Contents

Disclaimer and Legal Information ........................................................................................................5

2. Introduction .......................................................................................................................................7

3. Intel Software Downloads and Installation on Microsoft Windows Compute Cluster Server (Microsoft Windows CCS*) OS ................................................................. 11
   3.1 Microsoft Windows CCS* OS Installation ........................................................................ 12
   3.2 Silent Install Mode ........................................................................................................ 22
   3.3 Navigating through the Intel® Cluster Tools after the Install has been Completed .......... 23

4. Integrated Development Environments for Intel® Cluster Studio XE ........................................... 29

5. Getting Started with Intel® MPI Library .......................................................................................... 31
   5.1 Launching MPD Daemons .................................................................................................. 32
   5.2 How to Set Up MPD Daemons on Microsoft Windows CCS* OS .................................. 33
   5.3 Compiling and Linking with Intel® MPI Library on Microsoft Windows CCS* OS .... 34
   5.4 Selecting a Network Fabric ............................................................................................... 37
   5.5 Running an MPI Program Using Intel® MPI Library on Microsoft Windows CCS* OS .. 38
   5.6 Experimenting with Intel® MPI Library on Microsoft Windows CCS* OS ................... 40
   5.7 Controlling MPI Process Placement on Microsoft Windows CCS* OS ....................... 43
   5.8 Using the Automatic Tuning Utility Called mpitune .......................................................... 44
      5.8.1 Cluster Specific Tuning ............................................................................................. 46
      5.8.2 MPI Application-Specific Tuning ............................................................................ 47

6. Interoperability of Intel® MPI Library with the I_MPI_DEBUG Environment Variable .......... 48

7. Instrumenting MPI Applications with Intel® Trace Analyzer and Collector ............................... 49
   7.1 Instrumenting the Intel® MPI Library Test Examples ...................................................... 49
   7.2 Instrumenting the Intel® MPI Library Test Examples in a Fail-Safe Mode ....................... 54
   7.3 Using itcpin to Instrument an Application ........................................................................ 56
   7.4 Working with the Intel® Trace Analyzer and Collector Examples .................................... 59
   7.5 Experimenting with the Message Checking Component of Intel® Trace Collector ........ 62
   7.6 Saving a Working Environment through a Project File .................................................... 72
   7.7 Analysis of Application Imbalance .................................................................................... 75
   7.8 Analysis with the Ideal Interconnect Simulator ................................................................. 78
   7.9 Building a Simulator with the Custom Plug-in Framework ............................................. 81

8. Getting Started in Using the Intel® Math Kernel Library (Intel® MKL) ........................................ 82
   8.1 Experimenting with ScaLAPACK* ...................................................................................... 82
   8.2 Experimenting with the Cluster DFT Software ................................................................. 86
   8.3 Experimenting with the High Performance Linpack Benchmark* ..................................... 95

9. Using the Intel® MPI Benchmarks .................................................................................................. 99
   9.1 Building Microsoft Visual Studio* x64 Executables for the Intel® MPI Benchmarks .......... 100
   9.2 Building Microsoft Visual Studio* IA-32 Executables for the Intel® MPI Benchmarks ...... 104

10. Uninstalling the Intel® Cluster Studio XE on Microsoft* Windows* HPC Server 2008 OS .......................................................... 112

11. Hardware Recommendations for Installation on Microsoft* Windows* CCS OS ........ 118

12. System Administrator Checklist for Microsoft Windows CCS* OS ........................................ 119
13. User Checklist for Microsoft Windows CCS* OS ....................................................... 120
14. Using the Compiler Switch /Qtcollect ..................................................................... 121
15. Using Co-Array Fortran .......................................................................................... 131
16. Using the CEAN Language Extension and Programming Model ....................... 133
17. Using Intel® VTune™ Amplifier XE ................................................................. 135
   17.1 How do I get a List of Command-line Options for the Intel® VTune™ Amplifier XE Tool? ......................................................................................................... 135
   17.2 What does a Programming Example Look Like that I might run with Intel® VTune™ Amplifier XE? .......................................................... 135
   17.3 How do I Run and Collect Intel® VTune™ Amplifier XE Performance Information within an Intel® MPI Library Application? ..................................................... 137
   17.4 What does the Intel® VTune™ Amplifier XE Graphical User Interface Look Like? ........................................................................................................ 137
18. Using Intel® Inspector XE ..................................................................................... 139
   18.1 How do I get a List of Command-line Options for the Intel® Inspector XE Tool? 139
   18.2 What does a Programming Example Look Like that has a Memory Leak? .... 139
   18.3 How do I Run and Collect Memory Leak Information within an Intel® MPI Library Application? .......................................................... 141
   18.4 What does the Intel® Inspector XE Graphical User Interface Look Like? ..... 141
19. Using Intel® Parallel Advisor for non-MPI C/C++ Software Applications ............. 146
   19.1 How do I Access Intel® Parallel Advisor? .......................................................... 146
   19.2 How do I bring up the Intel® Parallel Advisor Workflow within Microsoft* Visual Studio*? ........................................................................... 146
## Revision History

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Revision Number</th>
<th>Description</th>
<th>Revision Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>325977-001EN</td>
<td>20111108</td>
<td>Updated Intel® Cluster Studio XE 2012 for Microsoft* Windows* Compute Cluster Server OS Tutorial to reflect changes and improvements to the software components.</td>
<td>11/08/2011</td>
</tr>
</tbody>
</table>
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/


* 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.
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
2. Introduction

In terms of the Intel® Cluster Studio XE software for Windows* OS, consider references within this document to Microsoft Windows Compute Cluster Server* (CCS*) OS, Microsoft* Windows* HPC Server 2008 OS, and Microsoft* Windows* HPC Server 2008 R2 OS as interchangeable. Intel® Cluster Studio XE 2012 on Microsoft* Windows* consists of:

1. Intel® C++ Compiler XE 12.1
2. Intel® Fortran Compiler XE 12.1
3. Intel® Inspector XE 2011 Update 6
4. Intel® Integrated Performance Primitives 7.0 Update 5
5. Intel® Math Kernel Library 10.3 Update 6
6. Intel® MPI Benchmarks 3.2.3
7. Intel® MPI Library 4.0 Update 3
8. Intel® Parallel Advisor 2011 Update 3 (For Windows* OS and installed only on the master node. The Intel® Parallel Advisor tool is designed strictly for a C/C++ threaded-only programming application, and is not intended for an MPI or hybrid parallel programming application. Intel Parallel Advisor needs a pre-installation of Microsoft Visual Studio 2005, 2008, or 2010 Standard Edition (or a higher edition) with the C++ compiler component installed)
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
12. Integration of Intel® Visual Fortran Compiler within Microsoft* Visual Studio* (For Windows* OS and installed only on the master node)

The software architecture of the Intel Cluster Studio XE for Microsoft Windows is illustrated in Figure 2.1:
Figure 2.1 – The Software Architecture of Intel® Cluster Studio XE on Microsoft® Windows® CCS OS

Intel Application Libraries
- Intel® Integrated Performance Primitives
- Intel® Math Kernel Library (includes ScaLAPACK and Cluster DFT)
- Intel® MPI Library
- Intel® Threading Building Blocks

Intel Analysis/Development Tools
- Intel® Inspector XE
- Intel® Parallel Advisor (The Intel® Parallel Advisor tool is designed strictly for a C/C++ threaded-only programming application, and is not intended for an MPI or hybrid parallel programming application. Parallel Advisor needs a pre-installation of Microsoft Visual Studio 2005, 2008, or 2010 Standard Edition (or a higher edition) with the C++ compiler component installed)
- Intel® Trace Analyzer and Collector
- Intel® VTune™ Amplifier XE
- Integration of Intel® Visual Fortran Compiler within Microsoft* Visual Studio*
As noted in Figure 2.1, Intel® Cluster Studio XE 2012 for Windows* OS, provides Visual Studio 2010 Shell. As a prerequisite, if you will be installing the included Microsoft Visual Studio 2010 Shell, additional Microsoft software may be required to be installed prior to beginning the installation of Intel® Cluster Studio XE 2012. **Microsoft .NET 4.0 Framework is required for Microsoft Visual Studio 2010 Shell. If you do not already have this installed, you can download the installer:**

[.NET 4.0 Framework 32-bit and 64-bit](#)

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

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI</td>
<td>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.</td>
</tr>
<tr>
<td>BLACS</td>
<td>Basic Linear Algebra Communication Subprograms – provides a linear algebra oriented message passing interface for distributed memory computing platforms.</td>
</tr>
<tr>
<td>BLAS</td>
<td>Basic Linear Algebra Subroutines</td>
</tr>
<tr>
<td>DAPL*</td>
<td>Direct Access Program Library - an Application Program Interface (API) for Remote Data Memory Access (RDMA).</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>Ethernet</td>
<td>Ethernet is the predominant local area networking technology. It is transports data over a variety of electrical or optical media. It transports any of several upper layer protocols through data packet transmissions.</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>ICS</td>
<td>Intel® Cluster Studio</td>
</tr>
<tr>
<td>ICSXE</td>
<td>Intel® Cluster Studio XE</td>
</tr>
<tr>
<td>IMB</td>
<td>Intel® MPI Benchmarks</td>
</tr>
<tr>
<td>IP</td>
<td>Internet protocol</td>
</tr>
<tr>
<td>ITA or ita</td>
<td>Intel® Trace Analyzer</td>
</tr>
<tr>
<td>ITAC or itac</td>
<td>Intel® Trace Analyzer and Collector</td>
</tr>
<tr>
<td>ITC or itc</td>
<td>Intel® Trace Collector</td>
</tr>
<tr>
<td>MPD</td>
<td>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.</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface - an industry standard, message-passing protocol that typically uses a two-sided send-receive model to transfer messages between processes.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS</td>
<td>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.</td>
</tr>
<tr>
<td>PVM*</td>
<td>Parallel Virtual Machine</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDMA</td>
<td>Remote Direct Memory Access - this capability allows processes executing on one node of a cluster to be able to &quot;directly&quot; access (execute reads or writes against) the memory of processes within the same user job executing on a different node of the cluster.</td>
</tr>
<tr>
<td>RDSSM</td>
<td>TCP + shared memory + DAPL* (for SMP clusters connected through RDMA-capable fabrics)</td>
</tr>
<tr>
<td>RPM*</td>
<td>Red Hat Package Manager* - a system that eases installation, verification, upgrading, and uninstalling Linux* packages.</td>
</tr>
<tr>
<td>ScaLAPACK*</td>
<td>SCAlable LAPACK - an acronym for Scalable Linear Algebra Package or Scalable LAPACK.</td>
</tr>
<tr>
<td>shm</td>
<td>Shared memory only (no sockets)</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric Multi-processor</td>
</tr>
<tr>
<td>ssm</td>
<td>TCP + shared memory (for SMP clusters connected through Ethernet)</td>
</tr>
<tr>
<td>STF</td>
<td>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.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol - a session-oriented streaming transport protocol which provides sequencing, error detection and correction, flow control, congestion control and multiplexing.</td>
</tr>
<tr>
<td>VML</td>
<td>Vector Math Library</td>
</tr>
<tr>
<td>VSL</td>
<td>Vector Statistical Library</td>
</tr>
</tbody>
</table>

[Back to Table of Contents]
3. Intel Software Downloads and Installation on Microsoft Windows Compute Cluster Server (Microsoft Windows CCS*) OS

In terms of the Intel® Cluster Studio XE software for Windows* OS, consider references within this document to Microsoft Windows CCS* OS, and Microsoft* Windows* HPC Server 2008 OS, and Microsoft* Windows* HPC Server 2008 R2 OS as interchangeable.

