, the Intel compiler and no compiler optimization, the sub-directory under `_results` might be called:

```
_libintel64_intelmpi_intel_noopt_lp64
```

The `*.txt` files for the execution results can be found here. You can invoke an editor to view the results in each of the `*.txt` files that have been created.

As an example result, the file

`"_results/_libintel64_intelmpi_intel_noopt_lp64/cdtlu.txt"` might have something like the following in terms of contents for an execution run on a cluster using four MPI processes. The cluster that generated this sample output consisted of four nodes. The text file was generated by the corresponding executable `xcdtlu`.

SCALAPACK banded linear systems.
'MPI machine'

Tests of the parallel complex single precision band matrix solve
The following scaled residual checks will be computed:

Solve residual = $\frac{\|Ax - b\|}{(\|x\| * \|A\| * \text{eps} * N)}$
Factorization residual = $\frac{\|A - LU\|}{(\|A\| * \text{eps} * N)}$
The matrix A is randomly generated for each test.

An explanation of the input/output parameters follows:

TIME : Indicates whether WALL or CPU time was used.
N : The number of rows and columns in the matrix A.
bwl, bwu : The number of diagonals in the matrix A.
NB : The size of the column panels the matrix A is split into. [-1 for default]
NRHS : The total number of RHS to solve for.
NBRHS : The number of RHS to be put on a column of processes before going on to the next column of processes.
P : The number of process rows.
Q : The number of process columns.
THRESH : If a residual value is less than THRESH, CHECK is flagged as PASSED
Fact time: Time in seconds to factor the matrix
Sol Time: Time in seconds to solve the system.
MFLOPS : Rate of execution for factor and solve using sequential operation count.
MFLOP2 : Rough estimate of speed using actual op count (accurate big P,N).

The following parameter values will be used:

N : 3 5 17
bwl : 1
bwu : 1
NB : -1
NRHS : 4
NBRHS: 1
P : 1 1 1 1
Q : 1 2 3 4

Relative machine precision (eps) is taken to be 0.596046E-07
Routines pass computational tests if scaled residual is less than 3.0000

TIME	TR	N	BWL	BWU	NB	NRHS	P	Q	L*U	Time	Slv	Time	MFLOPS	MFLOP2	CHECK
WALL	N	3	1	1	3	4	1	1	0.000	0.0001	1.06	1.00	PASSED		
WALL	N	5	1	1	5	4	1	1	0.000	0.0001	1.75	1.66	PASSED		
WALL	N	17	1	1	17	4	1	1	0.000	0.0001	6.10	5.77	PASSED		
WALL	N	3	1	1	2	4	1	2	0.000	0.0003	0.36	0.53	PASSED		
WALL	N	5	1	1	3	4	1	2	0.000	0.0002	0.90	1.35	PASSED		
WALL	N	17	1	1	9	4	1	2	0.000	0.0002	3.03	4.59	PASSED		
WALL	N	3	1	1	2	4	1	3	0.001	0.0006	0.19	0.27	PASSED		
WALL	N	5	1	1	2	4	1	3	0.001	0.0010	0.17	0.30	PASSED		
WALL	N	17	1	1	6	4	1	3	0.001	0.0010	0.75	1.16	PASSED		
WALL	N	3	1	1	2	4	1	4	0.001	0.0007	0.17	0.24	PASSED		
WALL	N	5	1	1	2	4	1	4	0.002	0.0026	0.08	0.13	PASSED		
WALL	N	17	1	1	5	4	1	4	0.001	0.0011	0.66	1.00	PASSED		

Finished 12 tests, with the following results:
12 tests completed and passed residual checks.
0 tests completed and failed residual checks.
0 tests skipped because of illegal input values.

END OF TESTS.

The text in the table above reflects the *organization* of actual output that you will see.

Recall from Intel MPI Library and Intel Trace Analyzer and Collector discussions that the above results are dependent on factors such as the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster. Therefore, the results will vary from one cluster configuration to another.

If you proceed to load the `cdtlu.txt` table above into a Microsoft Excel* spreadsheet, and build a chart to compare the Time in Seconds to Solve the System (SLV) and the Megaflop values, you might see something like the following (Figure 8.1):

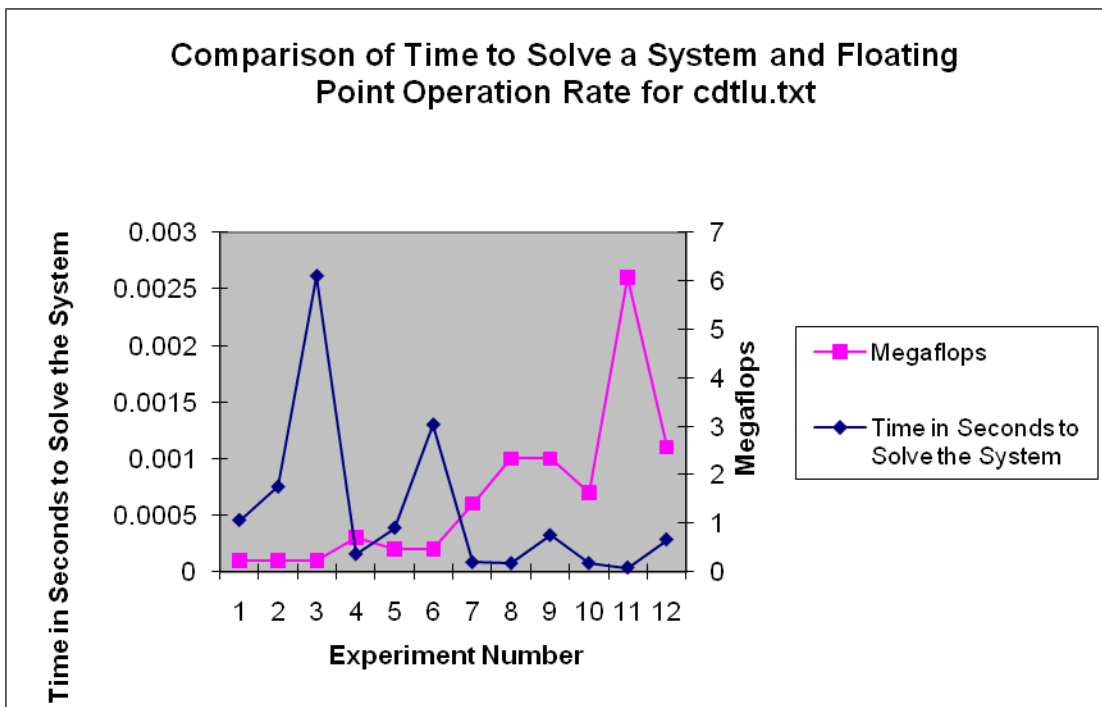


Figure 8.1 – Display of ScaLAPACK DATA from the executable `xcdtlu`

[Back to Table of Contents](#)

8.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK* Examples with the Intel® Trace Analyzer and Collector

In the chapter entitled Interoperability of Intel MPI Library with the Intel® Trace Analyzer and Collector, cursory explanations were provided in gathering trace data and opening various analyzer panels for viewing trace-file content. Analysis of the ScaLAPACK examples with Intel Trace Collector and Intel Trace Analyzer can also be done easily. This subsection will dwell further on the instrumentation and analysis process. The discussion will focus on how to alter the command-line options for the ScaLAPACK `gmake` command so that performance data collection will be possible. However, you will want to have plenty of disk storage available for collecting trace information on all of the examples because there are approximately 68 ScaLAPACK executables. To instrument the ScaLAPACK examples on an IA-32 cluster that is running Linux OS, you could use the following `gmake` command:

```
gmake libia32 mpi=intelmpi
LIBdir=/opt/intel/icsxe/2012.0.037/mkl/lib/ia32 INSLIB="-L${VT_LIB_DIR}
-lVT ${VT_ADD_LIBS}"
```

Finally, for the Intel® 64 architecture, the `gmake` command for gathering ScaLAPACK instrumentation data on Linux could possibly be:

```
gmake libintel64 mpi=intelmpi
LIBdir=/opt/intel/icsxe/2012.0.037/mkl/lib/intel64 INSLIB="-
L${VT_LIB_DIR} -lVT ${VT_ADD_LIBS}"
```

For all two command-line examples listed above, the make file variable `INSLIB` is used to specify the library path name and the libraries used for instrumentation by the Intel® Trace Collector. The variable name `INSLIB` is simply an acronym for instrumentation library.

Recall the instrumentation processes discussed in Chapter 6. The recommended amount of disk storage for collecting trace data on all of the ScaLAPACK test cases is about 5 gigabytes. For an executable such as `_results/_libintel64_intelmpi_intel_noopt_lp64/xzvec` located in `source/TESTING` that has been instrumented with the Intel Trace Collector, a trace file called `xzvec.stf` will be generated. For the `gmake` commands above, the STF files will also be located in the sub-directory path `source/TESTING` and the summary reports for each ScaLAPACK executable will be placed under a sibling directory path to `source` called `_results`. Recalling the protocol that was discussed in the chapter for using Intel Trace Analyzer, you can proceed to analyze the content of `xzvec.stf` with the following shell command:

```
traceanalyzer xzvec.stf &
```

This command for invoking the Intel Trace Analyzer will cause the Event Timeline Chart and the Function Profile Chart (Figure 8.2) to be produced as described previously:

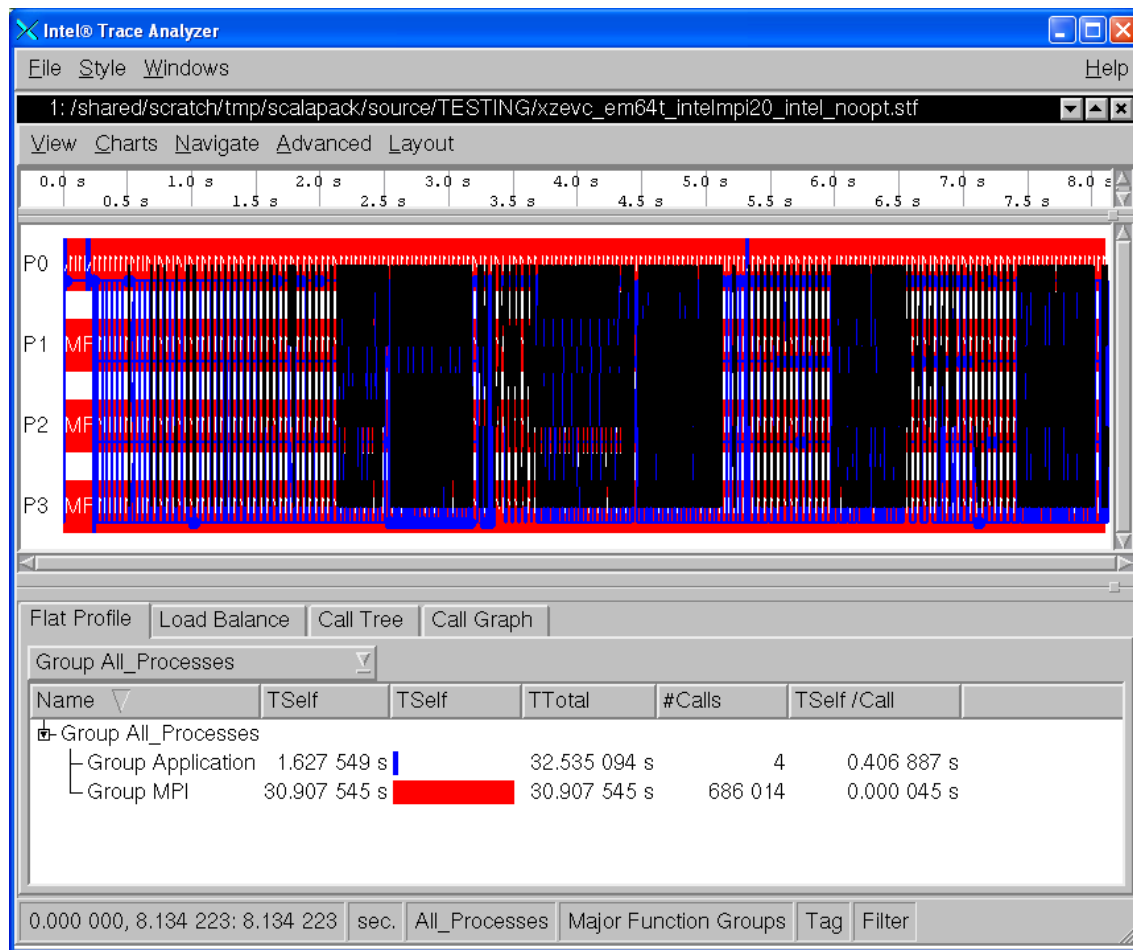


Figure 8.2 – Event Timeline Chart and the Function Profile Chart for the executable_results/_libintel64_intelmpi_intel_noopt_lp64/xzevc

By default, the ScaLAPACK makefile uses four MPI processes. If you wish to decrease or increase the number of MPI processes, you can adjust the `MPRUN` makefile variable. An example for doing this on a system based on Intel® 64 architecture might be the following:

```
gmake libintel64 mpi=intelmpi
LIBdir=/opt/intel/icsxe/2012.0.037/mkl/lib/intel64 MPILIB="-
L${VT_LIB_DIR} -lVT ${VT_ADD_LIBS}" MPRUN="mpiexec -n 6"
```

You should again realize that the contents of a trace file such as `source/TESTING/xzevc.stf` will vary from cluster configuration to cluster configuration due to factors such as the processor type, the memory configuration,

competing processes, and the type of interconnection network between the nodes of the cluster.

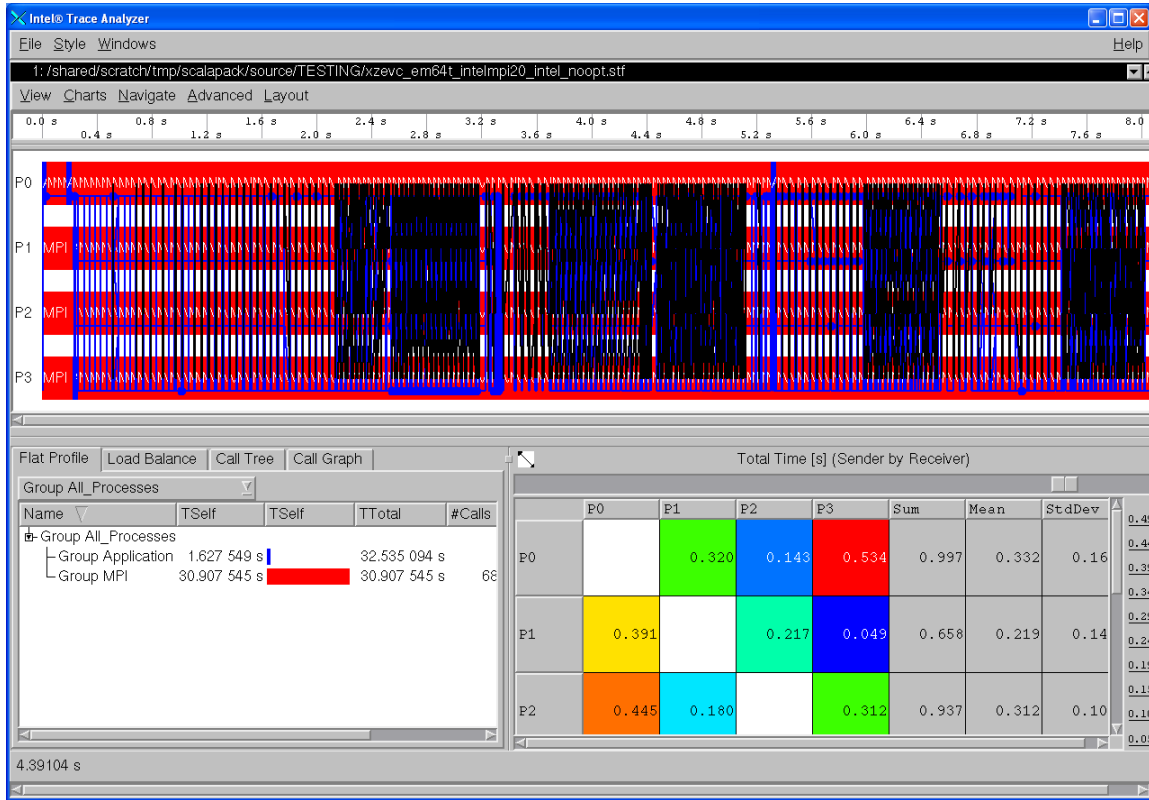


Figure 8.3 – The Message Profile Chart (lower right) for the executable `_results/_libintel64_intelmpi_intel_noopt_lp64/xzvc`

If you proceed to select **Charts->Message Profile**, you will generate the Message Profile Chart shown in Figure 8.3. Subsequently, if **Charts->Collective Operations Profile** is selected, then the chart shown in Figure 8.4 will be produced.

