# Intel® Cluster Toolkit 2.0 Tutorial

(Revision 20051122)

## Table of Contents

1. Disclaimer and Legal Information ................................................................. 2
2. Introduction ........................................................................................................ 3
3. Synopsis of New Features .................................................................................. 4
4. Acronyms and Definitions ................................................................................. 6
5. Conventions ....................................................................................................... 7
6. Intel Software Downloads and Installation for the Intel® Cluster Toolkit .............. 8
   6.1 Installation of Intel® Cluster Toolkit on 32 Bit and 64 Bit Nodes which Share a Common /opt Directory ................................................................. 13
7. The Parallel Install Capability for the Intel® Cluster Toolkit Installer .................. 14
8. The Software Architecture of the Intel® Cluster Toolkit ................................... 15
9. Getting Started with Intel® MPI Library ......................................................... 17
   9.1 Launching MPD Daemons ........................................................................... 18
      9.1.1 How to Set Up MPD Daemons ............................................................. 19
   9.2 The mpdboot Command ............................................................................. 20
   9.3 Compiling and Linking with Intel® MPI Library .......................................... 20
   9.4 Selecting a Network Fabric or Device ....................................................... 20
   9.5 Running an MPI Program Using Intel® MPI Library ..................................... 21
   9.6 Experimenting with Intel® MPI Library ..................................................... 21
   9.7 Controlling MPI Process Placement ......................................................... 22
10. Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer ................................................................. 24
   10.1 Compiling MPI Programs in Conjunction with the Intel® Trace Collector Header Files ................................................................. 26
   10.2 Linking MPI Programs with the Intel® Trace Collector Libraries .................. 27
      10.2.1 Simplifying the Compile and Link Command between Intel® MPI Library and the Intel® Trace Collector ......................................................... 46
   10.2.2 Internal Intel® MPI Library State Tracing with Intel® Trace Collector ........ 47
   10.2.3 How to Get Additional Help Regarding the Intel® Trace Collector ............. 51
   10.3 Experimenting with the MPI Examples in the Directory ${VT_ROOT}/examples .... 51
   10.4 Experimenting with Intel® Trace Analyzer Examples in the Directory <directory-path-to-ITA>/examples ................................................................. 57
      10.4.1 How to Get Additional Help Regarding the Intel® Trace Analyzer .......... 59
   10.5 Experimenting with Intel® Trace Analyzer and Collector in a Fail-Safe Mode .... 59
11. Getting Started in Using the Intel® Cluster Math Kernel Library (Intel® Cluster MKL) .... 61
   11.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK Examples with the Intel® Trace Collector and Intel® Trace Analyzer................................. 65
   11.2 Experimenting with the Cluster DFT Software .......................................... 69
12. Experimenting with the Intel® MPI Benchmarks .............................................. 73
13. Hardware and Recommendations for Installation .......................................... 86
14. System Administrator Checklist ....................................................................... 86
15. User Checklist ................................................................................................... 87
1. Disclaimer and Legal Information

This installation and tutorial document, called Intel® Cluster Toolkit Tutorial, as well as the software described in it, is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

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. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

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 software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

Intel, the Intel logo, Intel SpeedStep, Intel NetBurst, Intel NetStructure, MMX, i386, i486, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Celeron, Intel Centrino, Intel Xeon, Intel XScale, Itanium, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others. All trademarks and registered trademarks referenced in this Intel® Cluster Toolkit document are the property of their respective holders.

2. Introduction

At the time of this writing, the Intel® Cluster Toolkit 2.0 release consists of:

1. Intel® MPI Library 2.0
2. Intel® Cluster Math Kernel Library 8.0.1 (Intel® Cluster MKL 8.0.1), which includes ScaLAPACK and Cluster DFT (Discrete Fourier Transform)
3. Intel® Trace Analyzer\(^1\) and Collector\(^2\)
4. Intel® MPI Benchmarks\(^3\)

This Intel® Cluster Toolkit 2.0 Tutorial contains information for installing software packages to support homogeneous cluster computing for Intel® Pentium® 4, Intel® Xeon®, Intel® Itanium® 2, and Intel® EM64T-based processors running Red Hat Enterprise Linux* 3.0, Red Hat Enterprise Linux 4.0, SUSE Linux Enterprise Server* 9, SUSE Linux 9.0, or SUSE Linux 9.1. The tutorial was last checked and validated on November 22, 2005. The emphasis of this tutorial is on the interoperability of the software components listed above.

The user of the Intel® Cluster Toolkit may need assistance from their system administrator in installing the associated software packages on their cluster system, if the installation directory requires system administrative write privileges (e.g. /opt/intel). This assumes that the user’s login account does not have administrative privileges.

---

\(^1\) Intel® Trace Analyzer was formerly known as Vampir.