Intel® Cluster Studio XE 2012 for Windows OS, provides Visual Studio 2010 Shell. As a prerequisite, if you will be installing the included Microsoft Visual Studio 2010 Shell, additional Microsoft software may be required to be installed prior to beginning the installation of Intel® Cluster Studio XE 2012. **Microsoft .NET 4.0 Framework is required for Microsoft Visual Studio 2010 Shell. If you do not already have this installed, you can download the installer:**

**.NET 4.0 Framework 32-bit and 64-bit**

The Intel Cluster Studio XE installation process on Microsoft Windows CCS is comprised of invoking an installer wizard. The Intel Cluster Studio 2012 package consists of the following software components which have a folder structure which may look something like the following:

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Default Installation Directory on Intel® 64 Architecture for Microsoft Windows CCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler XE 12.1</td>
<td>C:\Program Files (x86)\intel\Composer XE 2011 SP1</td>
</tr>
<tr>
<td>Intel® Fortran Compiler XE 12.1</td>
<td>C:\Program Files (x86)\intel\Composer XE 2011 SP1</td>
</tr>
<tr>
<td>Intel® Inspector XE 2011 Update 4</td>
<td>C:\Program Files (x86)\intel\Inspector XE 2011</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives 7.0 Update 5</td>
<td>C:\Program Files (x86)\intel\Composer XE 2011 SP1\ipp</td>
</tr>
<tr>
<td>Product</td>
<td>Path</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>Intel® Math Kernel Library (MKL) 10.3 Update 6</td>
<td>C:\Program Files (x86)\intel\Composer XE 2011 SP1\mkl</td>
</tr>
<tr>
<td>Intel® MPI Library 4.0 Update 3</td>
<td>C:\Program Files (x86)\intel\icsxe\2012.0.0xx\mpi</td>
</tr>
<tr>
<td>Intel® MPI Benchmarks 3.2.3</td>
<td>C:\Program Files (x86)\intel\icsxe\2012.0.0xx\imb</td>
</tr>
<tr>
<td>Intel® Parallel Advisor 2011 Update 3</td>
<td>C:\Program Files (x86)\Intel\Parallel Studio 2011\Advisor</td>
</tr>
<tr>
<td>Intel® Threading Building Blocks 4.0</td>
<td>C:\Program Files (x86)\intel\Composer XE 2011 SP1\tbb</td>
</tr>
<tr>
<td>Intel® Trace Analyzer and Collector 8.0 Update 3</td>
<td>C:\Program Files (x86)\intel\icsxe\2012.0.0xx\itac</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier XE 2011 Update 3</td>
<td>C:\Program Files (x86)\intel\VTune Amplifier XE 2011</td>
</tr>
</tbody>
</table>

For the table above, references to 0xx in the folder 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.

If the installation directory requires system administrative write privileges (for example, C:\Program Files (x86)\intel on Microsoft Windows CCS), and your login account does not have administrative access, you may need assistance from your system administrator in installing the associated software packages on your cluster system.

Back to Table of Contents

### 3.1 Microsoft Windows CCS* OS Installation

The Microsoft Windows CCS installer package for the Intel Cluster Studio XE has the following general nomenclature:

```
w_ics_<major>..<update>..<package_num>..exe
```

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

```
2012.0.xxx
```
The \texttt{\textless package\_num\textgreater} meta-symbol is a string such as 037. This string indicates the package number.

A typical name might look something like:

\texttt{w\_ics\_2012.1.037.exe}

For the Intel Cluster Studio XE installation process on Microsoft Windows CCS, go to the staging folder where the installation package is located and left-click on your mouse to start the installation process. Figure 3.1 through Figure 3.9 illustrate key milestones in the installation process.

\textbf{NOTE:} The left side bar provides a progress report on which stage the installation process is currently in.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{welcome_panel.png}
\caption{Figure 3.1 – Welcome Panel for Intel® Cluster Tools Installer – Press the Next Button on your System Where the Install is Taking Place}
\end{figure}
Figure 3.2 – End User License Agreement Panel – On your System, Select the Radial Button to Accept the Terms of the License Agreement and then Press Next
Figure 3.3 – Select the “Choose alternative activation” Radial Button, if you have a License File, and then Press Next
Figure 3.4 – Select the “Use the license file” Radial Button, if you have a License File, and then Press Next
Figure 3.5 – Provide a Folder Path to the License File, and then Press Next
Figure 3.6 – Select the “Full installation (recommended)” Radial Button, and then Press Next
Figure 3.7 – Installation Options – Press the Install Button
Figure 3.8 – The “Installation” Process
By default, the global root directory for the installation of the Intel Cluster Studio XE is:

\texttt{C:\Program Files (x86)\intel\icsxe\<major>\.<update>\.<package_num>}

where \texttt{<major>}, \texttt{<update>}, and \texttt{<package_num>} are integers. An example would be 2012.0.037.

Within the folder path \texttt{C:\Program Files (x86)\intel\icsxe\<major>\.<update>\.<package_num>} you will find the text files:

\texttt{ictvars.bat}

and

\texttt{icsxesupport.txt}

The text file:

\texttt{ictvars.bat}
contains environment variables for the each of the cluster tools. Sourcing this file will initialize these environment variables and will make it easy to work with the cluster tools as an aggregate. See the topic Navigating through the Intel® Cluster Tools after the Install has been Completed for how to automatically initialize these environment variables.

The file called:

icsxesupport.txt

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

Back to Table of Contents

3.2 Silent Install Mode

The Intel® Cluster Studio XE installer has a command-line window capability where the silent installation mode can be used, and it requires very little human interaction. This helps you to install the Intel Cluster Studio XE software on clusters that contain many nodes.

For example, in regards to clusters running Microsoft Windows Compute Cluster Server* (CCS*) OS or Microsoft* Windows* HPC Server 2008 OS, use the following steps to install the Intel® Cluster Studio XE software through the silent install mode feature:

1. Go to the head node of the cluster.
2. Copy the license file into the folder path C:\Program Files(x86)\Common Files\Intel\Licenses.
3. Open a command-line window and go to the folder where the Intel® Cluster Studio XE installer package is located.
4. Within the command-line window, execute the following command:
   
   w_ics_<major>.<update>.<package_num>.exe --silent -a install --eula=accept --output=<logfile>

   An example install command-line might be something like:
   
   w_ics_2012.0.037.exe --silent -a install --eula=accept --output=install.log

   The silent installation is started and completed automatically.

   For step number 4 above, the command-line option --a connotes an action and therefore "-a install" after --silent means that the action is to do an install of the software package. The --output switch is used to specify the name of the installation log file.

   For clusters using Microsoft Windows CCS OS or Microsoft Windows HPC Server 2008 OS, the command interface install process described above, distributes all the Intel® Cluster Studio XE software components onto all the compute nodes and the head node of the cluster.

Back to Table of Contents
3.3 Navigating through the Intel® Cluster Tools after the Install has been Completed

Once the installation is completed, you can begin working with the Intel® Cluster Tools by:

1) Open up a DOS window by clicking on the “Build Environment for Intel Cluster Studio XE” shortcut that was created by the cluster tools installer, or

2) Go to the Start menu and follow a path of **All Programs->Intel(R) Software Development Products->Intel(R) Cluster Studio 2012 XE->Build Environment for Intel Cluster Studio XE 2012** as illustrated in Figure 3.10.
If you make the selection **Build Environment for Intel Cluster Studio XE 2012**, a DOS panel similar to what is shown in Figure 3.11 will be created.
Figure 3.11 – DOS panel display that is created as a result of selecting “Build Environment for Intel Cluster Studio XE 2012”

This DOS panel sources the batch script file ictvars.bat that was described in subsection 3.1 for sourcing cluster tool environment variables. An example of such an environment that you might find useful is \textit{I\_MPI\_ROOT}. In regards to Figure 3.11, if you type the DOS command:

\begin{verbatim}
set I_
\end{verbatim}

the `set` command will print all defined variables that begin with the characters `I_`. Figure 3.12 illustrates the result of issuing this command where the environment variable \textit{I\_MPI\_ROOT} has the value:

\begin{verbatim}
c:\Program Files (x86)\Intel\MPI\4.0.3.007\em64t\bin\..
\end{verbatim}

This value is dependent on where the Intel Cluster Studio XE package was installed. Referencing the environment variable \texttt{\%I\_MPI\_ROOT\%} might be useful during compilation phases.
Figure 3.12 – Display of the environment variable I_MPI_ROOT which has a value such as c:\Program Files (x86)\Intel\MPI\4.0.3.007\em64t\bin\..\..

See chapter 4 of the Intel® Cluster Studio XE 2012 Getting Started Guide for the use of these environment variables. Compilation commands to transform a source code application into an executable can be done from such a DOS panel. Also, use the mpiexec command within a DOS panel can launch the MPI executable onto the nodes of the cluster.

See Figure 3.10, at the same level as the leaf menu item Build Environment for Intel Cluster Studio XE 2012 is the menu item Intel® Cluster Studio XE 2012 Documentation Map. If you select Intel® Cluster Studio XE 2012 Documentation Map, a panel display that looks something like Figure 3.13 will appear.
The documentation map 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. The 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.
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.
4. Integrated Development Environments for Intel® Cluster Studio XE

For Microsoft* Windows* OS, there are two integrated development environments (IDEs) by which you can develop software through Intel® Cluster Studio XE. These integrated development environments are Eclipse* for Intel® C++ Compiler XE, and Intel® Visual Fortran Compiler from the Microsoft Visual Studio* IDE.

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/

Alternatively, for doing development work with Intel® Fortran Compiler XE on Windows* OS, Intel® Cluster Studio XE comes packaged with Intel® Visual Fortran Compiler. Figure 4.1 illustrates the integrated development environment of Intel® Visual Fortran Compiler within Microsoft* Visual Studio*. Within Microsoft* Visual Studio*, follow the menu path Tools->Intel Visual Fortran Composer XE->Intel Visual Fortran Module Wizard (Figure 4.1).
Figure 4.1 - Intel® Visual Fortran Compiler within Microsoft* Visual Studio*

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 please refer 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.
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.

5.1 Launching MPD Daemons
The Intel MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. In order 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 Microsoft Windows CCS* OS

The command for launching multi-purpose daemons on Microsoft Windows is called “smpd”, which is an acronym for simple multi-purpose daemons. When Intel MPI Library is installed on a cluster, the smpd service is automatically started. On the master node of your Windows* cluster, you can type the command:

```bash
clusrun smpd -status | more
```

as demonstrated in Figure 4.2.

![Figure 5.2 – DOS command line syntax for issuing the smpd –status query](image)

For a four node cluster, you might see response that looks something like:
---------------------clusternode01 returns 0-----------------------
smpd running on clusternode01
---------------------clusternode02 returns 0-----------------------
smpd running on clusternode02
---------------------clusternode03 returns 0-----------------------
smpd running on clusternode03
---------------------clusternode04 returns 0-----------------------
smpd running on clusternode04

For this example, the four nodes of the cluster are respectively called clusternode1, clusternode2, clusternode3, and clusternode4.

From the master node one can stop all of the smpd daemons by typing the command:

    clusrun smpd -uninstall

To restart the daemons from the master node, simply type:

    clusrun smpd -install

or

    clusrun smpd -regserver

To verify that the smpd daemons are running properly, simply repeat the command:

    clusrun smpd -status | more

To shut down the smpd daemons, on all of the nodes of the cluster, you can type:

    clusrun smpd -remove

or

    clusrun smpd -unregserver

or

    clusrun smpd -uninstall

In general to see the various options for the smpd command, simply type:

    smpd -help

Back to Table of Contents

5.3 Compiling and Linking with Intel® MPI Library on Microsoft Windows CCS* OS

This section describes the basic steps required to compile and link an MPI program, when using 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. For example, regarding the Intel® C++ Compilers 12.1 and Intel® Fortran Compilers 12.1, execution of the appropriate set-up scripts will do this automatically:

```bash
...\bin\iclvars.bat
```

and

```bash
...\bin\ifortvars.bat
```

If you have added the architecture argument `ia32` to the `iclvars.bat` and `ifortvars.bat` invocation as illustrated with the following examples:

```bash
...\bin\iclvars.bat ia32
```

and

```bash
...\bin\ifortvars.bat ia32
```

then you will also need to use the same type of argument for `icsvars.bat` as follows (See Figure 5.3):

```bash
...\icsvars.bat ia32
```

Figure 5.3 – Setting the Architecture to IA-32 on Intel® 64 Systems
To revert back to the Intel® 64 architecture, use the following command:

```bash
...\icsvars.bat
```

in your DOS session.