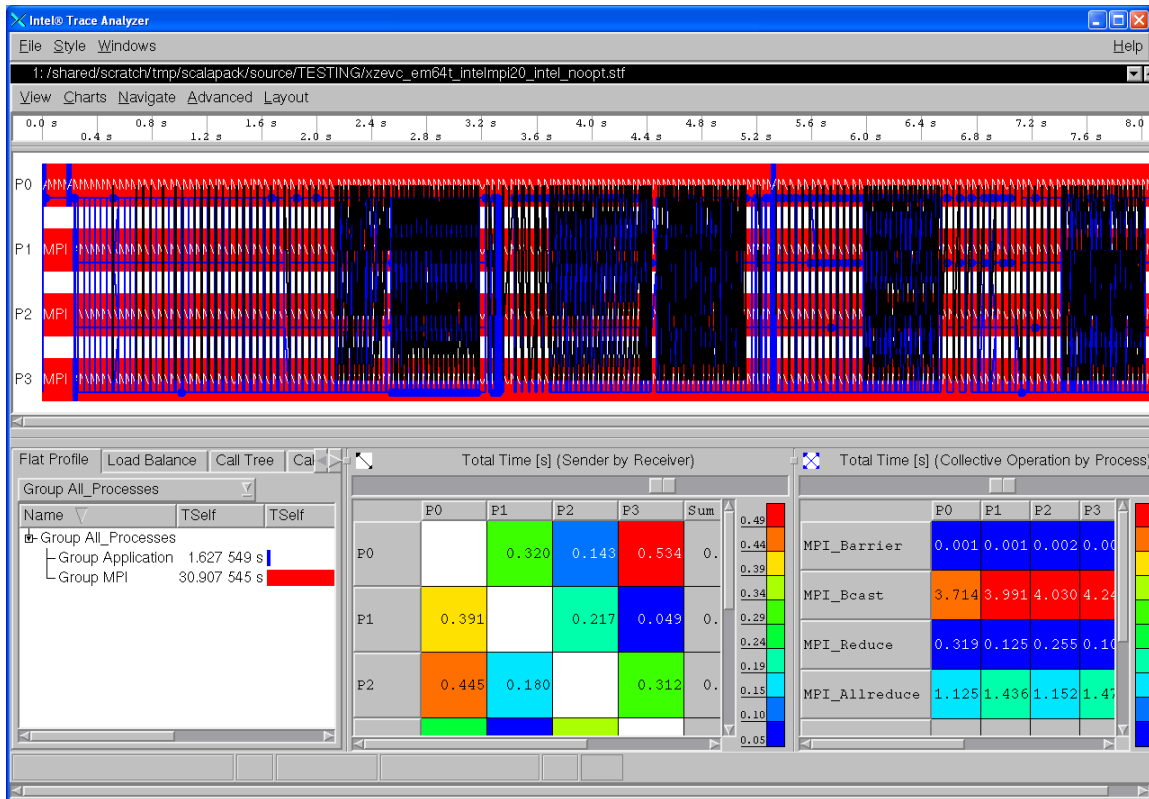


Figure 8.4 – Display of the Collective Operations Profile Chart (lower right) for `_results/_libintel64_intelmpi_intel_noopt_lp64/xzvc`

You can zoom in on a particular time interval for the Event Timeline Chart in Figure 8.4. Clicking on the left-most mouse button and panning across the desired time interval will cause the zoom in function. For example, Figure 8.5 shows zooming in to the time interval which spans from approximately 3.0 seconds to approximately 3.01 seconds.

NOTE: The number of message lines that are shown in black in Figure 8.5 is significantly reduced with respect to Figure 8.4.

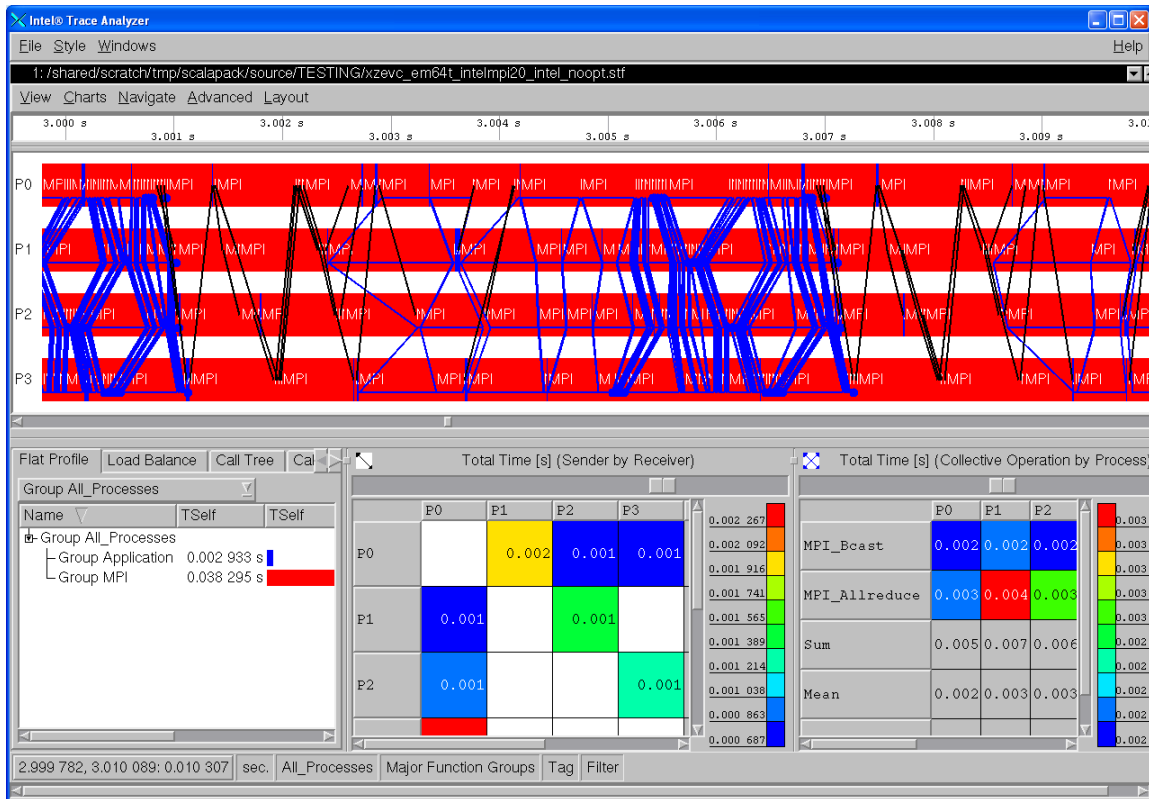


Figure 8.5 – Zooming in on the Event Timeline Chart for example `_results/_libintel64_intelmpi_intel_noopt_lp64/xzevc`

For Figure 8.5, the blue collective operation communication lines can be “drilled-down-to” by using the context menu as shown in Figure 8.6 to view the collective operation.

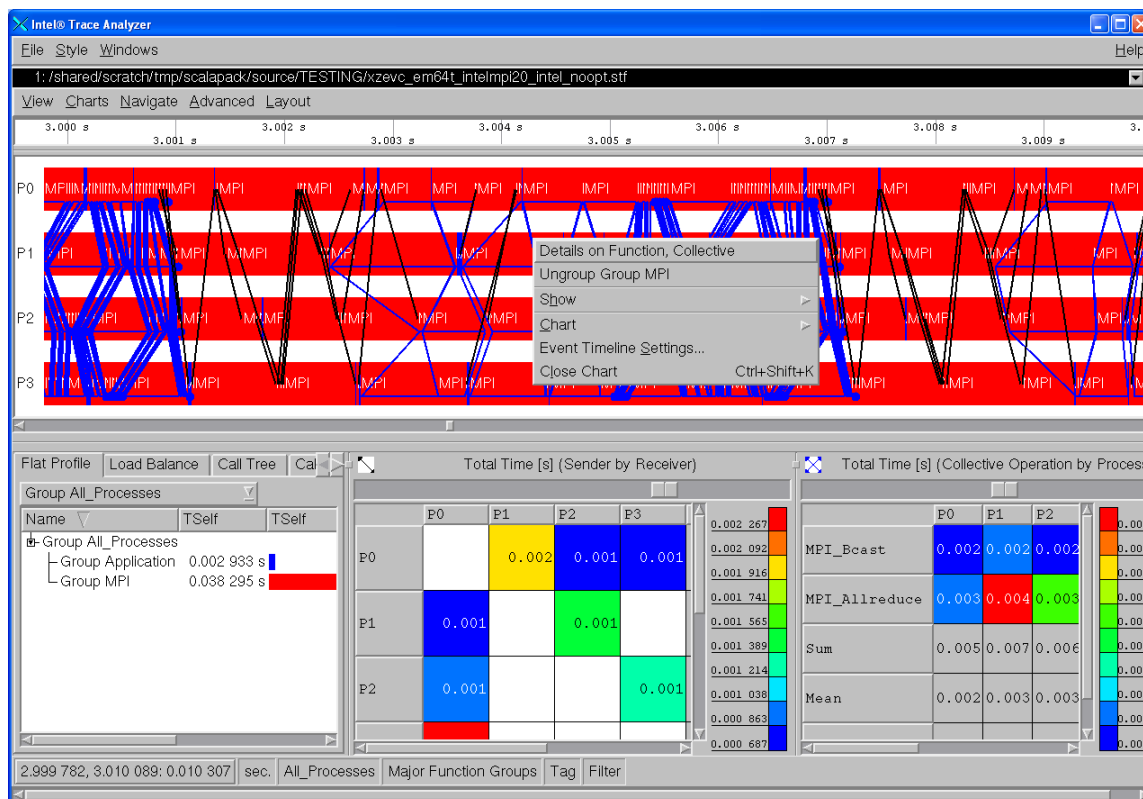


Figure 8.6 – Context Menu Selection for starting the process of drilling down to what the particular collective operation was executing (e.g. MPI_Allreduce) within the executable `_results/_libintel64_intelmpi_intel_noopt_lp64/xzevc`

NOTE: If you would like to do a drill-down to actual source, the source files used to build the executables would have to be compiled with the `-g` option, and the Intel Trace Collector `VT_PCTRACE` environment variable would have to be set. For the ScaLAPACK `gmake` command, you might set the `-g` option with the following makefile variable:

```
OPTS="-O0 -g"
```

[Back to Table of Contents](#)

8.2 Experimenting with the Cluster DFT Software

On Linux OS, in the directory path:

```
<directory-path-to-mkl>/examples
```

you will find a set of sub-directories that look something like:

```
./          cdftc/      fftw2x_cdft/  interval/    pdepoisson/  versionquery/
```

```

../      cdftf/   fftw2xf/   java/     pdettc/   vmlc/
blas/    dftc/     fftw3xc/   lapack/   pdettf/   vmlf/
blas95/  dftf/     fftw3xf/   lapack95/ solver/    vslc/
cblas/   fftw2xc/  gmp/       pdepoissonc/ spblas/   vslf/

```

The two sub-directories that will be discussed here are `cdftc` and `cdftf`. These two directories respectively contain C and Fortran programming language examples of the Cluster Discrete Fourier Transform (CDFT). To do experimentation with the contents of these two folders, a sequence of shell commands could be used to create instrumented executables and result information. For the C language version of the CDFT, the Bourne Shell or Korn Shell commands might look something like:

Intel Processor Architecture	Command-line Sequence for Linux	Trace Results are Located In	Execution Results are Located In
IA-32	<pre> #!/bin/sh export CWD=\${PWD} export VT_LOGFILE_PREFIX=\${CWD}/cdftc_inst rm -rf \${VT_LOGFILE_PREFIX} mkdir \${VT_LOGFILE_PREFIX} export VT_PCTRACE=5 export VT_DETAILED_STATES=5 cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftc gmake libia32 mpi=intel3 workdir=\${VT_LOGFILE_PREFIX} CS="mpiicc -t=log" RS="mpiexec -n 4" RES_DIR=\${VT_LOGFILE_PREFIX} </pre>	<pre> \${CWD}/cdftc_inst </pre>	<pre> \${CWD}/cdftc_inst </pre>
Intel® 64	<pre> #!/bin/sh export CWD=\${PWD} export VT_LOGFILE_PREFIX=\${CWD}/cdftc_inst rm -rf \${VT_LOGFILE_PREFIX} mkdir \${VT_LOGFILE_PREFIX} export VT_PCTRACE=5 export VT_DETAILED_STATES=5 cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftc gmake libintel64 mpi=intel3 workdir=\${VT_LOGFILE_PREFIX} CS="mpiicc -t=log" RS="mpiexec -n 4" RES_DIR=\${VT_LOGFILE_PREFIX} </pre>	<pre> \${CWD}/cdftc_inst </pre>	<pre> \${CWD}/cdftc_inst </pre>

where `<directory-path-to-mkl>/examples` in the shell command-sequence above is:

```

/usr/local/opt/intel/icsxe/2012.0.037/mkl/examples

```

NOTE: The folder path above will vary depending on where the Intel Cluster Studio XE was installed on your system. The change directory command above (for example, `cd ...`) transfers the Bourne Shell or Korn Shell session to:

```
/usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftc
```

The `gmake` command for the target `lib32` is one contiguous line that ends with `CS="mpiicc -t=log"`. This command references the makefile variables `libia32`, `mpi`, `workdir`, `CS`, and `RS`. As mentioned above, the target for the `gmake` command is `libia32`. The other target of this type is `libintel64`. The target `libintel64` is for Intel® 64 architecture. The makefile variable `CS` is set so that the resulting executable is linked against the logging versions of Intel MPI and the Intel Trace Collector. The `RS` makefile variable allows you to control the number of MPI processes. The default for `RS` is `"mpiexec -n 2"` when using Intel MPI Library. You can get complete information about this makefile by looking at its contents. There is also a `help` target built within the makefile, and therefore you can type:

```
gmake help
```

Assuming that `${CWD}` has been defined from above for the Fortran language version of the CDFT, the Bourne Shell or Korn Shell commands might look something like:

Intel Processor Architecture	Command-line Sequence for Linux	Trace Results are Located In	Execution Results are Located In
IA-32	<pre>export VT_LOGFILE_PREFIX=\${CWD}/cdftf_inst rm -rf \${VT_LOGFILE_PREFIX} mkdir \${VT_LOGFILE_PREFIX} export VT_PCTRACE=5 export VT_DETAILED_STATES=5 cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftf gmake libia32 mpi=intel3 workdir=\${VT_LOGFILE_PREFIX} CS="mpiifort -t=log -DMPI_KIND=4" RS="mpiexec -n 4" RES_DIR=\${VT_LOGFILE_PREFIX}"</pre>	<code>\${CWD}/cdftf_inst</code>	<code>\${CWD}/cdftf_inst</code>
Intel® 64	<pre>export VT_LOGFILE_PREFIX=\${CWD}/cdftf_inst rm -rf \${VT_LOGFILE_PREFIX} mkdir \${VT_LOGFILE_PREFIX} export VT_PCTRACE=5 export VT_DETAILED_STATES=5 cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftf gmake libintel64 mpi=intel3 workdir=\${VT_LOGFILE_PREFIX} CS="mpiifort -t=log -DMPI_KIND=4" RS="mpiexec -n 4"</pre>	<code>\${CWD}/cdftf_inst</code>	<code>\${CWD}/cdftf_inst</code>

RES_DIR=\${VT_LOGFILE_PREFIX}		
-------------------------------	--	--

If you consolidate the shell script commands for performing C and Fortran Cluster Discrete Fourier computation on a particular Intel processor architecture, say Intel® 64 architecture, the complete Bourne shell script content might look something like:

```
#!/bin/sh
export CWD=${PWD}
export VT_LOGFILE_PREFIX=${CWD}/cdftc_inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
export VT_DETAILED_STATES=5
cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftc
gmake libintel64 mpi=intel3 workdir=${VT_LOGFILE_PREFIX} CS="mpiicc -
t=log" RS="mpiexec -n 4" RES_DIR=${VT_LOGFILE_PREFIX}
export VT_LOGFILE_PREFIX=${CWD}/cdftf_inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
export VT_DETAILED_STATES=5
cd /usr/local/opt/intel/icsxe/2012.0.037/mkl/examples/cdftf
gmake libintel64 mpi=intel3 workdir=${VT_LOGFILE_PREFIX} CS="mpiifort -
t=log -DMPI_KIND_4" RS="mpiexec -n 4" RES_DIR=${VT_LOGFILE_PREFIX}
```

After executing the shell script above, the `${CWD}/cdftc_inst` and `${CWD}/cdftf_inst` folders should contain the respective executables and the output results. The executable and result contents of each folder path might look something like:

```
dm_complex_2d_double_ex1.exe
dm_complex_2d_double_ex2.exe
dm_complex_2d_single_ex1.exe
dm_complex_2d_single_ex2.exe
```

and

```
dm_complex_2d_double_ex1.res
dm_complex_2d_double_ex2.res
dm_complex_2d_single_ex1.res
dm_complex_2d_single_ex2.res
```

The files with the suffix `.res` are the output results. A partial listing for results file called `dm_complex_2d_double_ex1.res` might be something like:

```
Program is running on 4 processes

DM_COMPLEX_2D_DOUBLE_EX1
Forward-Backward 2D complex transform for double precision data inplace

Configuration parameters:
```

```

DFTI_FORWARD_DOMAIN = DFTI_COMPLEX
DFTI_PRECISION      = DFTI_DOUBLE
DFTI_DIMENSION      = 2
DFTI_LENGTHS (MxN) = {20,12}
DFTI_FORWARD_SCALE  = 1.0
DFTI_BACKWARD_SCALE = 1.0/(m*n)
...

Compute DftiComputeForwardDM

Forward result X, 4 columns

Row 0:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 1:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 2:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 3:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 4:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 5:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 6:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 7:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
Row 8:
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)( 1.000, 0.000)
...

```

Also, the setting of the environment variable `VT_LOGFILE_PREFIX` within the shell script results in the deposit of trace information into the directories `cdftc_inst` and `cdftf_inst` as demonstrated with a listing of the Structured Trace Format (STF) index files:

```

cdftc_inst/dm_complex_2d_double_ex1.exe.stf
cdftc_inst/dm_complex_2d_double_ex2.exe.stf
cdftc_inst/dm_complex_2d_single_ex1.exe.stf
cdftc_inst/dm_complex_2d_single_ex2.exe.stf

```


and

```
cdftf_inst/dm_complex_2d_double_ex1.exe.stf  
cdftf_inst/dm_complex_2d_double_ex2.exe.stf  
cdftf_inst/dm_complex_2d_single_ex1.exe.stf  
cdftf_inst/dm_complex_2d_single_ex2.exe.stf
```

You can issue the following Intel Trace Analyzer shell command to initiate performance analysis on `cdftc_inst/dm_complex_2d_double_ex1.exe.stf`:

```
traceanalyzer ./cdftc_inst/dm_complex_2d_double_ex1.exe.stf &
```

Figure 8.7 shows the result of simultaneously displaying the Function Profile Chart and the Event Timeline Chart.

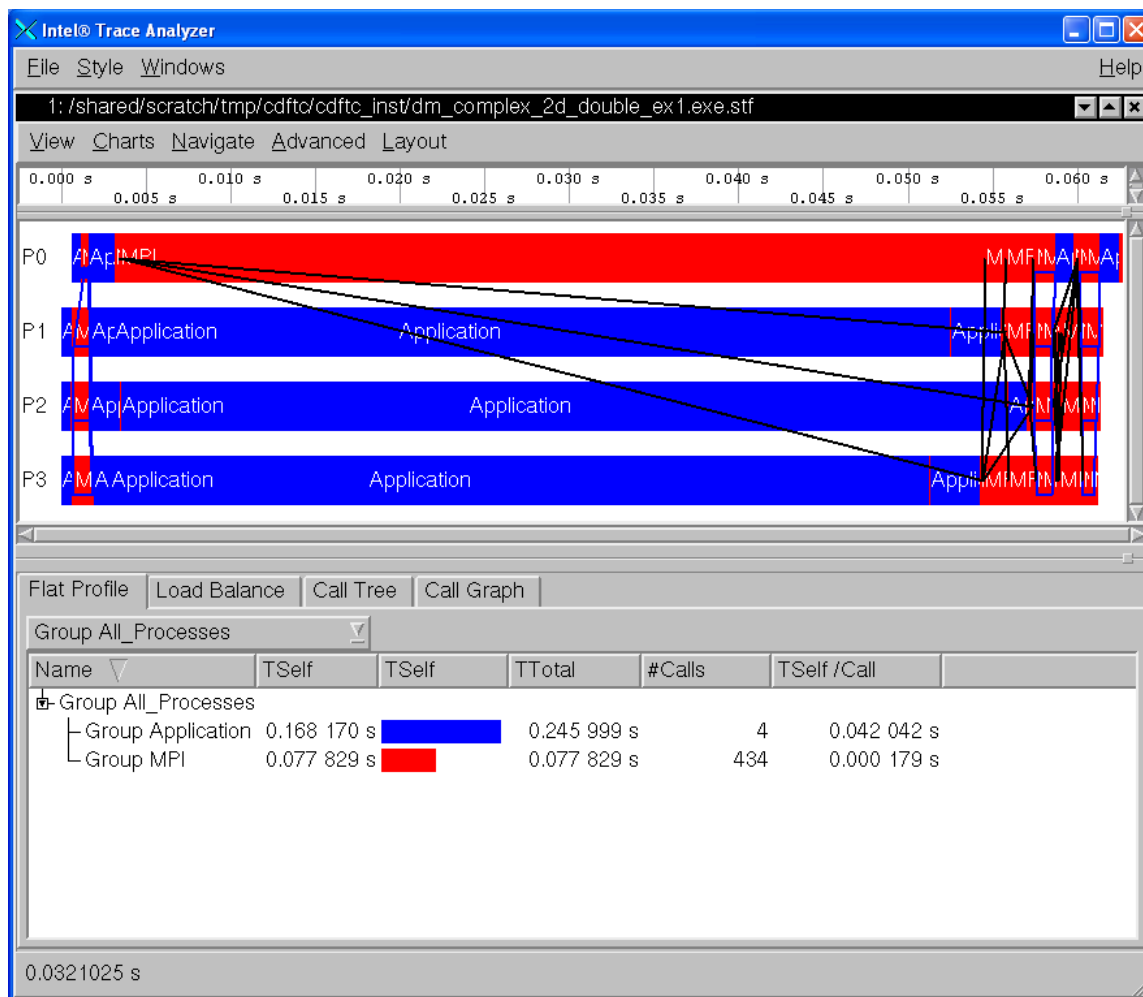


Figure 8.7 – The Event Timeline Chart and the Function Profile Chart for a Cluster Discrete Fourier Transform Example

[Back to Table of Contents](#)

8.3 Experimenting with the High Performance Linpack Benchmark*

On Linux OS, in the directory path:

`<directory-path-to-mkl>/benchmarks/mp_linpack`

you will find a set of files and subdirectories that look something like the following:

```
./          BUGS*          include/    Makefile*  Make.top*  setup/     TUNING*
../        COPYRIGHT*  INSTALL*   Make.ia32* man/        src/       www/
bin/       HISTORY*      lib/       Make.intel64* nodeperf.c* testing/
bin_intel/ HPL.build.log.220120040613* lib_hybrid/ makes/     README*   TODO*
```

If you make a scratch directory, say:

```
test_mp_linpck
```

on a file share for your cluster, and copy the contents of *<directory-path-to-mkl>/benchmarks/mp_linpck* into that scratch directory you can then proceed to build a High Performance Linpack executable. To create an executable for Intel® 64 architecture, you might issue the following `gmake` command:

```
gmake arch=intel64
LAdir=/usr/local/opt/intel/icsxe/2012.0.037/mkl/lib/intel64
LAinc=/usr/local/opt/intel/icsxe/2012.0.037/mkl/include
```

where the command sequence above is one continuous line. The macros `LAdir` and `LAinc` describe the directory path to the Intel® 64 Math Kernel library and the Intel® MKL include directory, respectively. The partial directory path `/usr/local/opt/intel/icsxe/2012.0.037` for the macros `LAdir` and `LAinc` should be considered an *example* of where an Intel® Math Kernel Library might reside.

NOTE: On your system, the path and a version number value such as `2012.0.037` may vary depending on *your* software release.

The High Performance Linpack* executable for the `gmake` command above will be placed into `.../test_mp_linpck/bin/intel64` and will be called `xhpl`. The table below summarizes makefile and associated `mpiexec` commands that might be used to create `xhpl` executables for IA-32, and Intel® 64 architectures, respectively. The command-line syntax in the table is that of Bourne* Shell or Korn* Shell. The `mpiexec` commands use 4 MPI processes to do the domain decomposition.

Intel Processor Architecture	Command-line Sequence for Linux	Executable is Located In	Execution Results are Located In
IA-32	#!/bin/sh export CWD=\${PWD} gmake clean_arch_all arch=ia32 gmake arch=ia32 LAdir=/usr/local/opt/intel/icsxe/2012.0.037/mkl/lib/ia32 LAinc=/usr/local/opt/intel/icsxe/2012.0.037/mkl/include cd \${CWD}/bin/ia32 mpiexec -n 4 ./xhpl > results.ia32.out	\${CWD}/bin/ia32	\${CWD}/bin/ia32
Intel® 64	#!/bin/sh export CWD=\${PWD} gmake clean_arch_all arch=intel64 gmake arch=intel64 LAdir=/usr/local/opt/intel/icsxe/2012.0.037/mkl/lib/intel64 LAinc=/usr/local/opt/intel/icsxe/2012.0.037/mkl/include cd \${CWD}/bin/intel64 mpiexec -n 4 ./xhpl > results.em64t.out	\${CWD}/bin/intel64	\${CWD}/bin/intel64

The output results *might* look something like the following for Intel® 64 architecture:

```

=====
HPLinpack 2.0 -- High-Performance Linpack benchmark -- September 10, 2008
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/V   : Wall time / encoded variant.
N     : The order of the coefficient matrix A.
NB    : The partitioning blocking factor.
P     : The number of process rows.
Q     : The number of process columns.
Time  : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :      1000
NB     :      112      120
PMAP   : Row-major process mapping
P      :      1      2      1      4
Q      :      1      2      4      1
PFACT  :      Left
NBMIN  :      4      2
NDIV   :      2
RFACT  :      Crout

```

```

BCAST : lring
DEPTH : 0
SWAP : Mix (threshold = 256)
L1 : no-transposed form
U : no-transposed form
EQUIL : no
ALIGN : 8 double precision words

```

...

```

=====
T/V          N    NB    P    Q          Time          Gflops
-----
WR00C2L2    1000  120    4    1          0.35          1.894e+00
=====
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 0.0052671 ..... PASSED
=====

```

```

Finished      16 tests with the following results:
              16 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.

```

```

-----
End of Tests.
=====

```

The file `<directory-path-to-mkl>/doc/mkl_documentation.htm` contains a landing page linking various documentation files associated with Intel MKL 10.3 Update 6. To make inquiries about Intel Math Kernel Library 10.3 Update 6, visit the URL: <http://premier.intel.com>.

[Back to Table of Contents](#)

9. Using the Intel® MPI Benchmarks

The Intel MPI Benchmarks have been ported to Linux* OS. The directory structure for the Intel® MPI Benchmarks 3.2.3 looks something like the following where the parenthesized text contains descriptive information:

- ./doc (ReadMe_IMB.txt; IMB_Users_Guide.pdf, the methodology description)
- ./src (program source code and Makefiles)
- ./license (Source license agreement, trademark and use license agreement)
- ./versions_news (version history and news)
- ./WINDOWS (Microsoft* Visual Studio* projects)

The WINDOWS folder as noted above contains Microsoft* Visual Studio* 2005 and 2008 project folders which allow you to use a pre-existing ".vcproj" project file in conjunction with Microsoft* Visual Studio* to build and run the associated Intel® MPI Benchmark application. This is not relevant to Linux* OS.

If you type the command `gmake` within the `src` subdirectory, then you will get general help information that looks something like the following:

```
IMB_3.2 does not have a default Makefile any more.  
This Makefile can be used to
```

```
gmake clean
```

For installing, please use:

```
gmake -f make_ict
```

```
to install the Intel(r) Cluster Tools (ict) version.  
When an Intel(r) MPI Library install and mpiicc path exists,  
this should work immediately.
```

Alternatively, use

```
gmake -f make_mpich
```

```
to install an mpich or similar version; for this,  
you normally have to edit at least the MPI_HOME  
variable provided in make_mpich
```

To clean up the directory structure, in the directory `src`, type:

```
gmake clean
```

To compile the Intel MPI Benchmarks with the Intel Cluster Tools, type the command:

```
gmake -f make_ict
```

The three executables that will be created with the all target are:

```
IMB-EXT  
IMB-IO  
IMB-MPI1
```

Assuming that you have a four node cluster, and the Bourne Shell is being used type the commands:

```
mpiexec -n 4 IMB-EXT > IMB-EXT.report 2>&1  
mpiexec -n 4 IMB-IO > IMB-IO.report 2>&1  
mpiexec -n 4 IMB-MPI1 > IMB-MPI1.report 2>&1
```

Similarly, if C Shell is the command-line interface, type the commands:

```
mpiexec -n 4 IMB-EXT >&! IMB-EXT.report  
mpiexec -n 4 IMB-IO >&! IMB-IO.report  
mpiexec -n 4 IMB-MPI1 >&! IMB-MPI1.report
```

[Back to Table of Contents](#)

10. Uninstalling the Intel® Cluster Studio XE on Linux* OS

To uninstall the Intel Cluster Studio XE from a Linux OS, you can use a shell script called `uninstall.sh`. This script can be found in folder path:

```
<Path-to-Intel-Cluster-Studio-XE>/uninstall.sh
```

An example folder might be:

```
/usr/local/opt/intel/icsxe/2012.0.037/uninstall.sh
```

When this uninstall script is invoked, it will prompt you for that location of the `machines.LINUX` file.