\(^2\) Intel® Trace Collector was formerly known as Vampirtrace.

\(^3\) Intel® MPI Benchmarks was formerly known as the Pallas MPI Benchmarks (PMB).
3. Synopsis of New Features

Intel® Cluster Toolkit 2.0
- There is additional operating system support
- The Intel® Cluster Toolkit has an `expect` script called `sshconnectivity.exp` which the user may find useful in establishing secure shell connectivity on a cluster system. Please see chapter 6 titled “Intel Software Downloads and Installation for the Intel® Cluster Toolkit”

Intel® Cluster MKL 8.0.1
- Cluster DFTs (Discrete Fourier Transforms)
  - There are examples of using the Cluster Fast Fourier Transform within the chapter on Intel® Cluster MKL
- Iterative sparse solver
- Increased sparse solver support
  - Sparse level 2 and 3 BLAS
- Interval Arithmetic
  - Solvers for interval systems of linear equations
  - Interval matrix inversion
  - Functions for testing regularity/singularity of interval matrices
- Vector Statistical Library additions
  - Convolution and correlation
  - New random number generators and service functions
- FFTW (“Fastest Fourier Transform in the West”) interface to the MKL DFTs
- Fortran 90/95 interfaces to LAPACK subroutines

Intel® MPI Library 2.0
- Updated code base
  - An increased level of MPI support from version "1.2" to "2.0"
  - Passive target one-sided communication
  - Generalized requests
  - MPI_IN_PLACE parameter in Fortran
  - MPI_Type_create_darray/subarray in Fortran
- Streamlined product setup
  - Installation under root or ordinary user id
  - `mpivars.sh` and `mpivars.csh` scripts for easy path setting
- Simplified process management
  - `mpiexec -perhost` and `nolocal` options
  - `mpirun` script that automates MPD startup and cleanup
  - System-, user-, and session-specific configuration files
  - Transparent support for alternative IP interfaces
- Environment variables for runtime control over
  - Process pinning
  - Optimized collective operations
  - Device-specific protocol thresholds
  - Collective algorithm thresholds
  - Enhanced memory registration cache
  - Platform-specific fine grain timer
- Increased interoperability
  - Support for DAPL* v1.1 and DAPL* v1.2 compliant providers
  - Message queue browsing with the TotalView* and DDT* debuggers
  - Internal MPI library state tracing with Intel® Trace Collector
- Enhanced support for operating systems and compilers
  - Red Hat Enterprise Linux* 4.0
  - SUSE Linux* Enterprise Server 9
  - GNU* compilers, version 4.0
Intel® Compilers for Linux®, version 9.0
• Getting Started and Reference Manual documents

Intel® Trace Analyzer 6.0 (ITA 6.0)
• The Intel® Trace Analyzer has been completely rewritten. Therefore, all of the Intel® Trace Analyzer figures in this document have been regenerated
• Some of the new features of the Intel® Trace Analyzer are:
  o New graphical user interface (GUI)
    • Scalability features are based on time, # processors, and # routines/methods
    • Aligned event timeline displays with quantitative and qualitative views
    • Profiling of functions, point-to-point, collective communications
  o Significant performance improvements
    • Efficient memory handling enables analysis of large traces on desktop
    • Consistent, platform independent intuitive GUI (Qt*)
    • Introduce advanced filter and aggregation mechanisms
  o Broad platform support
    • Linux and Windows*
    • Leverages multi-core platforms based on Intel processor architecture
  o Feature objectives
    • Next–generation event analysis tool
    • Superior replacement for Intel® Trace Analyzer 4.0 (aka Vampir*)

Intel® Trace Collector 6.0 (ITC 6.0)
• Tracing of arbitrary distributed applications (libVTcs)
• Tracing of failing MPI applications (libVTfs)
• Binary instrumentation to do function profiling and MPI tracing in executables which have not been linked against ITC (itcinstrument)
• Tracing of OS activity like disk/net IO and CPU utilization
• Shared versions of the VT libraries which can be used to insert instrumentation without relinking (via LD PRELOAD)
# 4. Acronyms and Definitions

The following are acronyms and definitions of those acronyms that are referenced within the Intel® Cluster Toolkit 2.0 User’s Guide.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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>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 transports data over a variety of electrical or optical media. It transports any of several upper layer protocols via data packet transmissions.</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>ICT</td>
<td>Intel® Cluster Toolkit</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 &quot;virtual machine&quot; 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>
<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>RPM</td>
<td>Red Hat Package Management - a system which 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>SMP</td>
<td>Symmetric Multi-processor</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>
</tbody>
</table>
5. Conventions

In this document, italicized text such as:

<directory-path-to-intel-mpi>

refers to a meta-symbol which is typically user dependent. The user will need to replace such meta-symbols with actual user text. An example of this for the above meta-symbol might be:

/opt/intel/ict/2.0/mpi/2.0

which would be an actual directory path on the user’s computing system for an installation of Intel® MPI Library 2.0. The second form of italicized text will be something like:

hostname

Again, the user will be required to substitute in a phrase that is specific to the user’s computing system, such as:

clusternode1

The last form of italicized notation:

<# of processes>

which is used in the mpiexec command:

mpiexec -n <# of processes> ./mpi_example

requires a user to supply an integer value that is greater than zero. A user-provided value for the shell command above might be:

mpiexec -n 4 ./mpi_example
6. Intel Software Downloads and Installation for the Intel® Cluster Toolkit

The Intel® Cluster Toolkit installation process is comprised of seven basic steps. The Intel® Cluster Toolkit 2.0 package consists of the following components:

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Default Installation Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Cluster MKL 8.0.1</td>
<td>/opt/intel/ict/2.0/cmk1/8.0.1</td>
</tr>
<tr>
<td>Intel® MPI Library 2.0</td>
<td>/opt/intel/ict/2.0/mpi/2.0</td>
</tr>
<tr>
<td>Intel® MPI Benchmarks 2.3</td>
<td>/opt/intel/ict/2.0/imb/2.3</td>
</tr>
<tr>
<td>Intel® Trace Analyzer 6.0</td>
<td>/opt/intel/ict/2.0/ita/6.0</td>
</tr>
<tr>
<td>Intel® Trace Collector 6.0</td>
<td>/opt/intel/ict/2.0/itc/6.0</td>
</tr>
</tbody>
</table>

Note that the Intel® Cluster Toolkit 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. The user does not have to worry about selecting the correct software component names for the given Intel architecture.

Recall that the user of the Intel® Cluster Toolkit may need assistance from their system administrator in installing the associated software packages on their cluster system, if the installation directory requires system administrative write privileges (e.g. /opt/intel). This assumes that the user’s login account does not have administrative privileges.

1. The 4.2.2 version of RPM on Red Hat Enterprise Linux* 3.0 for Itanium® 2 has a broken relocation feature. This will be a serious problem for users trying to do installs on clusters where there are shared devices. A recommended solution is for the user to upgrade to the latest release of RPM. A possible URL for retrieving a recent release of RPM that resolves this problem on the Itanium® 2 architecture is:

   http://www.redhat.com

2. A `machines.LINUX` file will either need to be created, or an existing `machines.LINUX` file can be used by the Intel® Cluster Toolkit installer to deploy the appropriate software packages from the toolkit amongst the nodes of the cluster. This `machines.LINUX` file contains a list of the computing nodes (i.e. the hostnames) for the cluster. The format is one hostname per line:

   hostname

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

   clusternode1
   clusternode2
   clusternode3
   clusternode4
   clusternode5
   clusternode6
   clusternode7
   clusternode8
It is always assumed that the first node in the list is the master node. The remaining nodes are the compute nodes. The text `clusternode1` and `clusternode2`, for example, represent the names of two of the nodes in a contrived computing cluster. The contents of the machines.LINUX file can also be used by users to construct an `mpd.hosts` file for the multi-purpose daemon (MPD) protocol. The MPD protocol is used for running MPI applications that utilize Intel® MPI Library.

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

```
/tmp/ict_staging_area
```

where `ict_staging_area` is an acronym for Intel® Cluster Toolkit staging area.

4. Upon registering for Intel® Cluster Toolkit 2.0, you will receive a serial number (E.g., C111-12345678) for this product. Your serial number can be found within the email receipt of your product purchase. Go to the Intel Registration Center site and provide the product serial number information. Once admission has been granted into the registration center, a user will be able to access the Intel® Premier web pages for software support.

5. The license for the Intel® Cluster Toolkit license file that is provided to the user should be placed in a location pointed to by the `INTEL_LICENSE_FILE` environment variable. Do not change the file name as the ".lic" extension is critical. Common locations for the attached license file are:

```
<installation path>/licenses
```

For example, on the cluster system where the Intel® Cluster Toolkit Software is to be installed, all licenses for Intel-based software products might be placed in:

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

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

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

For C Shell, the syntax might be:

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

6. Patrons can place the Intel® Cluster Toolkit software package into the staging area folder.

7. The installer package for the Intel® Cluster Toolkit has the following general nomenclature:

```
l_ict_<version>..<release>..tar.gz
```

where `<version>` is a string such as:

```
b_2.0, where b is an acronym for beta
```

9
or

\[ p_{2.0}, \text{where } p \text{ is an acronym for production} \]

The \(<release>\) meta-symbol is a string such as 018. This string indicates the package number.