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

```bash
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`.

<table>
<thead>
<tr>
<th>Supplier of Core Compiler</th>
<th>MPI Compilation Command</th>
<th>Core Compiler Compilation Command</th>
<th>Compiler Programming Language</th>
<th>Support Application Binary Interface (ABI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visual C++*</td>
<td>mpicc</td>
<td>cl.exe</td>
<td>C/C++</td>
<td>32/64 bit</td>
</tr>
<tr>
<td>Compiler or Intel C++</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler 11.1, 12.0, and 12.1</td>
<td>mpiicl</td>
<td>cl.exe</td>
<td>C/C++</td>
<td>32/64 bit</td>
</tr>
<tr>
<td></td>
<td>mpiicc</td>
<td>icl.exe</td>
<td>C/C++</td>
<td>32/64 bit</td>
</tr>
<tr>
<td>Intel Fortran Compiler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.1, 12.0, and 12.1</td>
<td>mpif77</td>
<td>ifort.exe</td>
<td>Fortran 77 and Fortran 95</td>
<td>32/64 bit</td>
</tr>
<tr>
<td></td>
<td>mpif90</td>
<td>ifort.exe</td>
<td>Fortran 95 and Fortran 95</td>
<td>32/64 bit</td>
</tr>
<tr>
<td></td>
<td>mpifc</td>
<td>ifort.exe</td>
<td>Fortran 95 and Fortran 95</td>
<td>32/64 bit</td>
</tr>
<tr>
<td></td>
<td>mpiifort</td>
<td>ifort.exe</td>
<td>Fortran 77 and Fortran 95</td>
<td>32/64 bit</td>
</tr>
</tbody>
</table>

**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.

You can also use the Intel® C++ Compiler, the Microsoft Visual C++ Compiler*, or the Intel Fortran Compiler directly. For example, on the master node of the Microsoft Windows CCS cluster, go to a shared directory where the Intel® MPI Library test-cases reside. For the test-case `test.c`, one can build an MPI executable using the following command-line involving the Intel C++ Compiler:

```bash
icl /Fetestc /I"%I_MPI_ROOT%\em64t\include" test.c 
"%I_MPI_ROOT%\em64t\lib\impi.lib"
```
The executable will be called testc.exe. This is a result of using the command-line option /Fe. The /I option references the path to the MPI include files. The library path reference is for the MPI library.

    mpiexec -machinefile machines.Windows -n 4 testc.exe

The -machinefile parameter has a file name reference called machines.Windows. This file contains a list of node names for the cluster. The results might look something like:

Hello world: rank 0 of 4 running on clusternode1
Hello world: rank 1 of 4 running on clusternode2
Hello world: rank 2 of 4 running on clusternode3
Hello world: rank 3 of 4 running on clusternode4

If you have a version of the Microsoft Visual C++ Compiler that was not packaged with Microsoft* Visual Studio* 2010, type the following command-line:

    cl /Fetestc /I"%I_MPI_ROOT%\em64t\include" test.c
    "%I_MPI_ROOT%\em64t\lib\impi.lib" bufferoverflowU.lib

If you have a version of the Microsoft Visual C++ Compiler that was packaged with Microsoft* Visual Studio* 2010, type the following command-line:

    cl /Fetestc /I"%I_MPI_ROOT%\em64t\include" test.c
    "%I_MPI_ROOT%\em64t\lib\impi.lib"

Back to Table of Contents

5.4 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. Prior to 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

<table>
<thead>
<tr>
<th>Interconnection Device Fabric Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>shm</td>
</tr>
<tr>
<td>dapl</td>
</tr>
<tr>
<td>tcp</td>
</tr>
</tbody>
</table>

The environment variable `I_MPI_FABRICS` has the following syntax:

```
```

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

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

---

**5.5 Running an MPI Program Using Intel® MPI Library on Microsoft Windows CCS® OS**

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

```
mpiexec -n <# of processes> .\myprog.exe
```

When launching the `mpiexec` command, you may be prompted for an account name and a password which might look something like the following:

```
User credentials needed to launch processes:
account (domain\user) [clusternode1\user001]:
account (domain\user) [clusternode1\user001]: password:
```

In the DOS panel simply hit the return key for the user name if you do not want to change it (in this example it is `user001`), and then enter the password for the associated account.

The only required option for the `mpiexec` command is the `-n` option to set the number of processes. However, you will probably want to use the working directory `−wdir`, and `−machinefile` command-line options that have the following syntax:

```
−wdir <working directory>
```
You may find these command-line options useful, if the nodes of the cluster are using a file share for example.

If your MPI application is using a network fabric other than the default fabric (sock), use the \texttt{--env} option to specify a value to be assigned to the \texttt{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:

\begin{verbatim}
mpiexec -n <# of processes> --env I_MPI_FABRICS=shm:tcp .\myprog.exe
\end{verbatim}

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

\begin{verbatim}
mpiexec -n <# of processes> --env I_MPI_FABRICS=shm:dapl .\myprog.exe
\end{verbatim}

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

To generate a \texttt{machines.Windows} text file to be used with the \texttt{--machinefile} command-line option, the following methodology might be useful (Figure 5.4).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{compute_cluster_administrator.png}
\caption{Compute Cluster Administrator display panel within Microsoft Windows HPC Server 2008*}
\end{figure}
If you select the ComputeNodes link in the left sub-panel (Figure 5.4) and then proceed to suppress the shift key and click in each node listed in the center panel of Figure 5.4 you will get a highlighted list as shown in Figure 5.5. You can then press Ctrl-C to do a copy and transfer the information to a text file. Remove the extraneous text when you do a Ctrl-V such that node names exist as 1 per line.

Figure 5.5 – Highlighting the selected nodes and using Ctrl-C to copy the node names in anticipation of creating the machines.Windows file for the mpiexec command-line option –machinefile

This file can be saved into shared file space area that can be used by all of the nodes of the cluster. An example might be:

```
z:\cluster_file_share\machines.Windows
```

If the –machinefile command-line option is used with the mpiexec command, the machines.Windows file might be reference in the following manner:

```
mpiexec -n 12 –machinefile z:\cluster_file_share\machines.Windows ...
```

5.6 Experimenting with Intel® MPI Library on Microsoft Windows CCS* 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.

Back to Table of Contents
Recall that in Section 5.2 that the command for launching multi-purpose daemons on Microsoft Windows is called “smpd”, which is an acronym for simple multi-purpose daemons. Also note that when Intel MPI Library is installed on a cluster, the smpd service is automatically started. In part 5.2 it was mentioned that you could type the command:

```
clusrun smpd –status | more
```

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:

```
---------------------clusternode01 returns 0-----------------------
smpd running on clusternode01
---------------------clusternode02 returns 0-----------------------
smpd running on clusternode02
```

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 C and C++ test applications into executables using the following commands with respect to the Intel C++ compiler:

```
icl /Fetestc /I"%I_MPI_ROOT%\em64t\include" test.c
"%I_MPI_ROOT%\em64t\lib\impi.lib"

icl /Fetestcpp /I"%I_MPI_ROOT%\em64t\include" test.cpp /link
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib impicxx.lib
```

If you have a version of the Microsoft Visual C++ Compiler that was not packaged with Microsoft* Visual Studio* 2010, type the following respective command-lines for the C and C++ test applications:

```
c1 /Fetestc_vc /I"%I_MPI_ROOT%\em64t\include" test.c
"%I_MPI_ROOT%\em64t\lib\impi.lib" bufferoverflowU.lib

and

c1 /Fetestcpp_vc /I"%I_MPI_ROOT%\em64t\include" test.cpp /link
bufferoverflowU.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib
impicxx.lib
```
If you have a version of the Microsoft Visual C++ Compiler that was packaged with Microsoft* Visual Studio* 2010, type the following respective command-lines for the C and C++ test applications:

```plaintext
cl /Fetestc_vc /I"%I_MPI_ROOT%\em64t\include" test.c
"%I_MPI_ROOT%\em64t\lib\impi.lib"
```

and

```plaintext
cl /Fetestcpp_vc /I"%I_MPI_ROOT%\em64t\include" test.cpp /link
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib impicxx.lib
```

The executable for test.c will be called testc.exe and testc_vc.exe, and the executable for test.cpp will be called testcpp.exe testcpp_vc.exe. The executable names are a result of using the command-line option /Fe. The /I option references the path to the MPI include files. The library path reference for the Intel MPI library is given by:

```
"%I_MPI_ROOT%\em64t\lib\impi.lib"
```

Similarly for the two test cases called test.f and test.f90, enter the following two respective commands to build executables:

```plaintext
ifort /Fetestf /I"%I_MPI_ROOT%\em64t\include" test.f
"%I_MPI_ROOT%\em64t\lib\impi.lib"
```

```plaintext
ifort /Fetestf90 /I"%I_MPI_ROOT%\em64t\include" test.f90
"%I_MPI_ROOT%\em64t\lib\impi.lib"
```

**Issue mpiexec commands which might look something like the following:**

```plaintext
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testf.exe
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testf90.exe
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testc.exe
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testcpp.exe
```

and for the Microsoft Visual C++ executables:

```plaintext
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testc_vc.exe
mpiexec -n 12 -machinefile z:\cluster_file_share\machines.Windows -wdir z:\cluster_file_share\test testcpp_vc.exe
```

**The output from testcpp.exe should look something like:**

```
Hello world: rank 0 of 12 running on clusternode0
Hello world: rank 1 of 12 running on clusternode1
Hello world: rank 2 of 12 running on clusternode2
Hello world: rank 3 of 12 running on clusternode3
Hello world: rank 4 of 12 running on clusternode4
Hello world: rank 5 of 12 running on clusternode5
```
Hello world: rank 6 of 12 running on clusternode6
Hello world: rank 7 of 12 running on clusternode7
Hello world: rank 8 of 12 running on clusternode8
Hello world: rank 9 of 12 running on clusternode9
Hello world: rank 10 of 12 running on clusternode10
Hello world: rank 11 of 12 running on clusternode11

The above mpiexec commands assume that there is a file share called:

```
z:\cluster_file_share
```

If your system is using only symmetric multiprocessing on a shared memory system, then the mpiexec commands could omit the -machinefile and -wdir options.

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 –machinefile machines.Windows -env I_MPI_FABRICS dapl -n 2 testf.exe
mpiexec –machinefile machines.Windows -env I_MPI_FABRICS dapl -n 2 testf90.exe
mpiexec –machinefile machines.Windows -env I_MPI_FABRICS dapl -n 2 testc.exe
mpiexec –machinefile machines.Windows -env I_MPI_FABRICS dapl -n 2 testcpp.exe
mpiexec –machinefile machines.Windows -env I_MPI_FABRICS dapl -n 2 testcpp_vc.exe
```

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
```

5.7 Controlling MPI Process Placement on Microsoft Windows CCS* 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 SMP (symmetric multi-processor) 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:
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 2 of the processes will execute on clusternode1 and 2 will execute on clusternode2. For example, you might issue the following commands:

```
mpiexec -n 2 -host clusternode1 .	estf : -n 2 -host clusternode2 .	estf
mpiexec -n 2 -host clusternode1 .	estf90 : -n 2 -host clusternode2 .	estf90
mpiexec -n 2 -host clusternode1 .	estc : -n 2 -host clusternode2 .	estc
mpiexec -n 2 -host clusternode1 .	estcpp : -n 2 -host clusternode2 .	estcpp
```

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, then 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:** You would have to 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`.

**5.8 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.exe, testcpp.exe, testf.exe, and testf90.exe in the directory test_intel_mpi could be used. The command invocation for mpitune might look something like the following:

mpitune --host-file machines.Windows --output-file testc.conf --application "mpiexec -n 4 testc.exe"

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

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>Semantic Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a &quot;&lt;app_cmd_line&gt;&quot;</td>
<td>Switch on the application tuning mode. Quote the full command line as shown</td>
</tr>
<tr>
<td>-cm</td>
<td>--cluster-mode [exclusive</td>
</tr>
<tr>
<td>-d</td>
<td>--debug Print debug information</td>
</tr>
<tr>
<td>-dl [d1[,d2...[,dN]]]</td>
<td>--device-list [d1[,d2...[,dN]]] Select the device(s) you want to tune. By default use all of the devices mentioned in the &lt;installdir&gt;/&lt;arch&gt;/etc/devices.xml file</td>
</tr>
<tr>
<td>-er</td>
<td>--existing-ring Try to use an existing MPD ring. By default, create a new MPD ring</td>
</tr>
<tr>
<td>-fl [f1[,f2...[,fN]]]</td>
<td>--fabric-list [f1[,f2...[,fN]]] Select the fabric(s) you want to tune. By default use all of the fabrics mentioned in the &lt;installdir&gt;/&lt;arch&gt;/etc/fabrics.xml file</td>
</tr>
<tr>
<td>-h</td>
<td>--help Display a help message</td>
</tr>
<tr>
<td>-hf &lt;hostsfile&gt;</td>
<td>--host-file &lt;hostsfile&gt; Specify an alternative host file name. By default, use the $PWD/mpd.hosts</td>
</tr>
<tr>
<td>-hr</td>
<td>--host-range {min:max</td>
</tr>
<tr>
<td>-i &lt;count&gt;</td>
<td>--iterations &lt;count&gt; 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</td>
</tr>
<tr>
<td>-mh</td>
<td>--master-host Dedicate a single host to mpitune</td>
</tr>
</tbody>
</table>
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.

**5.8.1 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:

```
--message-range {min:max | min: | :max}  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 `min:` or `:max` format will use the default values as appropriate.