The uninstall script has command-line options. Type a shell command referencing `uninstall.sh` such as:

```
uninstall.sh --help | less
```

You will see a list of options that look something like:

NAME

```
uninstall.sh - Uninstall Intel(R) Cluster Studio XE 2012 for Linux*.
```

SYNOPSIS

```
uninstall.sh [options]
```

Copyright(C) 1999-2011, Intel Corporation. All Rights Reserved.

[Back to Table of Contents](#)

11. Hardware Recommendations for Installation on Linux* OS

Processor System Requirements

Intel® Pentium® 4 processor, or
Intel® Xeon® processor, or
Intel® Core™2 Duo processor (example of Intel® 64 architecture)

NOTE: It is assumed that the processors listed above are configured into homogeneous clusters.

Disk-Space Requirements

20 GBs of disk space (minimum)

NOTE: During the installation process, the installer may need approximately 4 gigabytes of temporary disk storage to manage the intermediate installation files.

Operating System Requirements for Linux* OS

OS Distributions	IA-32 Architecture	Intel® 64 Architecture	
		32-Bit Applications	64-Bit Applications
Intel® Cluster Ready ²	N/A	N/A	S
Red Hat Enterprise Linux* 5.0	S	S	S
Red Hat Enterprise Linux* 6.0	S	S	S
SUSE Linux Enterprise Server* 10	S	S	S
SUSE Linux Enterprise Server* 11	S	S	S

S = Supported

² Intel® Cluster Ready* is an applications platform architecture standard for Linux* clusters. Convey to your users the Linux* platform needed for your MPI application with:

This application has been verified to run correctly on Linux* clusters which are conforming to the Intel® Cluster Ready platform architecture. Each Intel® Cluster Ready system is shipped and tested with a diagnostic tool: Intel® Cluster Checker. Intel® Cluster Checker is used to validate operability and compliance, as well as overall system health. On an Intel® Cluster Ready system, start with these commands to easily find out about diagnostic logs:

```
$ . /opt/intel/clck/<version>/clckvars.sh
```

```
$ cluster-check -report
```

For more information on Intel® Cluster Ready, and on the alliance of partner vendors, please visit <http://www.intel.com/go/cluster>.

Memory Requirements

2 GB of RAM (minimum)

[Back to Table of Contents](#)

12. System Administrator Checklist for Linux* OS

Intel license keys should be placed in a common repository for access by the software components of the Intel Cluster Studio XE. An example license directory path might be:

```
/opt/intel/licenses
```

[Back to Table of Contents](#)

13. User Checklist for Linux* OS

1. The Intel® Debugger graphical environment is a Java* application and requires a Java Runtime Environment* (JRE*) to execute. The debugger will run with a version 5.0 (also called 1.5) JRE.

Install the JRE according to the JRE provider's instructions.

Finally you need to export the path to the JRE as follows:

```
export PATH=<path_to_JRE_bin_DIR>:$PATH export
```

2. Configure the environment variables. For the `~/.bashrc` file, an example of setting environment variables and sourcing shell scripts might be the following for Intel® 64 architecture:

```
export INTEL_LICENSE_FILE=/opt/intel/licenses
. /opt/intel/icsxe/2012.0.037/ictvars.sh
```

Alternatively, for `~/.cshrc` the syntax might be something like:

```
setenv INTEL_LICENSE_FILE /opt/intel/licenses
source /opt/intel/icsxe/2012.0.037/ictvars.csh
```

3. For Bourne* Shell on Linux* OS, once the Intel® Cluster Studio XE environment variables referenced within "ictvars.sh" file have been sourced via a `.bashrc` file, users for a given Bourne* Shell login session can simply type:

```
. ictvars.sh ia32
```

for creating IA-32 executables. Alternatively, to restore the default Intel® Cluster Studio XE environment variable settings so as to build executables with Intel® 64 address extensions, type:

```
. ictvars.sh
```

within the Bourne* Shell login session.

NOTE: The full path to `ictvars.sh` can be omitted once it has been sourced in the `.bashrc` file.

For a C Shell login session on Linux* OS, IA-32 executables can be created with a login session command such as:

```
source /opt/intel/icsxe/2012.0.037/ictvars.csh ia32
```

Within a C Shell login session, to restore the default Intel® Cluster Studio XE environment variable settings so as to build executables with Intel® 64 address extensions, type:

```
source /opt/intel/icsxe/2012.0.037/ictvars.csh
```

4. When using the Intel Debugger (IDB) with Intel MPI Library, you also want to create or update the `~/.rhosts` file with the names of the nodes of the cluster. The `~/.rhosts` file should have node names that use the following general syntax:

```
<hostname as echoed by the shell command hostname> <your username>
```

The permission bit settings of `~/.rhosts` should be set to 600 using the `chmod` command. The shell command for doing this might be:

```
chmod 600 ~/.rhosts
```

[Back to Table of Contents](#)

14. Using the Compiler Switch *-tcollect*

The Intel® C++ and Intel® Fortran Compilers on Linux OS have the command-line switch called `-tcollect` which allows functions and procedures to be instrumented during compilation with Intel® Trace Collector calls. This compiler command-line switch accepts an optional argument to specify the Intel® Trace Collector library to link with.

Library Selection	Meaning	How to Request
libVT.a	Default library	<code>-tcollect</code>
libVTcs.a	Client-server trace collection library	<code>-tcollect=VTcs</code>
libVTfs.a	Fail-safe trace collection library	<code>-tcollect=Vtfs</code>

Recall once again that in the `test_intel_mpi` folder for Intel MPI Library, there are four source files called:

```
test.c test.cpp test.f test.f90
```

To build executables with the `-tcollect` compiler option for the Intel Compilers, one might use the following compilation and link commands:

```
mpiicc test.c -tcollect -g -o testc_tcollect
mpiicpc test.cpp -g -tcollect -o testcpp_tcollect
mpiifort test.f -tcollect -g -o testf_tcollect
mpiifort test.f90 -tcollect -g -o testf90_tcollect
```

The names of the MPI executables for the above command-lines should be:

```
testc_tcollect
testcpp_tcollect
testf_tcollect
testf90_tcollect
```

To make a comparison with the Intel Trace Collector STF files:

```
testc.stf testcpp.stf testf.stf testf90.stf
```

within the directory `test_inst`, use the following `mpiexec` commands:

```
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_tcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_tcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf_tcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf90_tcollect
```

The corresponding STF data will be placed into the folder `test_inst`. To do a comparison between the STF data in `testcpp.stf` and `testcpp_tcollect.stf` the following `traceanalyzer` command can be launched from a Linux command-line panel within the folder `test_intel_mpi`:

```
traceanalyzer
```

Figure 14.1 shows the base panel for the Intel Trace Analyzer as a result of invoking the command above from a Linux panel.

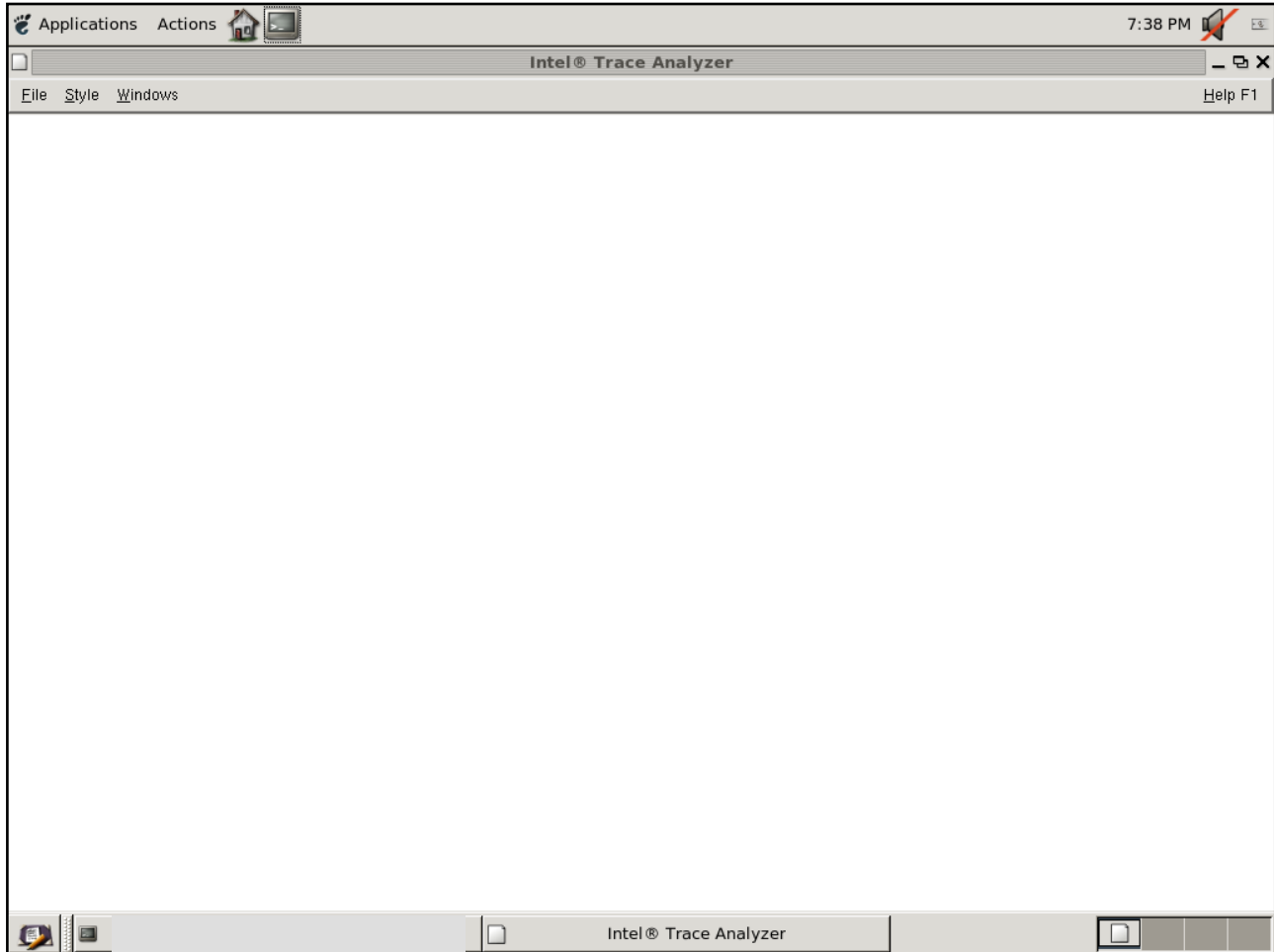


Figure 14.1 – Base panel for the Intel Trace Analyzer when invoking a Linux Shell Command: `traceanalyzer` without any arguments

If you select the menu path `File->Open` and click on the `test_inst` folder, the following panel will appear:

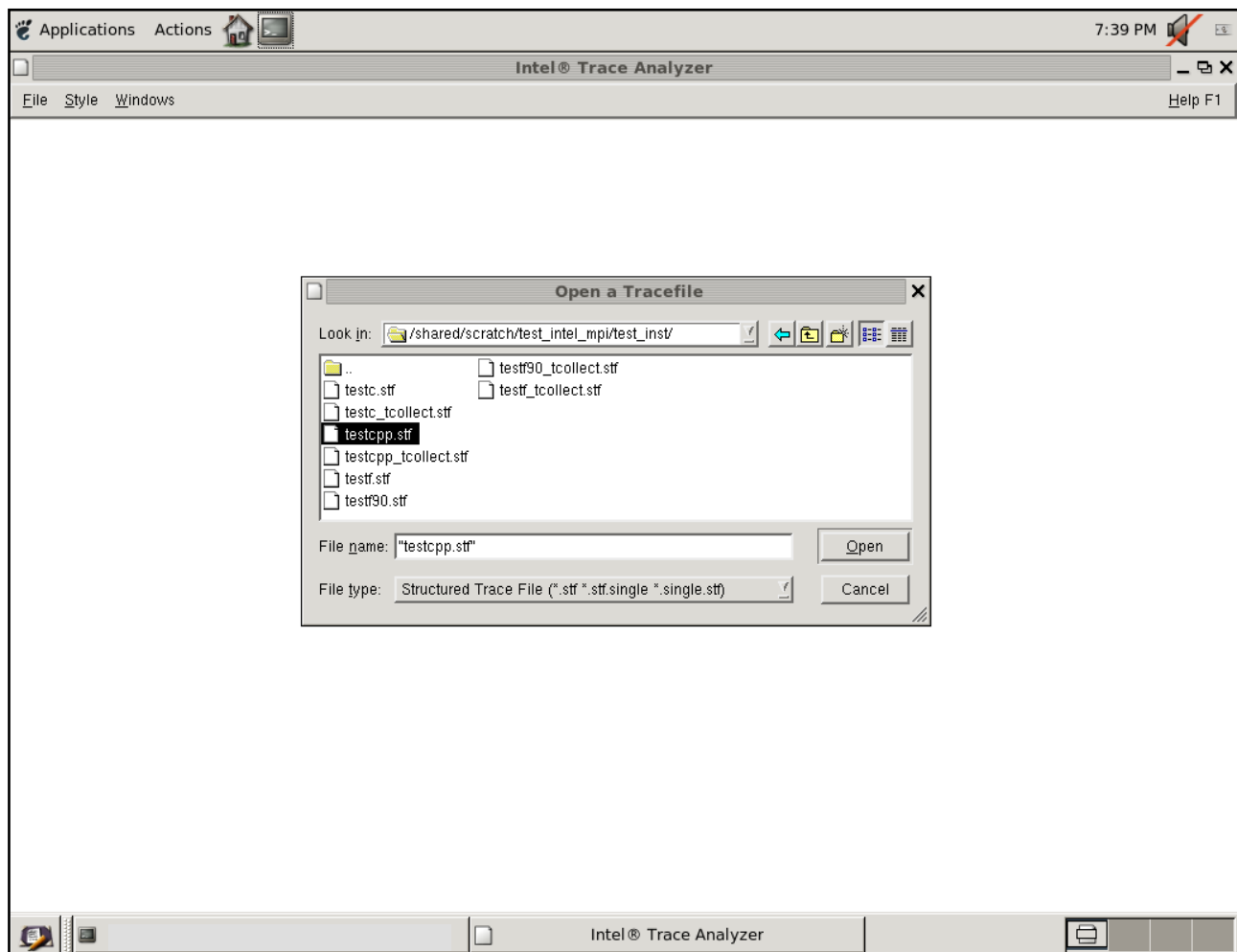


Figure 14.2 – Open a Tracefile Rendering for the test_inst Folder where testcpp.stf has been Highlighted

Selecting `testcpp.stf` will generate a Flat Profile panel within the Intel Trace Analyzer session that might look something like the following.

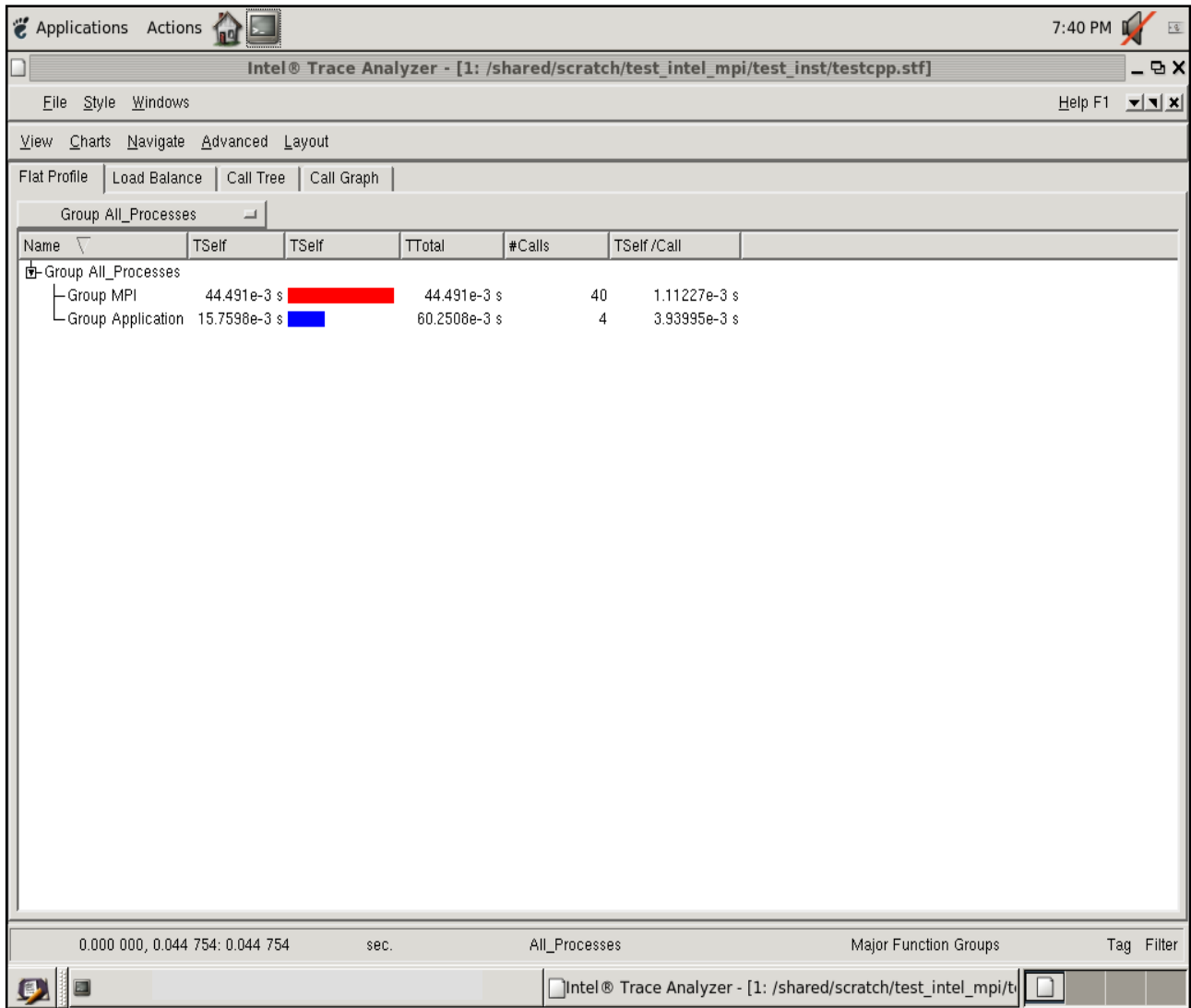


Figure 14.3 – Flat Panel Display for test_inst\testcpp.stf

For the Flat Panel Display, if you select File->Compare the following sub-panel will appear.

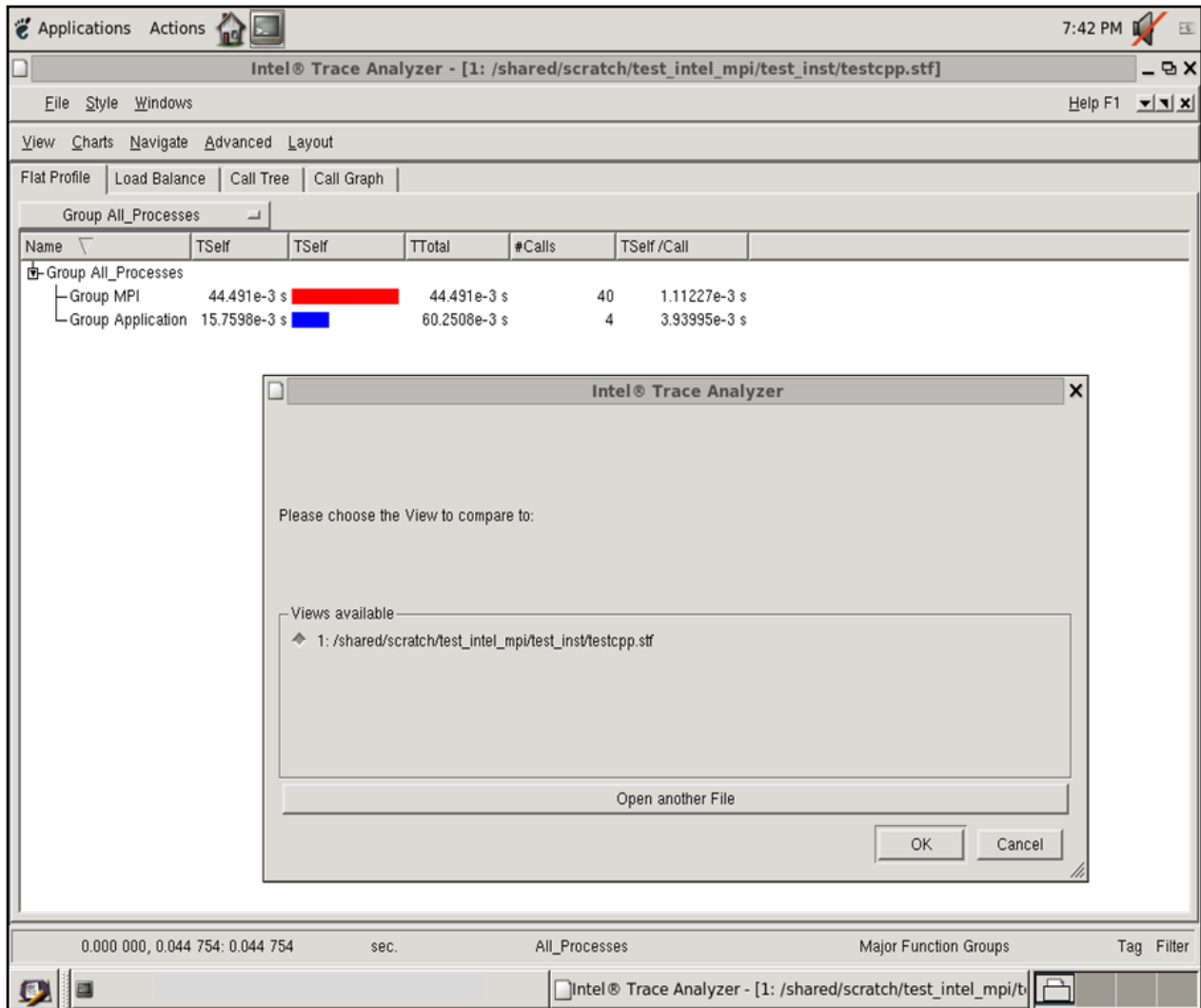


Figure 14.4 – Sub-panel Display for Adding a Comparison STF File

Click on the **Open another file** button and select `testcpp_tcollect.stf` and then proceed to push on the **Open** button with your mouse.

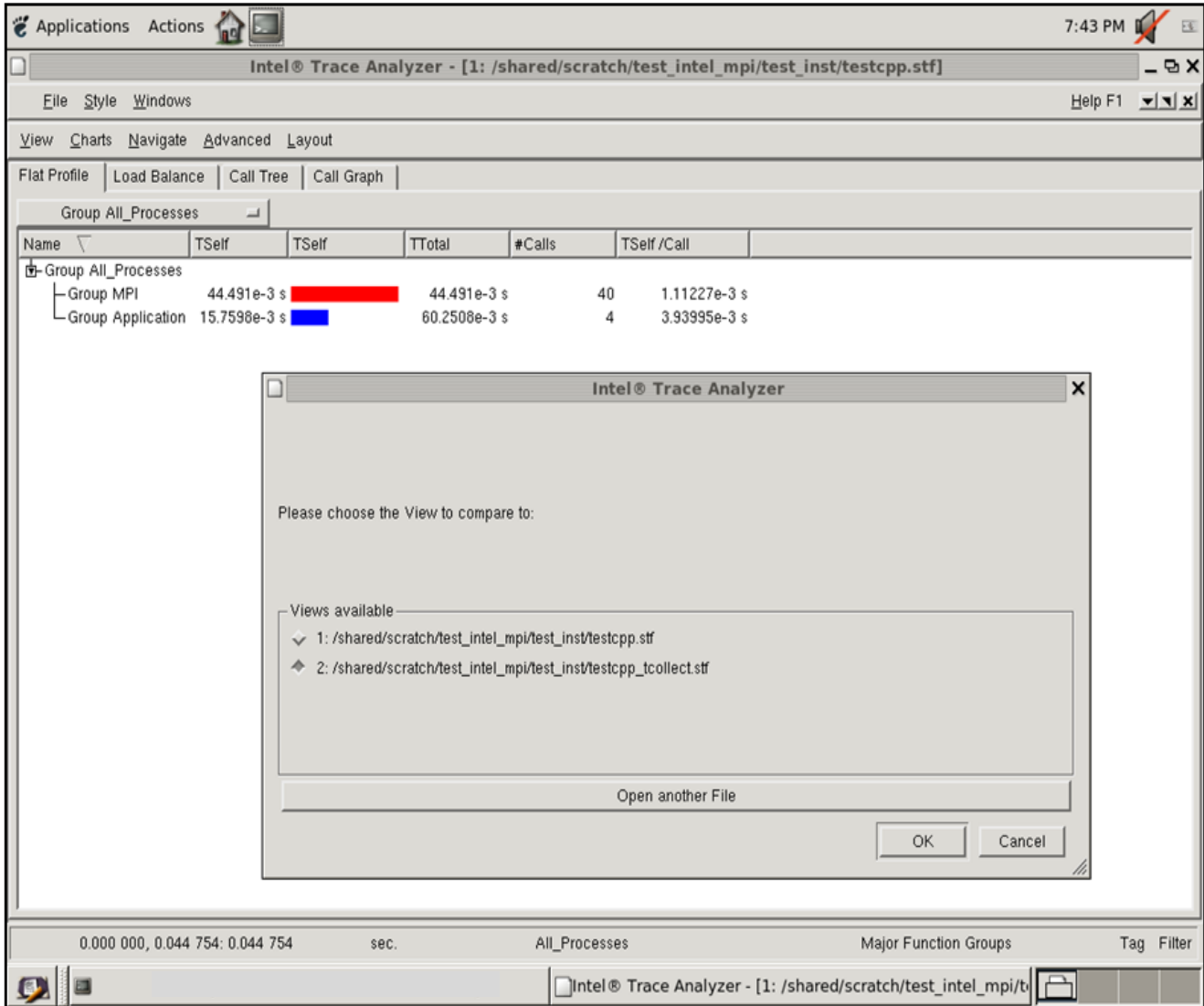


Figure 14.5 – Sub-panel Activating the Second STF File for Comparison

Click on the **OK** button in Figure 14.5 and the comparison display in Figure 14.6 will appear. In Figure 14.6, the timeline display for `testcpp_tcollect.stf` (for example, the second timeline) is longer than that of the top timeline display (`testcpp.stf`).

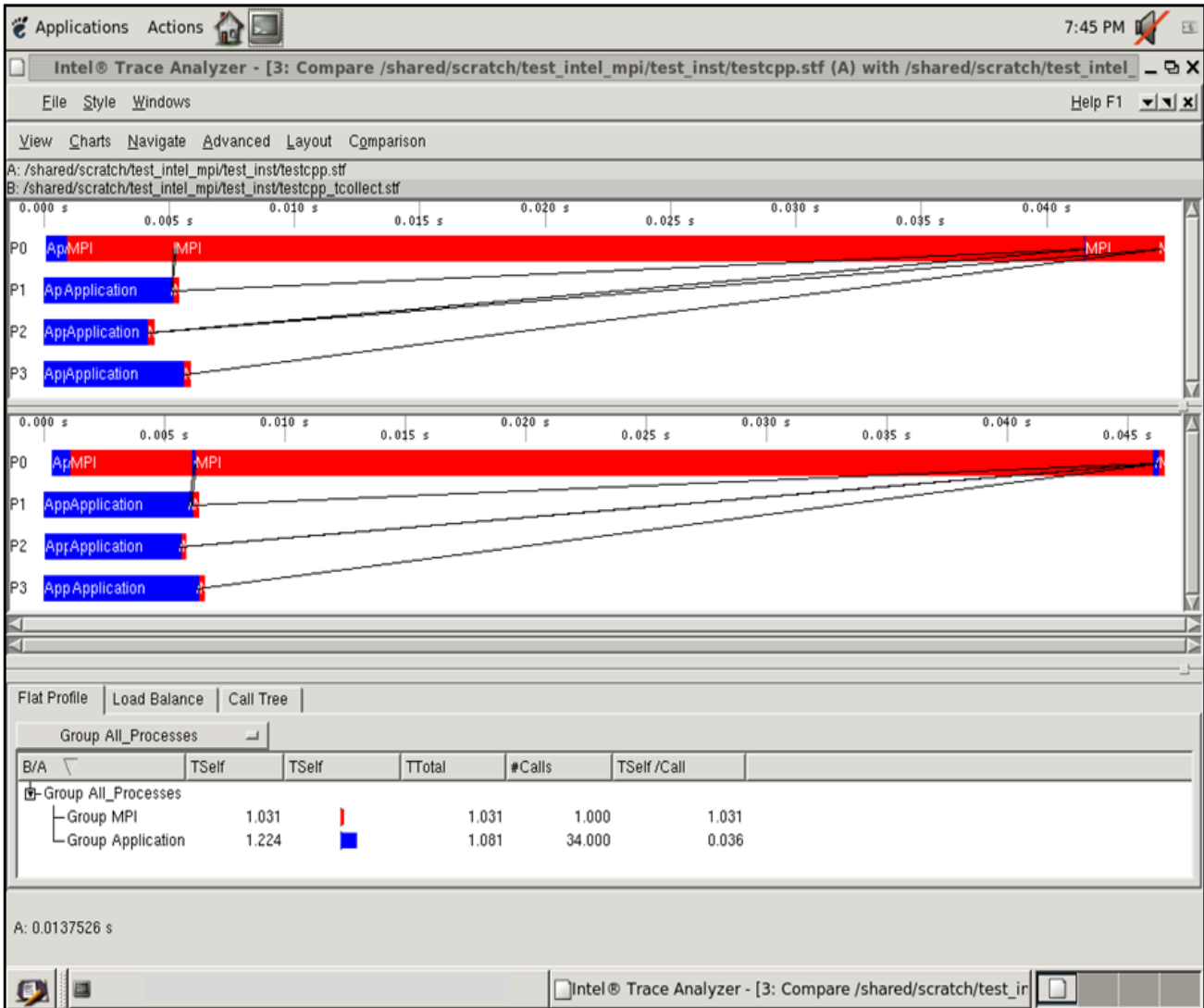


Figure 14.6 – Comparison of testcpp.stf and testcpp_tcollect.stf

At the bottom and towards the right of this panel there are two labels with the same name, namely, **Major Function Groups**. Click on the top label with this name, and a sub-panel will appear with the following information:

The screenshot shows the Intel Trace Analyzer interface. The main window displays a comparison of two .stf files: A: /shared/scratch/test_intel_mpi/test_inst/testcpp.stf and B: /shared/scratch/test_intel_mpi/test_inst/testcpp_tcollect.stf. The top panel shows a flat profile with process P0 (AppMPI) and P1 (AppApplication) highlighted in red and blue respectively. A dialog box titled "Function Group Editor for file A" is open, showing a tree view of function groups. The tree view is as follows:

Name	Depth	Children	Id
-/-			
Major Function Groups	2		22147483648
MPI	1		62147496644
Application	1		12147487039
All Functions	1		72147483649

Below the tree view, there is a section for "Apply chosen aggregation to other File (B):" with a dropdown menu set to "Never." and "OK", "Cancel", and "Apply" buttons.