-of <file-name> | --output-file <file-name>  Specify the application configuration file to be generated in the application-specific mode. By default, use the `$PWD/app.conf`.

-od <outputdir> | --output-directory <outputdir>  Specify the directory name for all output files. By default, use the current directory. The directory should be accessible from all hosts.

-pr {min:max | min: | :max} | --ppn-range {min:max | min: | :max} | --perhost-range {min:max | min: | :max}  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 `min:` or `:max` format will use the default values as appropriate.

-sf [file-path] | --session-file [file-path]  Continue the tuning process starting from the state saved in the `file-path` session file.

-s | --silent  Suppress all diagnostic output.

-td <dir-path> | --temp-directory <dir-path>  Specify a directory name for the temporary data. By default, use the `$PWD/mpitunertemp`. This directory should be accessible from all hosts.

-t "\<test_cmd_line>" | --test "\<test_cmd_line>"  Replace the default Intel® MPI Benchmarks by the indicated benchmarking program in the cluster-specific mode. Quote the full command line as shown.

-tl <minutes> | --time-limit <minutes>  Set `mpitune` execution time limit in minutes. The default value is 0, which means no limitations.

-V | --version  Print out the version information.

```
mpitune -hf machines.Windows -of testc.conf --test "testc.exe"

could be used, where machines.Windows 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 proceed to 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.exe

5.8.2 MPI Application-Specific Tuning

The mpitune invocation:

    mpitune -hf machines.Windows -of testf90.conf --application "mpiexec -n 4 testf90.exe"

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.exe

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

You might want to use mpitune utility on each of the test applications testc.exe, testcpp.exe, testf.exe, and testf90.exe. For a complete discussion on how to use the mpitune utility, see the Tuning Reference section of the Intel MPI Library for Windows* 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 I_MPI_DEBUG Environment Variable

As mentioned previously (for example, Figure 2.1), debugging of an MPI application can be achieved with the I_MPI_DEBUG environment variable. The syntax of the I_MPI_DEBUG environment variable is as follows:

\[ \text{I_MPI_DEBUG} = \text{<level>} \]

where \text{<level>} can have the values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Debug Level Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not set</td>
<td>Print no debugging information</td>
</tr>
<tr>
<td>1</td>
<td>Print warnings if specified I_MPIDEVICE could not be used</td>
</tr>
<tr>
<td>2</td>
<td>Confirm which I_MPIDEVICE was used</td>
</tr>
<tr>
<td>&gt; 2</td>
<td>Add extra levels of debugging information</td>
</tr>
</tbody>
</table>

To simplify process identification add the operators “+” or “-” in front of the numerical value for I_MPI_DEBUG level. This setting produces debug output lines which are prepended with the MPI process rank, a process id, and a host name as defined at the process launch time. For example, the command:

\[ \text{mpiexec -n <# of processes> -env I_MPI_DEBUG +2 myprog.exe} \]

produces output debug messages in the following format:

I_MPI: [rank#pid@hostname]Debug message

You can also compile the MPI the application with the /Z7 or /Z8 compiler options.

Back to Table of Contents
7. Instrumenting MPI Applications with Intel® Trace Analyzer and Collector

MPI applications can be easily instrumented with the Intel Trace Collector Library to gather performance data, and postmortem performance analysis can be visually assessed with Intel Trace Analyzer. The Intel Trace Analyzer and Collector supports instrumentation of applications written in C, C++, Fortran 77, and the Fortran 95 programming languages.

Back to Table of Contents

7.1 Instrumenting the Intel® MPI Library Test Examples

Recall 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
```

In a scratch version of the folder called `test`, one can set the environment variable `VT_LOGFILE_PREFIX` to the following:

```
set VT_LOGFILE_PREFIX=test_inst
```

where `test_inst` is an acronym for test instrumentation. After doing this you can create a test instrumentation folder by typing the command:

```
mkdir %VT_LOGFILE_PREFIX%
```

To compile and instrument the Fortran files called `test.f` and `test.f90` using the Intel Fortran compiler, you can issue the following respective DOS commands:

```
ifort /Fetestf /I"%I_MPI_ROOT%"\em64t\include test.f /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" ìmpi.lib /NODEFAULTLIB:LIBCMTD.lib
```

and

```
ifort /Fetestf90 /I"%I_MPI_ROOT%"\em64t\include test.f90 /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" ìmpi.lib /NODEFAULTLIB:LIBCMTD.lib
```

To compile and instrument the respective C and C++ files `test.c` and `test.cpp` using the Intel C++ compiler, you can issue the following respective DOS commands:

```
icl /Fetestc /I"%I_MPI_ROOT%"\em64t\include test.c /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
```
For C++ applications, the Intel MPI library `impicxx.lib` is needed in addition to `impi.lib`.

Alternatively, to compile and instrument the respective C and C++ files `test.c` and `test.cpp` using a Microsoft* Visual Studio* C++ Compiler that was not packaged with Microsoft* Visual Studio* 2010, you can issue the DOS commands:

```plaintext
icl /Fetestcpp /I"%I_MPI_ROOT%"\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib impicxx.lib
/LNODEFAULTLIB:LIBCMTD.lib
and

icl /Fetestcpp_vc /I"%I_MPI_ROOT%"\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

NoTE: When compiling and linking with a Microsoft Visual Studio C++ Compiler that was not packaged with Microsoft* Visual Studio* 2008 or 2010, the library `bufferoverflowu.lib` has been added as demonstrated above for the C and C++ test cases.

If you have a version of the Microsoft Visual C++ Compiler that was packaged with Microsoft* Visual Studio* 2008 or 2010, type the following respective command-lines for the C and C++ test applications:

```plaintext
cl /Fetestc_vc /I"%I_MPI_ROOT%"\em64t\include test.c /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
and

cl /Fetestcpp_vc /I"%I_MPI_ROOT%"\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impicxx.lib impi.lib
/NODEFAULTLIB:LIBCMTD.lib
```

After issuing these compilation and link commands, the following executables should exist in the present working directory:

- `testc.exe`
- `testcpp.exe`
- `testcpp_vc.exe`
- `testc_vc.exe`
- `testf.exe`
Recall that the environment variable \texttt{VT\_LOGFILE\_PREFIX} was set to \texttt{test\_inst} which was used as part of a \texttt{mkdir} command to create a directory where instrumentation data is to be collected. One method of directing the \texttt{mpiexec} command to place the Intel Trace Collector data into the folder called \texttt{test\_inst} is to use the following set of commands for the executables above:

\begin{verbatim}
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testc
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testcpp
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testcpp\_vc
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testc\_vc
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testf
mpiexec -n 4 -env VT\_LOGFILE\_PREFIX test\_inst testf90
\end{verbatim}

For the executables above, four MPI processes are created via the \texttt{mpiexec} command. These \texttt{mpiexec} commands will produce the STF files:

\texttt{testc.stf testcpp.stf testcpp\_vc.stf testc\_vc.stf testf.stf testf90.stf}

within the directory \texttt{test\_inst}.

Issuing the \texttt{traceanalyzer} command on the STF file \texttt{test\_inst\testcpp\_vc.stf} as follows:

\begin{verbatim}
traceanalyzer test\_inst\testcpp\_vc.stf
\end{verbatim}

will generate a profile panel which looks something like the following:
Figure 7.1 – The Profile display for testcpp_vc.stf

Figure 7.2 shows the Event Timeline display which results when following the menu path **Charts**->**Event Timeline** within Figure 7.1.
An alternative to the above mpiexec commands is to create a trace collector configuration file such as vtconfig.txt which could have the contents, beginning in column 1, of:

```
logfile-prefix test_inst
```

The directive called logfile-prefix is analogous to the Intel Trace Collector environment variable VT_LOGFILE_PREFIX. In general, you can place multiple Intel Trace Collector directives into this vtconfig.txt file. For additional information about Intel Trace Collector directives, you should look at Chapter 9 of <directory-path-to-ITAC>\doc\ITC_Reference_Guide.pdf. The file vtconfig.txt can be referenced by the mpiexec commands through the Intel Trace Collector environment variable directive called VT_CONFIG as follows:

```
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testc
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testcpp
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testcpp_vc
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testc_vc
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testf
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testf90
```
7.2 Instrumenting the Intel® MPI Library Test Examples in a Fail-Safe Mode

There may be situations where an application will end prematurely, and thus trace data could be lost. The Intel Trace Collector has a trace library that works in fail-safe mode.

To compile and instrument the Fortran files called test.f and test.f90 using the Intel Fortran compiler, you can issue the following respective DOS commands:

```bash
ifort /Fetestf_fs /I"%I_MPI_ROOT%"\em64t\include test.f /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

and

```bash
ifort /Fetestf90_fs /I"%I_MPI_ROOT%"\em64t\include test.f90 /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

where the special Intel Trace Collector Library for fail-safe (acronym fs) tracing is – VTfs.lib.

To compile and instrument the respective C and C++ files test.c and test.cpp using the Intel® C++ Compiler, you can issue the following respective DOS commands:

```bash
icl /Fetestc_fs /I"%I_MPI_ROOT%"\em64t\include test.c /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

and

```bash
icl /Fetestcpp_fs /I"%I_MPI_ROOT%"\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib impicxx.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

For C++ applications, the Intel MPI library impicxx.lib is needed in addition to impi.lib.

Alternatively, to compile and instrument the respective C and C++ files test.c and test.cpp using a Microsoft® Visual Studio® C++ compiler that was not packaged with Microsoft® Visual Studio® 2010, you can issue the DOS commands:

```bash
cl /Fetestc_fs_vc /I"%I_MPI_ROOT%"\em64t\include test.c /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

and

```bash
cl /Fetestcpp_fs_vc /I"%I_MPI_ROOT%"\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```
NOTE: When compiling and linking with a Microsoft Visual Studio C++ compiler that was not packaged with Microsoft* Visual Studio* 2008 or 2010, the library bufferoverflowu.lib has been added as demonstrated above for the C and C++ test cases.

If you have a version of the Microsoft Visual C++ Compiler that was packaged with Microsoft* Visual Studio* 2008 or 2010, type the following respective command-lines for the C and C++ test applications:

cl /Fetestc_fs_vc /I"%I_MPI_ROOT%\em64t\include" test.c /link /LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib

and

c1 /Fetestcpp_fs_vc /I"%I_MPI_ROOT%\em64t\include" test.cpp /link /LIBPATH:"%VT_LIB_DIR%" VTfs.lib Ws2_32.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impicxx.lib impi.lib /NODEFAULTLIB:LIBCMTD.lib

After issuing these compilation and link commands, the following executables should exist in the present working directory:

testc_fs.exe
testcpp_fs.exe
testcpp_fs_vc.exe
testc_fs_vc.exe
testf_fs.exe
testf90_fs.exe

Recall that the environment variable VT_LOGFILE_PREFIX was set to test_inst which was used as part of a mkdir command to create a directory where instrumentation data is to be collected. One method of directing the mpiexec command to place the Intel Trace Collector data into the folder called test_inst is to use the following set of commands for the executables above:

mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_fs
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_fs
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_fs_vc
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_fs_vc
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf_fs
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf90_fs

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

Complete user documentation regarding VTfs.lib 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 vtfs as a search phrase within the documentation.

Back to Table of Contents

7.3 Using itcpin to Instrument an Application

The itcpin utility is a binary instrumentation tool that comes with Intel Trace Analyzer and Collector. The Intel® architecture for itcpin on Microsoft Windows must be 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.

--profile (off)

Enables function profiling in the instrumented binary. Once enabled, all functions in the executable will be traced. It is recommended to control this to restrict the runtime overhead and the amount of trace data by disabling functions which do not need to be traced.

To obtain a list of all of the itcpin 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.

mpiicc pi.c /debug:all

set VT_LOGFILE_PREFIX=itcpin_inst

rmdir /S /Q %VT_LOGFILE_PREFIX%

mkdir %VT_LOGFILE_PREFIX%

mpiexec -mapall -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILEPrefix "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" -env VT_STATE "*.dll*: off" itcpin --run --profile -- pi.exe

where the environment variables that are being set for the mpiexec command are:

- env VT_DLL_DIR "%VT_DLL_DIR%" - env VT_MPI_DLL "%VT_MPI_DLL%" - env VT_LOGFILE_FORMAT STF - env VT_PCTRACE 5 - env VT_LOGFILE_PREFIX
"%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" -env VT_STATE "*.dll:* off"

The itcpin utility is included within the mpiexec command, and the itcpin options that are used are --run and --profile. The DOS 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. Also, the mpiicc batch script for compiling pi.c uses the /debug:all compiler option to create an executable that includes debug information which aids the instrumentation process.

An explanation for these instrumentation environment variables can be found in the Intel Trace Collector Users’ Guide under the search topic “ITC Configuration”.

The DOS shell commands above could be packaged into a DOS batch script. The output from the above sequence of DOS Shell commands looks something like the following:

Process 0 of 4 on cluster01
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
Wall clock time = 0.068866
[0] Intel(R) Trace Collector INFO: Writing tracefile pi.stf in
  Z:\test\itcpin_inst
Process 2 of 4 on cluster03
Process 3 of 4 on cluster04
Process 1 of 4 on cluster02

The exact output will be a function of your cluster configuration.

Figure 7.4 shows the timeline and function panel displays that were generated from the instrumentation data that was stored into the directory itcpin_inst as indicated by the environment variable VT_LOGFILE_PREFIX. The command that initiated the Intel Trace Analyzer with respect to the current directory was:

```
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 itcpin 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

7.4 Working with the Intel® Trace Analyzer and Collector Examples

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

C:\Program Files (x86)\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:
nmake distclean

nmake all MPIDIR="c:\Program Files (x86)\Intel\ICSXE\2012.0.037\MPI\em64t"

The makefile variable MPIDIR is explicitly set to the folder path where the version of Intel MPI Library resides that supports 64-bit address extensions. This set of commands will respectively clean up the folder content and compile and execute the following C and Fortran executables:

- mpiconstants.exe
- vnallpair.exe
- vnallpairc.exe
- vnjacobian.exe
- vnjacobianf.exe
- vtallpair.exe
- vtallpairc.exe
- vtcounterscopec.exe
- vtjacobian.exe
- vtjacobianf.exe
- vttimef.exe

of which the following STF files are created:

- timertest.stf
- vtallpair.stf
- vtallpairc.stf
- vtcounterscopec.stf
- vtjacobian.stf
- vtjacobianf.stf

If one invokes Intel Trace Analyzer with the command:

```
traceanalyzer vtjacobian.stf
```

the following display panel will appear (Figure 7.5):
Figure 7.5 - Intel Trace Analyzer Display for vtjacobc.stf

Figure 7.6 shows the Event Timeline display which results when following the menu path **Charts->Event Timeline** within Figure 7.5.
Figure 7.6 - 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

7.5 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 VTfs.lib 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 VTmc.lib no events are recorded and no trace file is written. This option enables recording of all events also supported by the normal VT.lib and the writing of a trace file. The trace file will also contain the errors found during the run.

Complete user documentation regarding message checking 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".

At the URL:

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

you can obtain the source to an MPI example using C bindings that demonstrates deadlock. This C programming language test case is called deadlock.c.

To compile and instrument deadlock.c using the Intel C++ Compiler, you can issue the following DOS command can be used:

icl /D_CRT_SECURE_NO_DEPRECATE /Fedeadlock
/I"%I_MPI_ROOT%\include /Zi deadlock.c /link /stack:8000000 /LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" mpi.lib /NODEFAULTLIB:LIBCMTD.lib

Alternatively, to compile and instrument deadlock.c using the Microsoft Visual Studio C++ compiler from say Microsoft Visual Studio 2005, you can issue the DOS command:

c1 /D_CRT_SECURE_NO_DEPRECATE /Fedeadlock_vc
/I"%I_MPI_ROOT%\include /Zi deadlock.c /link /stack:8000000 /LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib bufferoverflowu.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" mpi.lib /NODEFAULTLIB:LIBCMTD.lib

If the C++ compiler supplied with Microsoft Visual Studio 2008 or 2010 is used, you can issue the DOS command:

c1 /D_CRT_SECURE_NO_DEPRECATE /Fedeadlock
/I"%I_MPI_ROOT%\include /Zi deadlock.c /link /stack:8000000 /LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" mpi.lib /NODEFAULTLIB:LIBCMTD.lib

where the library bufferoverflowu.lib is omitted. For all three compilation scenarios, the option /Zi was used to instruct the compiler to insert symbolic debug information into an object file.

Before issuing an mpiexec command, a scratch folder for trace information called test_inst, can be created with the VT_LOGFILE_PREFIX environment variable by using the following process:
set VT_LOGFILE_PREFIX=test_inst

After doing this you can create a test instrumentation folder by typing the command:

mkdir %VT_LOGFILE_PREFIX%

When issuing the mpiexec command with settings for the VT_DEADLOCK_TIMEOUT, VT_DEADLOCK_WARNING, and VT_CHECK_TRACING environment variables two command-line examples are demonstrated. For execution of deadlock.exe on a local drive the mpiexec command might look something like:

mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -wdir "C:\MPI Share\MPI_Share_Area\test_correctness_checking" -n 2 .\deadlock.exe 0 80000

Alternatively, for a mapped drive that is shared on all nodes of the cluster, the mpiexec command might look something like:

mpiexec -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -mapall -wdir "Z:\MPI Share\MPI_Share_Area\test_correctness_checking" -n 2 .\deadlock.exe 0 80000

The execution diagnostics might look something like the following:

```plaintext
...
[0] ERROR: no progress observed in any process for over 1:00 minutes, aborting application
[0] WARNING: starting emergency trace file writing
[0] ERROR: GLOCAL:DEADLOCK:HARD: fatal error
[0] ERROR: Application aborted because no progress was observed for over 1:00 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
[0] ERROR: blocked
[0] ERROR: because the MPI might wait for the corresponding receive).
[0] ERROR: [0] no progress observed for over 1:00 minutes, process is currently in MPI call:
[0] ERROR: __MPI_Recv(*buf=00000000004D2A80, count=800000, datatype=MPI_INT,
[0] ERROR: source=1, tag=999, comm=MPI_COMM_WORLD, *status=00000000007DFE80)
[0] ERROR: main (C:\MPI Share\MPI_Share_Area\test_correctness_checking\deadlock.c:59)
[0] ERROR: __tmainCRTStartup
(f:\dd\vctools\crt_bld\self_64_amd64\crt\src\crt0.c:266)
[0] ERROR: BaseThreadInitThunk (kernel32)
[0] ERROR: RtlUserThreadStart (ntdll)
[0] ERROR: ()
[0] ERROR: [1] no progress observed for over 1:00 minutes, process is currently in MPI call:
[0] ERROR: __MPI_Recv(*buf=00000000004D2A80, count=800000, datatype=MPI_INT,
[0] ERROR: source=0, tag=999, comm=MPI_COMM_WORLD, *status=00000000007DFE80)
[0] ERROR: main (C:\MPI Share\MPI_Share_Area\test_correctness_checking\deadlock.c:59)
[0] ERROR: __tmainCRTStartup
(f:\dd\vctools\crt_bld\self_64_amd64\crt\src\crt0.c:266)
[0] ERROR: BaseThreadInitThunk (kernel32)
[0] ERROR: RtlUserThreadStart (ntdll)
[0] ERROR: ()
```
INFO: Writing tracefile deadlock.stf in 2:\MPI_Share_Area\test_correctness_checking\test_inst

INFO: GLOBAL:DEADLOCK:HARD: found 1 time (1 error + 0 warnings), 0 reports were suppressed
INFO: Found 1 problem (1 error + 0 warnings), 0 reports were suppressed.

1/2: receiving 80000
0/2: receiving 80000
job aborted:
rank: node: exit code[: error message]
0: clusternode1: 1: process 0 exited without calling finalize
1: clusternode2: 1: process 1 exited without calling finalize

Trace information was placed into the directory referenced by VT_LOGFILE_PREFIX because the environment variable VT_CHECK_TRACING was set for the mpiexec command.

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):
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

One 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:

```c
... 49   MPI_Recv (buffer_in, MAX_ARRAY_LENGTH, MPI_INT, other, 999,
```
This is illustrated in Figure 7.11. To avoid deadlock situations, you can resort to the following solutions:

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:

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

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

```c
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 with the Microsoft Visual C++ compiler:
cl /D_CRT_SECURE_NO_DEPRECATE /Fedeadlock2_vc /I"%I_MPI_ROOT%\em64t\include /Zi deadlock2.c /link /stack:8000000 /LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib bufferoverflowu.lib /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib

then the result of invoking the mpiexec command for deadlock2_vc.exe:

mpiexec -genv VT_DEADLOCK_TIMEOUT 20s -genv VT_DEADLOCK_WARNING 25s -n 2 -genv VT_CHECK_TRACING on .\deadlock2_vc.exe 0 80000

is the following:

[0] INFO: Error checking completed without finding any problems.

1/2: receiving 80000
1/2: sent 80000
0/2: receiving 80000
0/2: sent 80000

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, we 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

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):
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 7.13, a developer may want to know a summary of process imbalance for the executable. One 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

Clicking on the OK button in the subpanel will generate the following (Figure 7.17). You can verify the meaning of the histogram subcomponents by pressing 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
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”.

Notice 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, 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.
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. The simulator API can be find in the folder path:

```
<directory-path-to-ITAC>\examples\icpf\n```

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

If you encounter the following link error message:

```
LINK : fatal error LNK1181: cannot open input file 'bufferoverflowu.lib'
```

when creating executables for the experiments in this chapter and your `nmake` command is not part of Microsoft* Visual Studio* 2008 or 2010, please source the .bat file:

```
vccvarsx86_amd64.bat
```

in your DOS command-line window where you are doing the Intel® Math Kernel Library experiments. This .bat file should be located in a bin subfolder within the Microsoft* Visual Studio* folder path and the DOS command for sourcing this file might look something like the following:

```
"C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\x86_amd64\vccvarsx86_amd64.bat"
```

where the line above is contiguous.

If for the ScaLAPACK* experiments in this chapter, you are using a version of `nmake` from Microsoft Visual Studio 2008 or Microsoft Visual Studio 2010, then you can use the ScaLAPACK makefile variable `msvs=2008` or `msvs=2010` to prevent the link error referenced above. The setting of `msvs=2008` or `msvs=2010` will instruction the ScaLAPACK makefile to not use the library `bufferoverflowu.lib`.

Back to Table of Contents

8.1 Experimenting with ScaLAPACK*

On Microsoft Windows CCS, the MKL installation might be in the folder path:

```
C:\Program Files (x86)\intel\Composer XE 2011 SP1\mk1\n```

where `xxx` is the build number of the Intel® Cluster Studio XE 2012 package. The contents of `...\mk1` sub-folder should be:
To experiment with the ScaLAPACK (SCAlable LAPACK) 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
```

To build and run the ScaLAPACK executables, you can type the command:

```
nmake cleanall
nmake msvs=2010 arch=intel64 mpi=msmpi mpdir="c:\Program Files\Microsoft HPC Pack 2008 SDK" libdir="c:\program files (x86)\intel\Composer XE 2011 SP1\mkl" libtype=static run > msmpi.scalapack.static.report 2>&1
```

if you are using `nmake` from Microsoft* Visual Studio* 2010. Substitute in `msvs=2008`, if you have Microsoft* Visual Studio* 2008. Otherwise use the command:

```
nmake cleanall
nmake mpimapall=y arch=intel64 mpi=intelmpi mpdir="%I_MPI_ROOT%" libdir="c:\program files (x86)\intel\Composer XE 2011 SP1\mkl" libtype=static run > scalapack.static.report 2>&1
```

In the `scalapack` working directory where the `nmake` command was issued, the ScaLAPACK executables can be found in

```
“_results\_intel64_static_intelmpi_lp64
```

and the results of the computation will also be placed into this same sub-directory. The results will
be placed into "*.txt" files. You can invoke an editor to view the results in each of the "*.txt" files that have been created.

As an example result, "_results\_intel64\_static\_intelmpi\_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.exe.