Figure 14.7 – “Function Group Editor for file A” Display (i.e, for file testcpp.stf)

Highlight the **All Functions** tree entry and press the **Apply** but in the low right corner of this panel. Then click on the **OK** button. Repeat this process for the second **Major Function Groups** label at the bottom of the main **Trace Analyzer** panel. You should now see a panel rendering that looks something like:

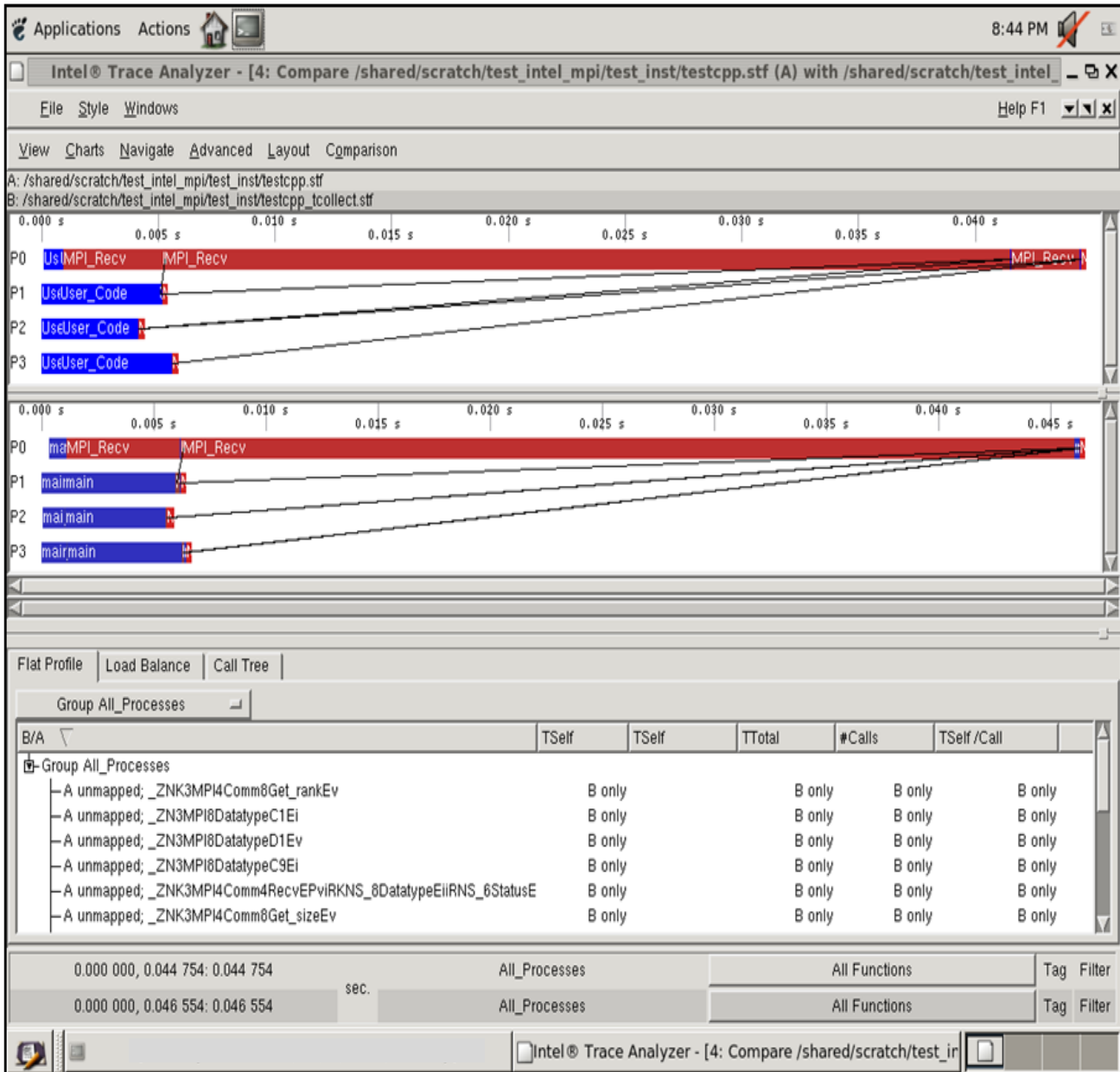


Figure 14.8 – Comparison of STF Files testcpp.stf and testcpp_tcollect.stf after making the All Functions Selection

At the top of the display panel, if you make the menu selection `Charts->Function Profile` you will be able to see a function profile comparison (lower middle and lower right) for the two executables:

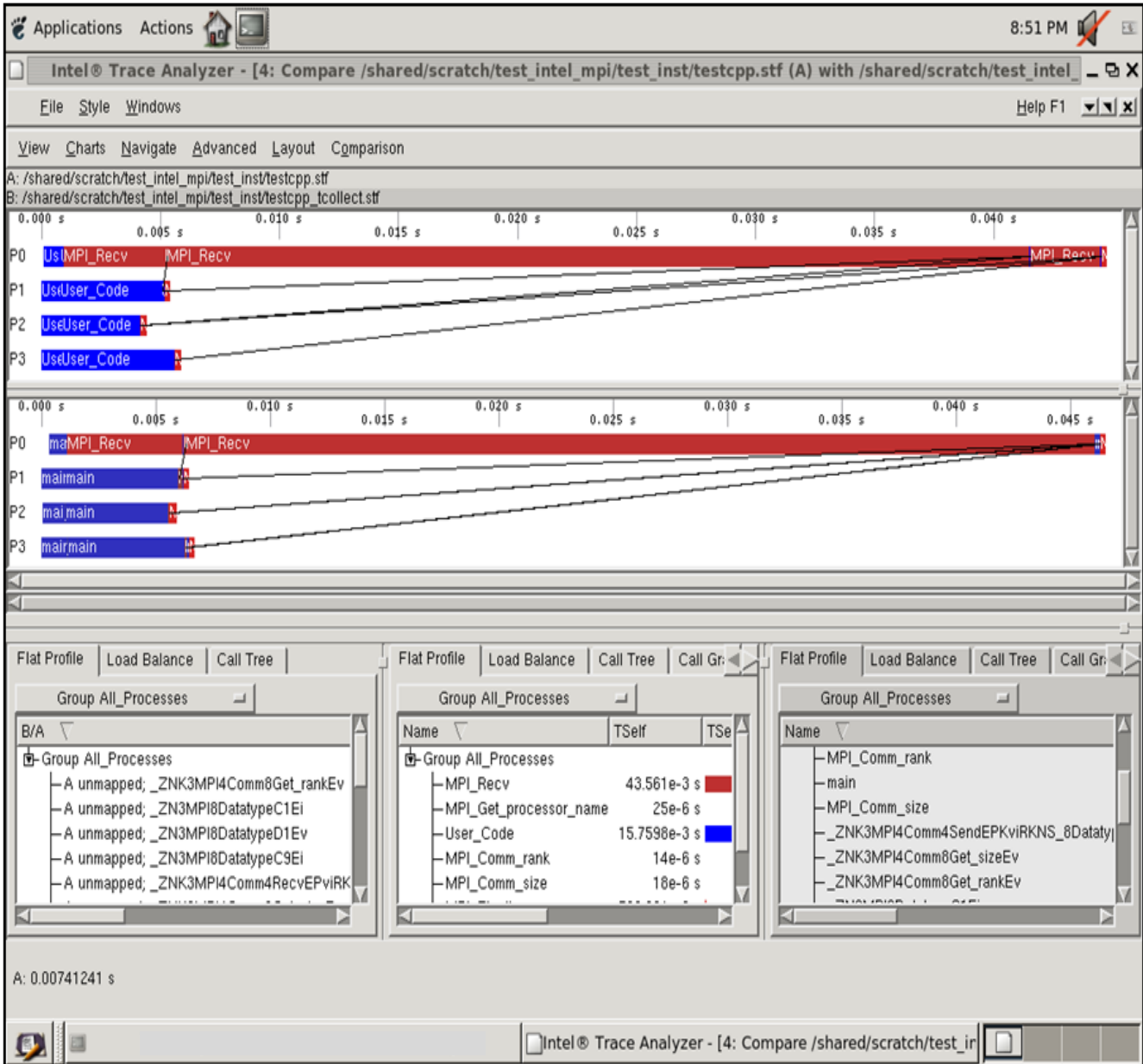


Figure 14.9 – Function Profile Sub-panels in the Lower Middle and Lower Right Sections of the Display for `testcpp.stf` and `testcpp_tcollect.stf`

NOTE: The lower right panel (`testcpp_tcollect.stf`) has much more function profiling information than the lower middle panel (`testcpp.stf`). This is the result of using the `-tcollect` switch during the compilation process. You can proceed to do similar analysis with:

- 1) `testc.stf` and `testc_tcollect.stf`
- 2) `testf.stf` and `testf_tcollect.stf`
- 3) `testf90.stf` and `testf90_tcollect.stf`

[Back to Table of Contents](#)

15. Using Co-Array Fortran

The Intel® Fortran Compiler XE, which is included as part of Intel® Cluster Studio XE, supports parallel programming using co-array semantics. These co-array semantics are defined by the Fortran 2008 Standard. You must specify the `-coarray` compiler option to enable use of co-array syntax. The possible configurations for using the `-coarray` compilation option are:

```
-coarray | -coarray=shared | -coarray=distributed
```

By default, when a co-array Fortran application is compiled with Intel® Fortran Compiler XE, the compiler creates as many images as there are processor cores on the host platform where the compilation takes place. The compilation command-line settings `-coarray` and `-coarray=shared` have the same semantic meaning.

No special procedure is necessary to run a program that uses co-arrays. You can simply run the executable file. The underlying parallelization implementation is Intel® MPI Library. Installation of the compiler automatically installs the necessary Intel® MPI run-time libraries. The use of co-array applications with any other MPI implementation, or with OpenMP*, is not supported at this time.

There are two methodologies for controlling the number of images that are created for a co-array Fortran executable.

By default, the number of images created is equal to the number of execution units on the current system. You can override that by specifying the option `-coarray-num-images=<n>` on the `ifort` command that compiles the main program. `<n>` is a positive integer. You can also specify the number of images through an environment variable called `FOR_COARRAY_NUM_IMAGES`. Setting this environment variable will control the number of images that the executable will spawn at run-time.

To access a co-array Fortran example for Linux OS click on the following Co-array Fortran path where Intel® Cluster Studio XE is installed. This path points to a tar/Zip package that has a co-array Fortran example. Copy this tar package to a scratch directory and untar it into the scratch folder. After completing the untar step, you should see a folder called "coarray_samples". Within this sub-directory, you should find a Fortran source file called `hello_image.f90` which has contents which look something like the following:

```
program hello_image

  write(*,*) "Hello from image ", this_image(), &
    "out of ", num_images()," total images"

end program hello_image
```

To build an executable for `hello_image.f90`, you can simply type something like:

```
ifort -coarray hello_image.f90 -o hello_image.exe
```

The executable that is created from the command above is called `hello_image.exe` and it can be executed by typing the command:

```
./hello_image.exe
```

The resulting output might look something like the following:

```
Hello from image          1 out of          4 total images
Hello from image          2 out of          4 total images
Hello from image          3 out of          4 total images
Hello from image          4 out of          4 total images
```

The *exact* results that you observe on your system will be a function of your processor architecture, OS configuration, etc.

[Back to Table of Contents](#)

15.1 Running a Co-array Fortran Example on a Distributed System

Suppose that you have a four node cluster that you wish to run the `hello_image.f90` application on. So as to *verify* that the corresponding executable is running on each of the four nodes, you can augment the original source so that it looks something like the following:

```
program hello_image

  use IFPORT

  character(MAX_HOSTNAM_LENGTH + 1) hostname
  integer istat

  istat = hostnam(hostname)
  write(*,*) "Hello from image ", this_image(), &
    "on host: ", hostname, "out of ", &
    num_images()," total images"

end program hello_image
```

For the above source, the Fortran module `IFPORT` has been added so that the function called `hostnam` can be used to extract the hostname from each of the nodes for which the co-array Fortran application is running on. The `hostname` variable is included in the write statement contents above.

Here is a quick recipe for compiling and executing the modified co-array Fortran example on Linux*-based distributed system.

- 1) Issue the command `mpdtrace` to see if there are any multipurpose daemons running on your cluster. Let us assume that this is a four node cluster where the nodes are respectively called `clusternode1`, `clusternode2`, `clusternode3`, and `clusternode4`. The command for verifying the presence of the multipurpose daemons should look something like the following:

```
mpdtrace
```

If these daemons are in existence, you should see a list of compute nodes. In our case, the list would be:

```
clusternode1
clusternode2
clusternode3
clusternode4
```

If instead, you see a message that looks something like:

```
mpdtrace: cannot connect to local mpd (/tmp/mpd2.console_user01);
possible causes:
1. no mpd is running on this host
2. an mpd is running but was started without a "console" (-n
option)
```

Then proceed to section 15.2 before proceeding to step 2.

- 2) Create a configuration file that might look something like the following for a four node cluster:

```
-n 1 -host clusternode1 ./hello_image.exe : \
-n 1 -host clusternode2 ./hello_image.exe : \
-n 1 -host clusternode3 ./hello_image.exe : \
-n 1 -host clusternode4 ./hello_image.exe
```

Save the contents of the above into a file called `configuration`. The file called `configuration` will be used as part of the `ifort` compilation line.

- 3) Compile the modified co-array Fortran application above using the `configuration` file as follows:

```
ifort hello_image.f90 -coarray=distributed -coarray-config-
file=configuration -o hello_image.exe
```

- 4) Run the executable by typing the command:

```
./hello_image.exe
```

The results might look *something* like the following:

```
Hello from image 1 on host: clusternode1 out of 4 total images
Hello from image 2 on host: clusternode2 out of 4 total images
Hello from image 3 on host: clusternode3 out of 4 total images
Hello from image 4 on host: clusternode4 out of 4 total images
```

Notice in the resulting output that each of the nodes in our cluster example (that is, `clusternode1`, `clusternode2`, `clusternode3`, and `clusternode4`) is referenced.

[Back to Table of Contents](#)

15.2 Trouble Shooting for the Absence of Multipurpose Daemons

This subsection is designed to help you if you issued an `mpdtrace` command on cluster system and you received a message that looks something like the following:

```
mpdtrace: cannot connect to local mpd (/tmp/mpd2.console_user01);
possible causes:
1. no mpd is running on this host
2. an mpd is running but was started without a "console" (-n
option)
```

- 1) Either locate or create a text file with the list of nodes (one per line) that make up the cluster. Suppose that you have a four node cluster where the nodes of the cluster are respectively:

```
clusternode1
clusternode2
clusternode3
clusternode4
```

Place these node names into a file where this file that contains the cluster names might be called `machines.LINUX`. Next, issue the command:

```
mpdboot -n 4 -r ssh -f machines.LINUX
```

- 2) After issuing the `mpdboot` command, verify that nodes in say the `machines.LINUX` have been registered properly by issuing the command `mpdtrace`. The results of the `mpdtrace` command should look something like the following:

```
clusternode1
clusternode2
clusternode3
clusternode4
```

you can proceed to step 2 in Section 15.1.

[Back to Table of Contents](#)

16. Using the CEAN Language Extension and Programming Model

For Intel® Cluster Studio XE, CEAN is an array language extension to C/C++, providing array section notations for SIMD vector parallelism and parallel function maps for multi-threading. CEAN is an acronym for C/C++ Extensions for Array Notations. This is an Intel-specific programming language extension supported by the Intel compiler. For the complete language extension specification, see the C/C++ Extension for Array Notation (CEAN) Specification Version 1.0.

The example below combines the use of C/C++ Extensions for Array Notations along with using the MPI_Gather communication collective.

```
#include <malloc.h>
#include "mpi.h"
#include <stdio.h>
#include <string.h>
const int MAX_ARRAY_SIZE = 100;

int main (int argc, char *argv[])
{

int i, namelen, rank, root_process = 0, size;
char name[MPI_MAX_PROCESSOR_NAME];
int a[MAX_ARRAY_SIZE], b[MAX_ARRAY_SIZE], c[MAX_ARRAY_SIZE];
int *d;
MPI_Status stat;

MPI_Init (&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(name, &namelen);

// The root process will allocated array storage for gathering results
// from each of
// the processes

if (rank == root_process) {
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    d = (int *) malloc(size * 100 * sizeof(int));
}

// Use C/C++ array notation to do partial array computation within each
MPI process
a[0:MAX_ARRAY_SIZE] = 1 + rank;
b[0:MAX_ARRAY_SIZE] = 2 + rank;
c[0:MAX_ARRAY_SIZE] = a[0:MAX_ARRAY_SIZE] + b[0:MAX_ARRAY_SIZE];
```

```

fprintf(stdout,"Process rank %d of %d running on %s ready to call
MPI_Gather\n",
        rank,size,name);

// Use the MPI Gather communication collective to gather the partial
results
MPI_Gather(c, 100, MPI_INT, d, 100, MPI_INT, root_process,
MPI_COMM_WORLD);

MPI_Finalize();

// Print out the first and last result elements that were computed by
each MPI process
if (rank == root_process) {
    for (i = 0; i < size; i++)
        fprintf(stdout,"Strided array elements d[%d] = %d; d[%d] =
        %d\n",i*MAX_ARRAY_SIZE,
        d[i*MAX_ARRAY_SIZE],i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1,
        d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
    free(d);
}

return (0);
}

```

The MPI_Gather communication collective gathers partial results from adding vectors "a", and "b" together and storing the computations into array "c". Each MPI process transfers its "c" data into an array "d" which has the capacity to store all the values for each array "c" instance as defined by each MPI process.

You can cut and paste the code fragment above into a C file such as `cean.c`, and create an executable by issuing the following command:

```
mpiicc cean.c -o cean.exe
```

This may be following by issuing an `mpiexec` command such as:

```
mpiexec -n 4 ./cean.exe
```

where 4 MPI processes are used. The output results might look something like:

```

Process rank 0 of 4 running on clusternode1 ready to call MPI_Gather
Process rank 2 of 4 running on clusternode3 ready to call MPI_Gather
Process rank 1 of 4 running on clusternode2 ready to call MPI_Gather
Process rank 3 of 4 running on clusternode3 ready to call MPI_Gather
Strided array elements d[0] = 3; d[99] = 3
Strided array elements d[100] = 5; d[199] = 5
Strided array elements d[200] = 7; d[299] = 7
Strided array elements d[300] = 9; d[399] = 9

```

The type of results that you obtain will be dependent on your cluster configuration and the number of MPI processes that you use.

[Back to Table of Contents](#)

17. Using Intel® VTune™ Amplifier XE

To analyze the performance of an MPI program at the threading level, the Intel® VTune™ Amplifier XE performance analyzer should be used. It is installed at `/opt/intel/vtune_amplifier_xe_2011`.

To use Intel® VTune™ Amplifier XE, follow three basic steps:

1. Use the `amplxe-cl` command line tool to analyze the program. By default all processes are analyzed, but it is possible to filter the data collection using the `amplxe-cl` tool to limit the number of processes analyzed to that of a subset. An individual result directory will be created for each spawned MPI program process that is to be analyzed.
2. The finalization is done automatically for each result directory once the performance data collection has finished.
3. Each result directory from step 1 can be opened in an Intel® VTune™ Amplifier XE GUI standalone viewer to analyze the data for the specific process.

For Intel® Cluster Studio XE 2012 on Linux* OS, the behavior of `ictvars.sh` and `ictvars.csh` is different. On Linux* OS, `ictvars.csh` is unable to initialize environment variables for Intel® VTune™ Amplifier XE. This defect will be resolved in a future release of Intel® Cluster Studio XE. **In the meantime, if you wish to use Intel® VTune™ Amplifier XE, source `ictvars.sh`.** For Bourne* Shell on Linux* OS, once the Intel® Cluster Studio XE environment variables referenced within the "`ictvars.sh`" file have been sourced via a `.bashrc` file, users for a given Bourne* Shell login session can simply type:

```
. ictvars.sh ia32
```

for creating IA-32 executables. Alternatively, to restore the default Intel® Cluster Studio XE environment variable settings so as to build executables with Intel® 64 address extensions, type:

```
. ictvars.sh
```

within the Bourne* Shell login session.

NOTE: The full path to `ictvars.sh` can be omitted once it has been sourced in the `.bashrc` file.

Chapter 13, which is titled [User Checklist for Linux* OS](#), provides details on setting up `ictvars.sh` within Bourne* Shell.

[Back to Table of Contents](#)

17.1 How do I get a List of Command-line Options for the Intel® VTune™ Amplifier XE Tool?

Within a Bourne* Shell login session, type the command:

```
amplxe-cl -help
```

[Back to Table of Contents](#)

17.2 What does a Programming Example Look Like that I might run with Intel® VTune™ Amplifier XE?

This programming example uses the C/C++ array notation that was discussed earlier.

```
#include <malloc.h>
#include "mpi.h"
#include <stdio.h>
#include <string.h>
const int MAX_ARRAY_SIZE = 100;

int main (int argc, char *argv[])
{

int i, namelen, rank, root_process = 0, size;
char name[MPI_MAX_PROCESSOR_NAME];
int a[MAX_ARRAY_SIZE], b[MAX_ARRAY_SIZE], c[MAX_ARRAY_SIZE];
int *d;
MPI_Status stat;

MPI_Init (&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(name, &namelen);

// The root process will allocated array storage for gathering results
// from each of
// the processes

if (rank == root_process) {
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    d = (int *) malloc(size * 100 * sizeof(int));
}
}
```

```

// Use C/C++ array notation to do partial array computation within each
MPI process
a[0:MAX_ARRAY_SIZE] = 1 + rank;
b[0:MAX_ARRAY_SIZE] = 2 + rank;
c[0:MAX_ARRAY_SIZE] = a[0:MAX_ARRAY_SIZE] + b[0:MAX_ARRAY_SIZE];

fprintf(stdout, "Process rank %d of %d running on %s ready to call
MPI_Gather\n",
rank, size, name);

// Use the MPI Gather communication collective to gather the partial
results
MPI_Gather(c, 100, MPI_INT, d, 100, MPI_INT, root_process,
MPI_COMM_WORLD);

MPI_Finalize();

// Print out the first and last result elements that were computed by each
MPI process
if (rank == root_process) {
for (i = 0; i < size; i++)
fprintf(stdout, "Strided array elements d[%d] = %d; d[%d] =
%d\n", i*MAX_ARRAY_SIZE,
d[i*MAX_ARRAY_SIZE], i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1,
d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
free(d);
}
return (0);
}

```

[Back to Table of Contents](#)

17.3 How do I Run and Collect Intel® VTune™ Amplifier XE Performance Information within an Intel® MPI Library Application?

A command-line that uses Intel® Amplifier XE might look something like:

```

mpiexec -n 4 amplxe-cl -r amplifierxe_results -collect hotspots --
./cean.exe

```

[Back to Table of Contents](#)

17.4 What does the Intel® VTune™ Amplifier XE Graphical User Interface Look Like?

One method of launching the graphical user interface for Intel® Amplifier XE is through the command-line:

```
amplxe-gui amplifierxe_results.0
```

where `amplifierxe_results.0` is a results folder for an MPI process.

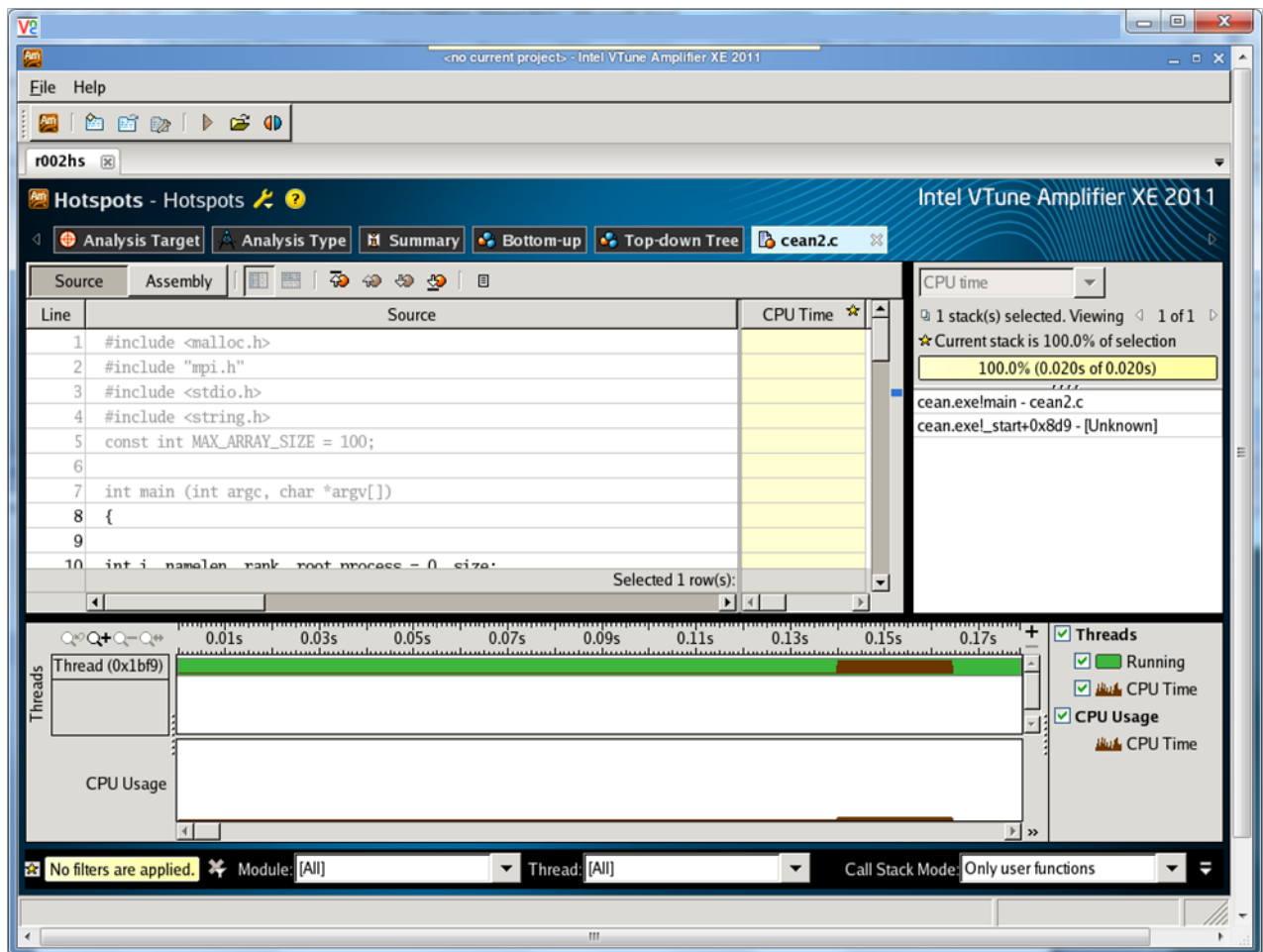


Figure 17.1 – Launching the Intel® VTune™ Amplifier XE GUI

[Back to Table of Contents](#)

18. Using Intel® Inspector XE

To analyze the correctness of an MPI program at the threading level, the Intel® Inspector XE checker should be used. It is installed in the folder path `/opt/intel/inspector_xe_2011`.

To use Intel® Inspector XE, there are three basic steps:

1. Use the `inspxe-cl` command line tool to analyze the program. By default all processes are analyzed, but it is possible to filter the data collection using the `inspxe-cl` tool to limit the number of processes checked to that of a subset. An individual result directory will be created for each spawned MPI program process that is to be checked.
2. The finalization is done automatically for each result directory once the checking analysis has finished.
3. Each result directory from step 1 can be opened in an Intel® Inspector XE GUI standalone viewer to analyze the data for the specific process.

For Intel® Cluster Studio XE 2012 on Linux* OS, the behavior of `ictvars.sh` and `ictvars.csh` is different. On Linux* OS, `ictvars.csh` is unable to initialize environment variables for Intel® Inspector XE. This defect will be resolved in a future release of Intel® Cluster Studio XE. **In the meantime, if you wish to use Intel® Inspector XE, source `ictvars.sh`.** For Bourne* Shell on Linux* OS, once the Intel® Cluster Studio XE environment variables referenced within the "`ictvars.sh`" file have been sourced via a `.bashrc` file, users for a given Bourne* Shell login session can simply type:

```
. ictvars.sh ia32
```

for creating IA-32 executables. Alternatively, to restore the default Intel® Cluster Studio XE environment variable settings so as to build executables with Intel® 64 address extensions, type:

```
. ictvars.sh
```

within the Bourne* Shell login session.

NOTE: The full path to `ictvars.sh` can be omitted once it has been sourced in the `.bashrc` file.

Chapter 13, which is titled [User Checklist for Linux* OS](#), provides details on setting up `ictvars.sh` within Bourne* Shell.