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 = ||Ax - b|| / (||x|| * ||A|| * eps * N)
Factorization residual = ||A - LU|| / (||A|| * 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

<table>
<thead>
<tr>
<th>TIME</th>
<th>TR</th>
<th>N</th>
<th>BWL</th>
<th>BWU</th>
<th>NB</th>
<th>NRHS</th>
<th>P</th>
<th>Q</th>
<th>L^U Time</th>
<th>Slv Time</th>
<th>MFLOPS</th>
<th>MFLOP2</th>
<th>CHECK</th>
</tr>
</thead>
<tbody>
<tr>
<td>WALL</td>
<td>N</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.0003</td>
<td>0.45</td>
<td>0.43</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.0003</td>
<td>0.82</td>
<td>0.77</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.0003</td>
<td>2.77</td>
<td>2.63</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0.000</td>
<td>0.0004</td>
<td>0.05</td>
<td>0.07</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0.000</td>
<td>0.0004</td>
<td>0.56</td>
<td>0.84</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0.000</td>
<td>0.0004</td>
<td>1.96</td>
<td>2.97</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0.000</td>
<td>0.0005</td>
<td>0.24</td>
<td>0.35</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0.000</td>
<td>0.00038</td>
<td>0.09</td>
<td>0.16</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0.000</td>
<td>0.0009</td>
<td>0.82</td>
<td>1.25</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.001</td>
<td>0.0011</td>
<td>0.13</td>
<td>0.19</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.001</td>
<td>0.0011</td>
<td>0.19</td>
<td>0.33</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>N</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.001</td>
<td>0.0041</td>
<td>0.27</td>
<td>0.42</td>
<td>PASSED</td>
</tr>
</tbody>
</table>

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 "results\_intel64\_static\_intelmpi\_lp64\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](chart.png)

You can also link the libraries dynamically. The following command will provide for this if `nmake` is part of Microsoft Visual Studio 2010:

```
nmake msvs=2010 mpimapall=y arch=intel64 mpi=intelmpi mpidir="%I_MPI_ROOT%" libdir="c:\program files (x86)\intel\Composer XE 2011 SP1\mkl" libtype=dynamic run > scalapack.dynamic.report 2>&1
```

Substitute in `msvs=2008`, if you have Microsoft* Visual Studio* 2008. Otherwise, use the command:

```
nmake mpimapall=y arch=intel64 mpi=intelmpi mpidir="%I_MPI_ROOT%" libdir="c:\program files (x86)\intel\Composer XE 2011 SP1\mkl" libtype=dynamic run > scalapack.dynamic.report 2>&1
```

Before issuing the command above you should clean up from the static library build. This can be done by using the following `nmake` command:
8.2 Experimenting with the Cluster DFT Software

On Microsoft Windows CCS, in the folder path:

<directory-path-to-mkl>\examples

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

The two sub-folders that will be discussed here are cdftc and cdftf. These two directories respectively contain C and Fortran programming language examples that can be built and executed for the Cluster Discrete Fourier Transform (CDFT). Within each of these folders, there is a help target built within the makefile, and therefore you can type:

nmake cleanall
To do experimentation with the contents of these two folders within a DOS window, you can issue the `nmake` commands:

```
nmake libintel64 mpi=intelmpi MIRUNOPTS="-n 4 -machinefile "z:\global machine files folder\machines.Windows\"" workdir="z:\MPI_Share_Area\cdftc_test" SPEC_OPT="/debug:all"
```

and

```
nmake libintel64 mpi=intelmpi MIRUNOPTS="-n 4 -machinefile "z:\global machine files folder\machines.Windows\"" workdir="z:\MPI_Share_Area\cdftf_test" SPEC_OPT="/debug:all"
```

where the `MIRUNOPTS` macro is used. In general for the `cdftc` and `cdftf` makefiles, the `MIRUNOPTS` macro is used to pass command-line arguments to the `mpiexec` command. For the `nmake` command-line examples above, the `MIRUNOPTS` macro is being used to override the default number of MPI processes (where two processes is the default), and we are providing a `-machinefile` argument to select which nodes of the cluster, the MPI processes will run on. Note that there are spaces in the subfolder name "global machine files folder", and therefore the folder path name `z:\global machine files folder\machines.Windows` is preceded and followed by the escape character sequence `\"` and so the `-machinefile` folder argument is:

```
"z:\global machine files folder\machines.Windows\"
```

The first `nmake` command listed above should be used for the folder `cdftc`, and the second `nmake` command should be used for the folder `cdftf`. The `nmake` commands are each contiguous lines that end with `workdir="z:\MPI_Share_Area\cdftc_test"`. These commands reference the makefile target `libem64t`, and the makefile variables `mpi, mpidir, and workdir`. You can obtain complete information about this makefile by looking at its contents within the folders `...\cdftc` and `...\cdftf`. Note that on your cluster system, the test directories `z:\MPI_Share_Area\cdftc_test` and `z:\MPI_Share_Area\cdftf_test` may be substituted with folder paths that you may prefer to use.

After executing the `nmake` commands above within the respective folders `...\cdftc` and `...\cdftf`, the `workdir` folders `z:\MPI_Share_Area\cdftc_test` and `z:\MPI_Share_Area\cdftf_test` should each have subfolder directories that look something like:

```
_results\lp64_intel64_intel3_lib_parallel
```

The executable and result contents of each of the subfolder paths `_results\lp64_intel64_intel3_lib_parallel` might respectively 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
```
The files with the suffix .res are the output results. A partial listing for results file called dm_complex_2d_double_ex1.res might look 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) = {19,12}  
DFTI_FORWARD_SCALE = 1.0  
DFTI_BACKWARD_SCALE = 1.0/(m*n)

INPUT Global vector X, n columns

<table>
<thead>
<tr>
<th>Row</th>
<th>Vector X (n columns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( 1.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>1</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>2</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>3</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>4</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>5</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>6</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>7</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>8</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>9</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
<tr>
<td>10</td>
<td>( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)</td>
</tr>
</tbody>
</table>
NOTE: The output results that you obtain will be a function of your cluster configuration. At the top of the report that four MPI processes were used, the information is consistent with the MPIRUNOPTS="-n 4 ... " macro that was referenced through the nmake command.

Recall the itcpin discussion in Section 7.3 Using itcpin to Instrument an Application, where itcpin is the instrumentation tool used to insert Intel Trace Collector calls into the executables. Using itcpin technology, the following sequence of shell commands could be used to create instrumented executables and generate result information for the executables located in _results\lp64_intel64_intel3_lib_parallel.

For the C language version of the Cluster Discrete Fourier Transform, the DOS Shell commands might look something like:
<table>
<thead>
<tr>
<th>Intel® Processor Architecture</th>
<th>Command-line Sequence for Microsoft Windows</th>
<th>Trace Results are Located In</th>
<th>Execution Results are Located In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® 64</td>
<td>set VT_LOGFILE_PREFIX=cdftc_inst</td>
<td>%CD%\cdftc_inst</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rmdir /S /Q %VT_LOGFILE_PREFIX%</td>
<td>%CD_results\lp64_intel64_intel3_lib_parallel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mkdir %VT_LOGFILE_PREFIX%</td>
<td>%CD_results\lp64_intel64_intel3_lib_parallel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:N ON&quot; -env VT_STATE &quot;:.dll*: off&quot; itcpin --run --profile -- %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.exe&quot; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk!\examples\cdftc\data\dm_complex_2d_double_ex1.dat&quot; &gt; %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:N ON&quot; -env VT_STATE &quot;:.dll*: off&quot; itcpin --run --profile -- %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex2.exe&quot; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk!\examples\cdftc\data\dm_complex_2d_double_ex2.dat&quot; &gt; %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex2.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:N ON&quot; -env VT_STATE &quot;:.dll*: off&quot; itcpin --run --profile -- %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex1.exe&quot; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk!\examples\cdftc\data\dm_complex_2d_single_ex1.dat&quot; &gt; %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex1.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:N ON&quot; -env VT_STATE &quot;:.dll*: off&quot; itcpin --run --profile -- %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex2.exe&quot; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk!\examples\cdftc\data\dm_complex_2d_single_ex2.dat&quot; &gt; %CD_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex2.res&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The "Composer XE 2011 SP1" folder path component needs to be replaced with the appropriate version of Intel® Composer XE that is associated with the Intel® Cluster Studio XE installation on your system.
In this table, the four executables are supplemented with instrumentation calls to the Intel Trace Collector. These DOS commands could be copied from the table above and pasted into a .bat file.

The DOS environment variable %CD% might be set to something like "z:\MPI_Share_Area\cdftc_test". The setting of %CD% will be a function of where you conduct the instrumentation experiments above on your cluster system.

From this table, an mpiexec command in conjunction with itcpin might look something like:

```bash
mpiexec -mapall -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" -env VT_STATE "*.dll*: off" itcpin --run --profile -- "%CD%\_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.exe" "C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mkl\examples\cdftc\data\dm_complex_2d_double_ex1.dat" > "%CD%\_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.res"
```

**NOTE:** The DOS command-line above is a single line of text. The -mapall option for the mpiexec command will map all of the current network drives. This mapping will be removed when the MPI processes exit. This mpiexec option is used to prevent "pin" errors that look something like the following:

```
pin error: System error 0x3 : "Z:\MPI_Share_Area\cdftc_test\_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.exe" : The system cannot find the path specified.
E:pin is exiting due to fatal error
```

As a review from the earlier section on itcpin technology, recall that the executable that is being instrumented for this DOS command is dm_complex_2d_double_ex1.exe. The environment variables that are being set for the mpiexec command are:

```bash
-env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" -env VT_PROCESS "*.dll*: off"
```

As mentioned previously, an explanation for these instrumentation environment variables can be found in the Intel Trace Collector Users’ Guide under the search topic “ITC Configuration”.

In continuing the itcpin review, the itcpin component as part of the overall mpiexec command-line for a C language version of the Cluster Discrete Fourier Transform test case is:

```bash
itcpin --run --profile -- "%CD%\_results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.exe" "C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mkl\examples\cdftc\data\dm_complex_2d_double_ex1.dat"
```
The data input file for executable `dm_complex_2d_double_ex1.exe` is:

"C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mkl\examples\cdftc\data\dm_complex_2d_double_ex1.dat"

The actual reference to the Intel Math Kernel Library folder path for the data input file on your cluster system will be dependent on version of Intel® Composer XE that is installed on your system.

In general, recall that the `itcpin` command-line component has the syntax:

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

where -- is a delimiter between Intel Trace Collector (ITC) options above and the application command-line. The Intel Trace Collector options for the actual `itcpin` example invocation are:

```
    --run --profile
```

The switch called --run instructs `itcpin` to run the application executable. The --profile option is follows. The default instrumentation library is VT.lib which is for the Intel Trace Collector. Also, remember that you can find out additional information about `itcpin` in the Intel Trace Collector User’s Guide under the search topic `itcpin`.

With regards to the test area referenced by the folder path `c:\MPI_Share_Area\cdftf_test`, the Fortran language version of Cluster Discrete Fourier Transform could be instrumented with `itcpin` as follows:
<table>
<thead>
<tr>
<th>Intel® Processor Architecture</th>
<th>Command-line Sequence for Microsoft Windows</th>
<th>Trace Results are Located In</th>
<th>Execution Results are Located In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® 64</td>
<td>set VT_LOGFILE_PREFIX=cdfitf_inst</td>
<td>%CD%\cdftf_inst</td>
<td>%CD%_results\lp64_intel64_intel3_lib_parallel</td>
</tr>
<tr>
<td></td>
<td>rmdir /S /Q %VT_LOGFILE_PREFIX%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mkdir %VT_LOGFILE_PREFIX%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_MPI_DLL &quot;%VT_MPI_DLL%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:1:0:0:0:0:0:0&quot; -env VT_STATE &quot;<em>:</em>.dll*:*.off&quot; itcpin --run --profile -- &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.exe&quot; &lt; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk\examples\cdftc\data\dm_complex_2d_double_ex1.dat&quot; &gt; &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex1.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_MPI_DLL &quot;%VT_MPI_DLL%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:1:0:0:0:0:0:0&quot; -env VT_STATE &quot;<em>:</em>.dll*:*.off&quot; itcpin --run --profile -- &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex2.exe&quot; &lt; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk\examples\cdftc\data\dm_complex_2d_double_ex2.dat&quot; &gt; &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_double_ex2.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_MPI_DLL &quot;%VT_MPI_DLL%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:1:0:0:0:0:0:0&quot; -env VT_STATE &quot;<em>:</em>.dll*:*.off&quot; itcpin --run --profile -- &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex1.exe&quot; &lt; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk\examples\cdftc\data\dm_complex_2d_single_ex1.dat&quot; &gt; &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex1.res&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mpiexec -mapall -n 4 -env VT_DLL_DIR &quot;%VT_DLL_DIR%&quot; -env VT_MPI_DLL &quot;%VT_MPI_DLL%&quot; -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX &quot;%VT_LOGFILE_PREFIX%&quot; -env VT_PROCESS &quot;0:1:0:0:0:0:0:0&quot; -env VT_STATE &quot;<em>:</em>.dll*:*.off&quot; itcpin --run --profile -- &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex2.exe&quot; &lt; &quot;C:\Program Files (x86)\Intel\Composer XE 2011 SP1\mk\examples\cdftc\data\dm_complex_2d_single_ex2.dat&quot; &gt; &quot;%CD\results\lp64_intel64_intel3_lib_parallel\dm_complex_2d_single_ex2.res&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** An input data file for each of the Fortran executables is preceded with the "less than" symbol "<". This is because the Fortran executables read the input data from standard input. As was mentioned previously for the C version of the Cluster Discrete Fourier Transform examples, the folder path "Composer XE 2011 SP1" needs to be replaced with the appropriate
name that is associated with the Intel® Composer XE installation on your system.

The DOS commands above could be copied and pasted into a .bat file within a test area such as z:\MPI_Share_Area\cdftf_test. In regards to the test areas z:\MPI_Share_Area\cdftc_test and z:\MPI_Share_Area\cdftf_test, the tracing data for the executables should be deposited respectively into the folders cdftc_inst and cdftf_inst. Note that in the two tables above, the setting of the environment variable VT_LOGFILE_PREFIX resulted 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.stf
cdftc_inst\dm_complex_2d_double_ex2.stf
cdftc_inst\dm_complex_2d_single_ex1.stf
cdftc_inst\dm_complex_2d_single_ex2.stf
```

and

```
cdftf_inst\dm_complex_2d_double_ex1.stf
cdftf_inst\dm_complex_2d_double_ex2.stf
cdftf_inst\dm_complex_2d_single_ex1.stf
cdftf_inst\dm_complex_2d_single_ex2.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.stf
```