[Back to Table of Contents](#)

18.1 How do I get a List of Command-line Options for the Intel® Inspector XE Tool?

Within a Bourne* Shell login session, type the command:

```
inspxe-cl -help
```

[Back to Table of Contents](#)

18.2 What does a Programming Example Look Like that has a Memory Leak?

This programming example has a call to `malloc` without a call to `free`.

```
#include <malloc.h>
#include "mpi.h"
#include <stdio.h>
#include <string.h>
const int MAX_ARRAY_SIZE = 100;

int main (int argc, char *argv[])
{

int i, namelen, rank, root_process = 0, size;
char name[MPI_MAX_PROCESSOR_NAME];
int a[MAX_ARRAY_SIZE], b[MAX_ARRAY_SIZE], c[MAX_ARRAY_SIZE];
int *d;
MPI_Status stat;

MPI_Init (&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(name, &namelen);

// The root process will allocated array storage for gathering results
// from each of
// the processes

if (rank == root_process) {
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    d = (int *) malloc(size * 100 * sizeof(int));
}

// Use C/C++ array notation to do partial array computation within each
// MPI process
a[0:MAX_ARRAY_SIZE] = 1 + rank;
b[0:MAX_ARRAY_SIZE] = 2 + rank;
c[0:MAX_ARRAY_SIZE] = a[0:MAX_ARRAY_SIZE] + b[0:MAX_ARRAY_SIZE];
```

```

fprintf(stdout,"Process rank %d of %d running on %s ready to call
    MPI_Gather\n",
        rank,size,name);

// Use the MPI Gather communication collective to gather the partial
    results
MPI_Gather(c, 100, MPI_INT, d, 100, MPI_INT, root_process,
    MPI_COMM_WORLD);

MPI_Finalize();

// Print out the first and last result elements that were computed by
    each MPI process
if (rank == root_process) {
    for (i = 0; i < size; i++)
        fprintf(stdout,"Strided array elements d[%d] = %d; d[%d] =
            %d\n",i*MAX_ARRAY_SIZE,
                d[i*MAX_ARRAY_SIZE],i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1,
                d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
}

return (0);
}

```

[Back to Table of Contents](#)

18.3 How do I Run and Collect Memory Leak Information within an Intel® MPI Library Application?

A command-line that uses Intel® Inspector XE might look something like:

```

mpiexec -n 4 inspxe-cl -r inspectorxe_results -collect mil --
    ./cean.exe

```

[Back to Table of Contents](#)

18.4 What does the Intel® Inspector XE Graphical User Interface Look Like?

One method of launching the graphical user interface for Intel® Inspector XE is through the command-line:

inspxe-gui inspectorxe_results.0

where inspectorxe_results.0 is a results folder for an MPI process.

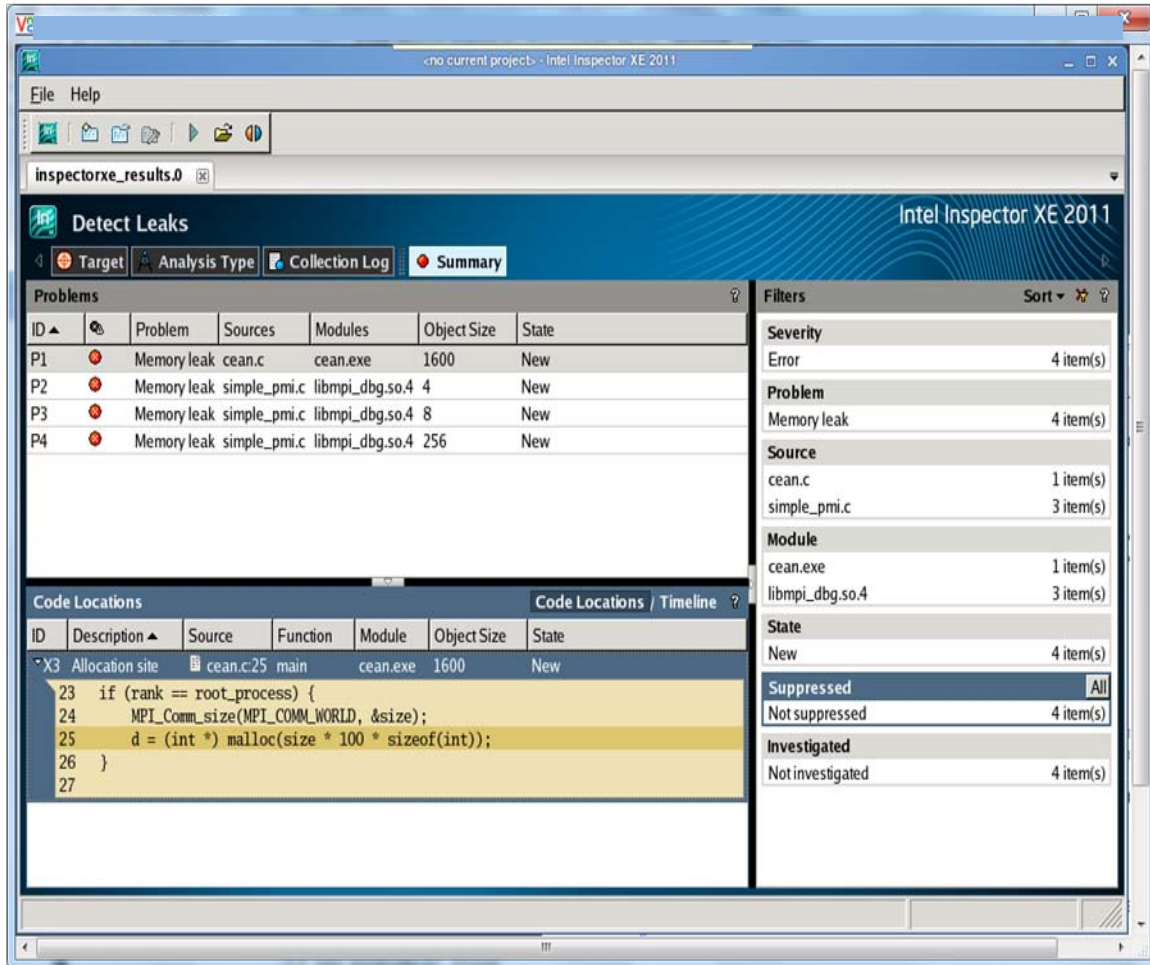


Figure 18.1 – Launching the Intel® Inspector XE GUI

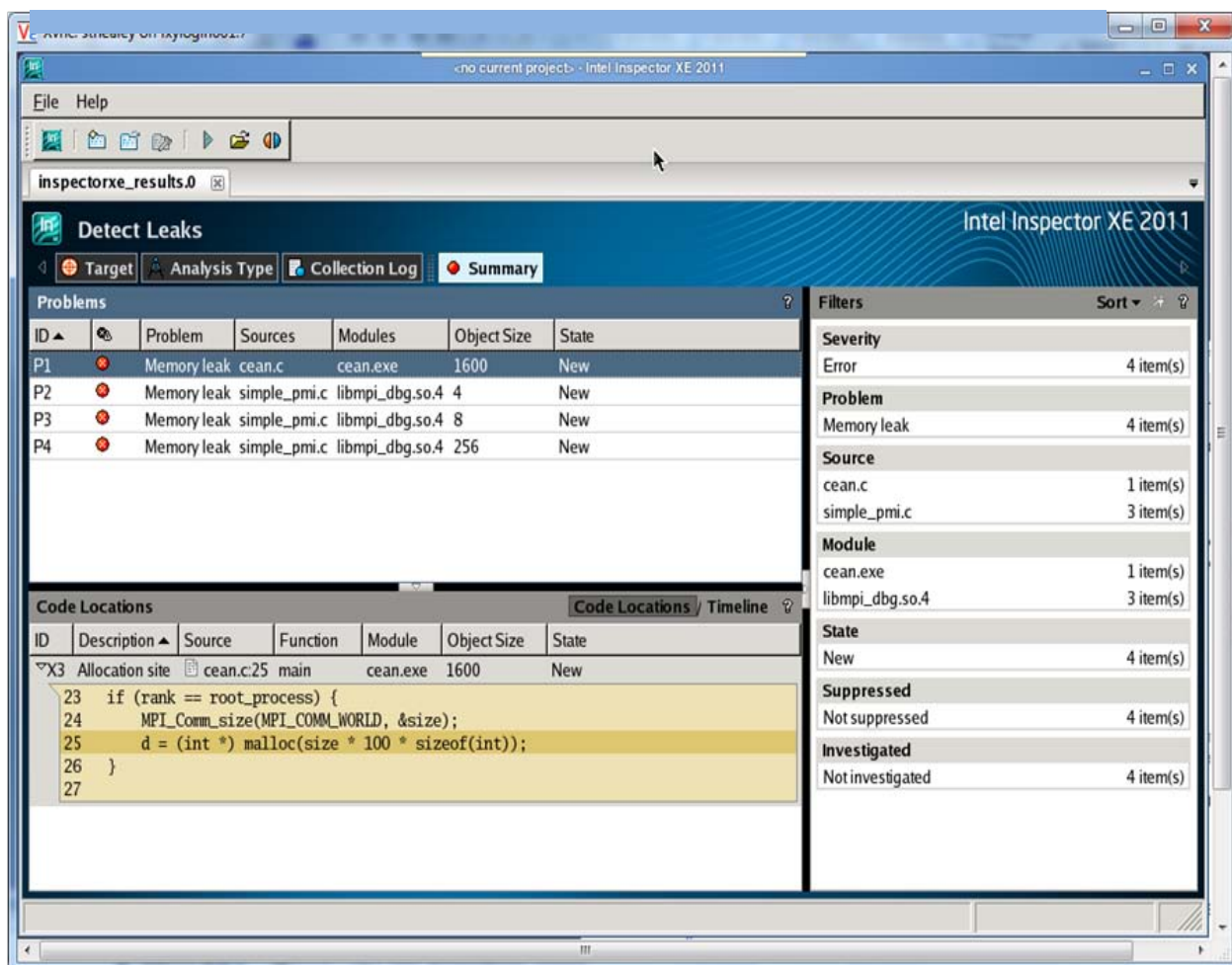


Figure 18.2 – Isolating a Memory Leak in the User’s Applications by Pressing on ID Row P1

The way to resolve this memory leak in Figure 18.2 is to add a call to the free function for the pointer object “d”. The C/C++ code fragment:

```

...

// Print out the first and last result elements that were computed by each
MPI process
if (rank == root_process) {
    for (i = 0; i < size; i++)
        fprintf(stdout, "Strided array elements d[%d] = %d; d[%d] =
%d\n", i*MAX_ARRAY_SIZE,
                d[i*MAX_ARRAY_SIZE], i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1,
                d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
}

```

...

will be modified to:

...

```
// Print out the first and last result elements that were computed by each
MPI process
if (rank == root_process) {
    for (i = 0; i < size; i++)
        fprintf(stdout, "Strided array elements d[%d] = %d; d[%d] =
%d\n", i*MAX_ARRAY_SIZE,
                d[i*MAX_ARRAY_SIZE], i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1,
                d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
    free(d);
}
```

...

where a `free` statement for object “d” has been added. The `mpiexec` command for rerunning the Intel® Inspector XE application might look something like:

```
mpiexec -n 4 inspxe-cl -r inspectorxe_results2 -collect mil --
./cean2.exe
```

Rerunning the GUI analysis tool:

```
inspxe-gui inspectorxe_results2.0
```

demonstrates that the memory leak for pointer object “d” has been removed (Figure 18.3).

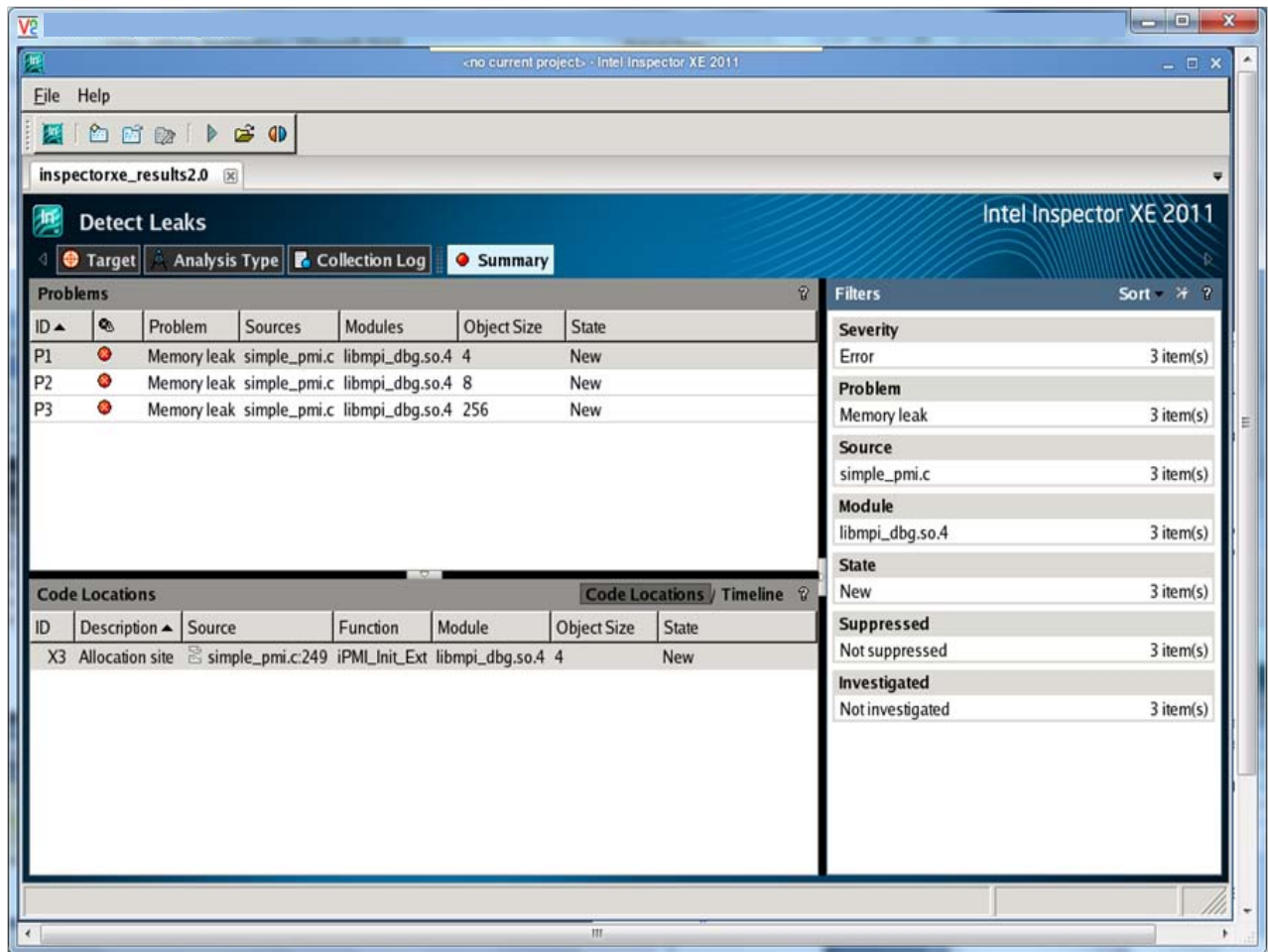


Figure 18.3 – The memory leak for pointer object “d” has been removed

[Back to Table of Contents](#)

19. Using Intel® Parallel Advisor for non-MPI C/C++ Software Applications

Intel® Parallel Advisor is only available on Microsoft* Windows* OS.

[Back to Table of Contents](#)