Figure 8.2 shows the result of simultaneously displaying the Function Profile Chart and the Event Timeline Chart.
Figure 8.2 – The Event Timeline Chart and the Function Profile Chart for a Cluster Discrete Fourier Transform Example

8.3 Experimenting with the High Performance Linpack Benchmark*

On Microsoft Windows CCS, 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:
If you make a scratch directory, say:

```
test_mp_linpack
```

on a file share for your cluster, and copy the contents of `<directory-path-to-mkl>/benchmarks/mp_linpack` 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 `nmake` command:

```
nmake arch=intel64 HOME="%CD%" LAdir="c:\Program Files (x86)\Intel\Composer XE 2011 SP1\mkl" LAinc="c:\Program Files (x86)\Intel\Composer XE 2011 SP1\mkl\include" MPIdir="%I_MPI_ROOT%" install
```

where the command sequence above is one continuous line. The macro variable `HOME` references the work folder where the `nmake` command was invoked. In this situation the working directory is:

```
...\test_mp_linpack
```

The macros `LAdir` and `LAinc` describe the folder path to the Intel® 64 Math Kernel library and the Intel® MKL include folder, respectively. The partial directory path `c:\Program Files (x86)\Intel\Composer XE 2011 SP1` for the macros `LAdir` and `LAinc` should be considered an example of where an Intel® Math Kernel Library might reside.
The High Performance Linpack executable for the nmake command above will be placed into ...\test_mp_linpack\bin\intel64 and will be called xhpl. The table below summarizes makefile and associated mpiexec commands that might be used to create xhpl executable for Intel® 64 architectures, respectively. The mpiexec commands use four MPI processes to do the domain decomposition.

<table>
<thead>
<tr>
<th>Intel Processor Architecture</th>
<th>Command-line Sequence for Microsoft Windows</th>
<th>Executable is Located In</th>
<th>Execution Results are Located In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® 64</td>
<td>nmake arch=intel64 HOME=&quot;%CD%&quot; LAdir=&quot;c:\Program Files (x86)\Intel\Composer XE 2011 SP1&quot; LAinc=&quot;c:\Program Files (x86)\Intel\Composer XE 2011 SP\mkl\include&quot; mpidir=&quot;%I_MPI_ROOT%&quot; install cd %CD%\bin\intel64 mpiexec -mapall -n 4 .\xhpl.exe &gt; results.intel64.out</td>
<td>%CD%\bin \intel64</td>
<td>%CD%\bin\intel64</td>
</tr>
</tbody>
</table>

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 : 1ring
DEPTH : 0
SWAP  : Mix (threshold = 256)
L1    : no-transposed form
U     : no-transposed form
EQUIL : no
ALIGN : 8 double precision words
```

================================================================================
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 : 1ring
DEPTH : 0
SWAP  : Mix (threshold = 256)
L1    : no-transposed form
U     : no-transposed form
EQUIL : no
ALIGN : 8 double precision words
================================================================================
The file `<directory-path-to-mkl-documentation>`\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](http://premier.intel.com).

[Back to Table of Contents]
9. Using the Intel® MPI Benchmarks

The Intel MPI Benchmarks have been ported to Microsoft Windows OS. The folder 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, 2008, and 2010 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.

Within Microsoft Windows Explorer and starting at the Windows folder, you can go to one of the subfolders IMB-EXT_VS_2005, IMB-EXT_VS_2008, IMB-EXT_VS_2010, IMB-IO_VS_2005, IMB-IO_VS_2008, IMB-IO_VS_2010, IMB-MPI1_VS_2005, IMB-MPI1_VS_2008, or IMB-MPI1_VS_2010 and click on the corresponding ".vcproj" file and open up Microsoft Visual Studio* (Figure 9.1).

The three executables that will be created from the respective Visual Studio 2005, Visual Studio 2008, or Visual Studio 2010 projects will be:

    IMB-EXT.exe
    IMB-IO.exe
    IMB-MPI1.exe

In Figure 9.1 Microsoft Visual Studio* 2008 is being used.
Figure 9.1 – Illustration for Starting Microsoft Visual Studio* 2008 on the project file IMB-EXT.vcproj

9.1 Building Microsoft Visual Studio* x64 Executables for the Intel® MPI Benchmarks

From the Visual Studio Project panel:

1) Change the "Solution Platforms" dialog box to "x64". See Figure 9.2.
2) Change the "Solution Configurations" dialog box to "Release". See Figure 9.2.
3) Follow the menu path Project->Properties or Alt+F7 and check to make sure that the following are set by expanding "Configuration Properties":

a) General->Project Defaults - Change "Character Set" to "Use Multi-Byte Character Set"

b) Debugging
   i) Set the "Debugger to launch" value to "Local Windows Debugger", for example. Note that "Local Windows Debugger" is one possible setting. See Figure 9.3.
   ii) For the row "Command" add "$(I_MPI_ROOT)\em64t\bin\mpiexec.exe". Note that for the mpiexec.exe folder path, quotes are optional with Visual Studio* 2005 and 2008, but should NOT be included in the case of Visual Studio* 2010.
   iii) For the row "Command Arguments" add "-n 2 $(TargetPath)" where $(TargetPath) should be quoted as in:

   
   -n 2 "$\{TargetPath\}"

   $\{TargetPath\}

   c) C/C++->General
   i) For the row "Additional Include Directories", add "$\{I_MPI_ROOT\}\em64t\include".

Figure 9.2 – The Solution Configuration is set to “Release” and Solution Platforms is set to “x64”. Also note that IMB-EXT is highlighted in the Solution Explorer panel on the left in preparation for the context sensitive operations outlined in step 3
ii) For the row "Warning Level", set the warning level to "Level 1 (/W1)".

d) C/C++->Preprocessor

i) For the row "Preprocessor definitions" within the Visual Studio projects IMB-EXT_VS_2005, IMB-EXT_VS_2008, or IMB-EXT_VS_2010, add the conditional compilation macro references to WIN_IMB, _CRT_SECURE_NO_DEPRECATE, EXT

ii) For the row "Preprocessor definitions" within the Visual Studio projects IMB-IO_VS_2005, IMB-IO_VS_2008, or IMB-IO_VS_2010, add the conditional compilation macro references to WIN_IMB, _CRT_SECURE_NO_DEPRECATE, MPIIO

iii) For the row "Preprocessor definitions" within the Visual Studio projects, IMB-MPI1_VS_2005, IMB-MPI1_VS_2008, or IMB-MPI1_VS_2010, add the conditional compilation macro references to WIN_IMB, _CRT_SECURE_NO_DEPRECATE, MPI1

e) Linker->Input

i) For the row "Additional Dependencies" add "$(I_MPI_ROOT)\em64t\lib\impi.lib". Be sure to include the quotes. If items "a" through "e" are already set, then proceed to step 4.

Figure 9.3 – Setting the Command and Command Arguments for Debugging under Configuration Properties

4) Use F7 or Build->Build Solution to create an executable. See Figure 9.4.
Figure 9.4 – Microsoft* Visual Studio* 2008 Illustration for building a solution for IMB-EXT

5) Use Debug->Start Without Debugging or Ctrl+F5 to run the executable. See Figure 9.5.
The steps outlined above can be applied to the 2005, and/or 2008, and/or 2010 Microsoft* Visual Studio* project folders in building executables for IMB-MPI1.exe and IMB-IO.exe.

**9.2 Building Microsoft Visual Studio* IA-32 Executables for the Intel® MPI Benchmarks**

Before opening a Microsoft* Visual Studio* project folder, you may want to check the environment variable settings for Include, Lib, and Path. To do this for Microsoft* Windows* HPC Server 2008 OS, or Microsoft Windows* HPC Server 2008 R2 OS, start at the Start menu and select Start->Control Panel->System and Maintenance->System->Change settings and a System Properties panel will appear where you should click on the **Advanced** tab (Figure 9.6).
Regarding Figure 9.6, click on the **Environment Variables...** button and see if the System Environment Variables **Include**, **Lib**, and **Path** need editing for the display panel shown in Figure 9.7.
For example, if the Include, Lib, and Path environment variables have respectively the settings:

```plaintext
%I_MPI_ROOT%\em64t\include
%I_MPI_ROOT%\em64t\lib
%I_MPI_ROOT%\em64t\bin
```

at the beginning of the Variable value dialog panel as shown in Figure 9.7, the these paths should be changed from the subfolder reference of em64t to that of ia32:

```plaintext
%I_MPI_ROOT%\ia32\include
%I_MPI_ROOT%\ia32\lib
%I_MPI_ROOT%\ia32\bin
```

After making the appropriate changes, the OK button should be clicked regarding the Environment Variables panel shown in Figure 9.7.
After checking out the settings of these environment variables and saving any necessary changes, one can proceed to open the relevant Visual Studio 2005, Visual Studio 2008, or Visual Studio 2010 projects under the WINDOWS subfolder for the Intel® MPI Benchmarks.

From the Microsoft Visual Studio* Project panel for the Visual Studio* 2008 project IMB-MPI1:

1) Change the "Solution Platforms" dialog box to "ia32". See Figure 9.8.
2) Change the "Solution Configurations" dialog box to "Release". See Figure 9.8.

![Figure 9.8 – The Solution Configuration is set to “Release” and Solution Platforms is set to “ia32”. Also note that IMB-MPI1 is highlighted in the Solution Explorer panel on the left in preparation for the context sensitive operations outlined in step 3](image)

3) Follow the menu path Project->Properties or Alt+F7 and check to make sure that the following are set by expanding "Configuration Properties":
   a) General->Project Defaults - Change "Character Set" to "Use Multi-Byte Character Set"
b) Debugging
   i) Set the "Debugger to launch" value to "Local Windows Debugger", for example. Note that "Local Windows Debugger" is one possible setting. See Figure 9.9.
   ii) For the row "Command" add "$(I_MPI_ROOT)\ia32\bin\mpiexec.exe". Note that for the mpiexec.exe folder path, quotes are optional with Visual Studio* 2005 and 2008, but should NOT be included in the case of Visual Studio* 2010.
   iii) For the row "Command Arguments" add "-n 2 $(TargetPath)" where

   \[ -n 2 "$(TargetPath)" \]

   $(TargetPath) should be quoted as in:

   \[-n 2 "$\{(TargetPath)\}" \]

c) C/C++->General
   i) For the row "Additional Include Directories", add

   \"$(I_MPI_ROOT)\ia32\include\".

   ii) For the row "Warning Level", set the warning level to "Level 1 (/W1)".

d) C/C++->Preprocessor
   i) For the row "Preprocessor definitions" within the Visual Studio projects IMB-EXT_VS_2005, IMB-EXT_VS_2008, and IMB-EXT_VS_2010, add the conditional compilation macro references to WIN_IMB,

   \_CRT_SECURE_NO_DEPRECATE, EXT

   ii) For the row "Preprocessor definitions" within the Visual Studio projects IMB-IO_VS_2005, IMB-IO_VS_2008, and IMB-IO_VS_2010, add the conditional compilation macro references to WIN_IMB,

   \_CRT_SECURE_NO_DEPRECATE, MPIIO

   iii) For the row "Preprocessor definitions" within the Visual Studio projects IMB-MPI1_VS_2005, IMB-MPI1_VS_2008, or IMB-MPI1_VS_2010, add the conditional compilation macro references to WIN_IMB,

   \_CRT_SECURE_NO_DEPRECATE, MPI1

e) Linker->Input
   i) For the row "Additional Dependencies" add

   \"$(I_MPI_ROOT)\ia32\lib\impi.lib\". Be sure to include the quotes.

If items "a" through "e" are already set, then proceed to step 4.
Figure 9.9 – Setting the Command and Command Arguments for Debugging under Configuration Properties

4) Use F7 or Build->Build Solution to create an executable. See Figure 9.10.
Figure 9.10 – Microsoft* Visual Studio* 2008 Illustration for building a solution for IMB–MPI1

5) Use Debug->Start Without Debugging or Ctrl+F5 to run the executable. See Figure 9.11.
Figure 9.11 – Generation of the command-line panel using the keys Ctrl+F5. The command-line panel shows the execution results for IMB-MPI1.exe within Microsoft Visual Studio* 2008

The steps outlined above can be applied to the 2005, 2008, and/or 2010 Microsoft* Visual Studio* project folders in building executables for IMB-EXT.exe and IMB.IO.exe.

Back to Table of Contents
10. Uninstalling the Intel® Cluster Studio XE on Microsoft* Windows* HPC Server 2008 OS

For Microsoft® Windows® HPC Server 2008 OS, if you want to uninstall the Intel Cluster Studio XE, follow the menu path **Start->Control Panel**.

![Control Panel Display for Microsoft® Windows® HPC Server 2008 OS](image)

**Figure 10.1** – Control Panel Display for Microsoft® Windows® HPC Server 2008 OS

Click on the **Programs Uninstall a program** icon to launch a **Programs and Features** display panel (Figure 10.2). For this display panel, find the appropriate version of the Intel Cluster Studio XE that you want to remove, highlight it.
Figure 10.2 shows the row for the Intel® Cluster Studio XE application as being highlighted.

If you click on the **Uninstall/Change** button that is shown in the tool bar of the **Program and Features** screen (see Figure 10.2 above), the software components of the Intel® Cluster Studio XE will be removed from all of the nodes of the cluster. Clicking on the **Uninstall/Change** button produces the following dialog panel (Figure 10.3):
Make sure that the **Remove** radial button is checked as shown in Figure 10.3. Once this is done, the **Next** button can be toggled. This will invoke the uninstallation process (Figure 10.4).
Figure 10.4 – Options display for the uninstall process

In Figure 10.4, click on the **Remove** button at the bottom of the display panel. You will see the progress panel shown in Figure 10.5.
Finally, when all of the components have been uninstalled, the following panel will appear (Figure 10.6):
Figure 10.6 – “Removal Complete” Status Panel for Intel® Cluster Studio XE

The Finish button should be clicked on, and this will update the Programs and Features panel as initially shown in Figure 10.2.
11. **Hardware Recommendations for Installation on Microsoft® Windows® CCS 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 Microsoft® Windows® OS**

<table>
<thead>
<tr>
<th>OS Distributions</th>
<th>IA-32 Architecture</th>
<th>Intel® 64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft® Windows® HPC Server 2008</td>
<td>N/A</td>
<td>S</td>
</tr>
<tr>
<td>Microsoft® Windows® HPC Server 2008 R2</td>
<td>N/A</td>
<td>S</td>
</tr>
</tbody>
</table>

S = Supported

**Memory Requirements**
2 GB of RAM (minimum)

[Back to Table of Contents]
12. **System Administrator Checklist for Microsoft Windows CCS* 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 folder path might be:

C:\Program Files (x86)\intel\licenses

[Back to Table of Contents]
13. **User Checklist for Microsoft Windows CCS* OS**

To create IA-32 executables within a DOS command-line session, type the command:

```
ictvars.bat ia32
```

For the given DOS login session, to restore the DOS command-line session default of creating executables with Intel® 64 address extensions, type:

```
ictvars.bat
```

[Back to Table of Contents]
14. Using the Compiler Switch /Qtcollect

The Intel® C++ and Intel® Fortran Compilers on Microsoft Windows have the command-line switch called /Qtcollect 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.

<table>
<thead>
<tr>
<th>Library Selection</th>
<th>Meaning</th>
<th>How to Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT.lib</td>
<td>Default library</td>
<td>/Qtcollect</td>
</tr>
<tr>
<td>VTcs.lib</td>
<td>Client-server trace collection library</td>
<td>/Qtcollect=VTcs</td>
</tr>
<tr>
<td>VTfs.lib</td>
<td>Fail-safe trace collection library</td>
<td>/Qtcollect=VTfs</td>
</tr>
</tbody>
</table>

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 /Qtcollect compiler option for the Intel compilers, one might use the following compilation and link commands:

icl /Fetestc_Qtcollect /Qtcollect /I"%I_MPI_ROOT%"\em64t\include test.c
/link /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib
/NODEFAULTLIB:LIBCMTD.lib

icl /Fetestcpp_Qtcollect /Qtcollect /I"%I_MPI_ROOT%"\em64t\include test.cpp /link /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impicxx.lib
/NODEFAULTLIB:LIBCMTD.lib

ifort /Fetestf_Qtcollect /Qtcollect /I"%I_MPI_ROOT%"\em64t\include test.f /link /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib
/NODEFAULTLIB:LIBCMTD.lib

ifort /Fetestf90_Qtcollect /Qtcollect /I"%I_MPI_ROOT%"\em64t\include test.f90 /link /LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib
/NODEFAULTLIB:LIBCMTD.lib

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

```
testc_Qtcollect.exe
testcpp_Qtcollect.exe
testf_Qtcollect.exe
testf90_Qtcollect.exe
```

So as to make a comparison with the Intel Trace Collector STF files:

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

within the directory test_inst, we will use the following mpiexec commands:
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf90_Qtcollect

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_Qtcollect.stf the following traceanalyzer command can be launched from a DOS 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 DOS window.

![Base panel for the Intel Trace Analyzer](image)

**Figure 14.1 – Base panel for the Intel Trace Analyzer when invoking the DOS 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 panel](image)
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).
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_Qtcollect.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, notice that the timeline display for testcpp_Qtcollect.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_Qtcollect.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:
Highlight the **All Functions** tree entry and press the **Apply** button in the lower right corner of this panel. Then press 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_Qtcollect.stf

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

1) testc.stf and testc_Qtcollect.stf
2) testf.stf and testf_Qtcollect.stf
3) testf90.stf and testf90_Qtcollect.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 /Qcoarray compiler option to enable use of co-array syntax. The possible configurations for using the /Qcoarray compilation options are:

/Qcoarray | /Qcoarray=shared

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 /Qcoarray and /Qcoarray=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. 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 if 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 /Qcoarray-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 Windows click on the following Co-array Fortran path where Intel® Cluster Studio XE is installed. This path points to a Zip package, that has a Microsoft* Visual Studio* solution. Copy this Zip package to a scratch directory and extract the Zip package contents into the scratch folder. After completing the extraction 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

You can open the associated Microsoft* Visual Studio* solution, where you can compile, and execute the solution in the Microsoft* Visual Studio* environment. Alternatively, within a DOS panel, to build an executable for hello_image.f90, you can simply type an ifort command that looks something like:

ifort /Qcoarray hello_image.f90
The executable that is created from the command above is called `hello_image.exe` and it can be executed by typing the command:

```bash
.\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]
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.

```c
#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"
             , rank, size, name);

    MPI_Finalize();
}
```
// 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
", 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

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 C:\Program Files (x86)\Intel\VTune Amplifier XE 2011.

To use Intel® VTune™ Amplifier XE, there are 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.

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

Within a DOS panel, type the command:

```
amplxe-cl -help
```

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.

```c
#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], d[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 DOS command-line that uses Intel® VTune™ Amplifier XE might look something like:

```
mpiexec -mapall -n 4 amplxe-cl -r amplifierxe_results -collect hotspots -- .\cean.exe
```

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

One method of launching the graphical user interface for Intel® VTune™ Amplifier XE is through the DOS 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 C:\Program Files (x86)\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.

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

Within a DOS panel, type the command:

```
inspxe-cl -help
```

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.

```c
#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);
}
18.3 How do I Run and Collect Memory Leak Information within an Intel® MPI Library Application?

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

```
mpiexec -mapall -n 4 inspxe-cl -r inspectorxe_results -collect mi1 -- .\cean.exe
```

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 DOS 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
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:

```c
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],d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1],
                d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
}
```

will be modified to:

```c
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],d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1],
                d[i*MAX_ARRAY_SIZE+MAX_ARRAY_SIZE-1]);
    free(d);  // Added call to free function
}
```
// 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 -mapall -n 4 inspxe-cl -r inspectorxe_results2 -collect mi1 --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 2011 is an advanced threading tool for Microsoft Visual Studio* C/C++ developers who are planning to add threading semantics to serial or parallel software applications.

Within the Intel® Cluster Studio XE 2012 development environment, Intel® Parallel Advisor is only installed on the master node of the Microsoft* Windows* CCS OS computing cluster.

The Intel® Parallel Advisor tool is designed strictly for a C/C++ threaded-only programming application, and is not intended for an MPI or hybrid parallel programming application.

19.1 How do I Access Intel® Parallel Advisor?
You can access Intel® Parallel Advisor through Microsoft* Visual Studio*.

19.2 How do I bring up the Intel® Parallel Advisor Workflow within Microsoft* Visual Studio*?
Within Microsoft* Visual Studio*, follow the menu path Tools->Intel Parallel Advisor 2011->Open Advisor Workflow.
Figure 19.1 – Bring up the Intel® Parallel Advisor Plug-in within Microsoft® Visual Studio®

Figure 19.2 shows a sequence of Parallel Advisor workflow steps that can be used to restructure your programming application to utilize parallel computation.
Figure 19.2 – The Intel® Parallel Advisor Workflow Panel

The buttons referenced in Figure 19.2 have the following meanings:

1. **Survey Target**
   - *Where* should I consider adding parallelism? Locate the loops and functions where your program needs parallelism.
   - Collect Survey Data
   - View Survey Result

2. **Annotate Sources**
   - Add Advisor annotations to identify possible parallel tasks and their enclosing parallel sites.
   - Steps to annotate
   - View Annotations

3. **Check Suitability**
   - Analyze the annotated program to check its predicted parallel performance.
   - Collect Suitability Data
   - View Suitability Result

4. **Check Correctness**
   - Predict parallel data sharing problems for the annotated tasks. Fix the reported sharing problems.
   - Collect Correctness Data
   - View Correctness Result

5. **Add Parallel Framework**
   - Steps to replace automated annotations
   - View Annotations

Current Project: IMB-MPI11
1. You identify code regions in your program application as possible places to add parallelism. These code regions are usually places where your program application spends most of its execution time. To help you select the proposed application segments where you might add parallelism, use the Advisor Survey (profile) tool.

2. Through the Visual Studio coding editor, you modify your source code with one-line Advisor Annotations that mark the possible parallel places in your program. These annotations are recognized by the Advisor Suitability and Correctness tools, which predict your serial program's parallel behavior.

3. You run the Suitability tool to analyze your running program to predict the approximate utilization for the selected parallel programming segments. This utilization is based on available cores. If the Suitability tool shows that one of the selected parallel segments does not improve performance, then modify or remove the corresponding annotations.

4. You run the Correctness tool to analyze your running program to predict and locate data sharing problems that can occur as a result of adding parallelism. An example of correctness problems might be data race conditions. The Correctness tool helps you examine the sources to locate the cause of the data sharing problems. After you modify your program to fix the data sharing problems, continue to rerun the Correctness tool until you eliminate the potential data sharing problems. After you fix the remaining sharing problems, run the Suitability tool again.

5. After you test your modified serial program, add code that actually introduces parallelism. You do this by selecting a parallel framework such as Intel® Cilk™ Plus or Intel® Threading Building Blocks (Intel® TBB). After you have added parallel programming semantics, test your program again.

The Intel® Parallel Advisor 2011 Getting Started Tutorial is available on the web at the URL:


Back to Table of Contents