The command:

\[
\text{tar } -xvzf \text{l_ict}_<\text{version}>.<release>.tar.gz
\]

will create a subdirectory called \(\text{l_ict}_<\text{version}>.<release>\). Change to that directory with the shell command:

\[
\text{cd } \text{l_ict}_<\text{version}>.<release>
\]

For example, suppose the installation package is called \(\text{l_ict}_p_{2.0.018}.tar.gz\). In the staging area that has been created, type the command:

\[
\text{tar } -xvzf \text{l_ict}_p_{2.0.018}.tar.gz
\]

This will create a subdirectory called \(\text{l_ict}_p_{2.0.018}\). Change to that directory with the shell command:

\[
\text{cd } \text{l_ict}_p_{2.0.018}
\]

In that folder make sure that \(\text{machines.LINUX}\) file, as mentioned in item 2 above, is either in this directory or the user should know the directory path to this file.

The \text{expect} shell script file called "\text{sshconnectivity.exp}\" can be used to help the user establish secure shell connectivity on a cluster system. The syntax for the command is:

\[
./\text{sshconnectivity.exp} \text{machines.LINUX}
\]

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

Enter your cluster password:
Re-enter your cluster password:

Each time the user enters the cluster password, asterisks will appear in lieu of the password text so as to provide security. Upon successful completion of the script, the following message fragment will appear:

\[
\ldots
\text{Node count} = 4
\text{Secure shell connectivity was established on all nodes.}
\ldots
\]

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

\[
/tmp/\text{sshconnectivity.<login-name>}.log
\]

where \(<login-name>\) is a meta-symbol for the user’s actual login.
Once secure shell connectivity is established, type a variation of the `install` command as demonstrated by the table below and follow the prompts issued by this install script.

<table>
<thead>
<tr>
<th>Installation Command</th>
<th>Is root password required initially?</th>
<th>Installer prompts to be aware of</th>
<th>Default installation area</th>
</tr>
</thead>
<tbody>
<tr>
<td>./install</td>
<td>Yes</td>
<td>Installing Intel(R) Math Kernel Library for Linux* version p_... Software packages that are installed using RPMs are available system-wide. We recommend that you install the software using RPM (option 1). This would require root password. If you do not have root password, you can do a local installation in your home folder by choosing option 2 below. Which of the following would you like to do? 1. Install the software using RPM (root password required) - Recommended. 2. Install the software without using RPM database (root password not required). x. Exit Please make a selection: For --nonroot and --nonrpm</td>
<td>./opt/intel/ict/_</td>
</tr>
<tr>
<td>./install --nonroot</td>
<td>No</td>
<td></td>
<td>./intel/ict/_ in user’s home directory</td>
</tr>
<tr>
<td>./install --nonrpm</td>
<td>Yes</td>
<td>Super-user or &quot;root&quot; privileges are required in order to continue. Please enter &quot;root&quot; password. Password:</td>
<td>./opt/intel/ict/_</td>
</tr>
<tr>
<td>./install --nonroot</td>
<td>No</td>
<td></td>
<td>./intel/ict/_ in user’s home directory</td>
</tr>
</tbody>
</table>

Note that Intel® Trace Analyzer and Intel® MPI Benchmarks are only installed on the master node.

By default, the global root directory for the installation of the Intel® Cluster Toolkit is:
/opt/intel/ict/<version#>.<minor-version#>

where <version#> is an integer, and <minor-version#> is an integer. An example would be 2.0.

Within the folder path /opt/intel/ict/<version#>.<minor-version#> one will find the text files:

ictvars.csh
ictvars.sh

and

ictsupport.txt

If one is using Bourne Shell or Korn Shell for the login session, one should type:

. ./ictvars.sh

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

source ./ictvars.csh

The file called:

ictsupport.txt

contains the Package ID and Package Contents information. Please use the information in ictsupport.txt when submitting customer support requests.

For the default installation path, an index file, an FAQ file, and the user’s guide are located in the directory path:

/opt/intel/ict/<version#>.<minor-version#/doc

where as mentioned above, <version#> is an integer, and <minor-version#> is an integer. A complete default folder path to the documentation directory might be:

/opt/intel/ict/2.0/doc

The name of the index file is:

index.htm

The index file can be used to navigate to the FAQ, the release notes, the user’s guide, and an internet accessible Intel® Cluster Toolkit Tutorial.

The name of the FAQ file is:

ICT_FAQ_2.0.htm

The name of the user’s guide file is:

ICT_Users_Guide_2.0.pdf
By default, the local version of the release notes is located in the directory path:

\[/opt/intel/ict/<version#>.<minor-version#>/release_notes\]

The name of the release notes file is:

ICT_Release_Notes_2.0.htm

### 6.1 Installation of Intel® Cluster Toolkit on 32 Bit and 64 Bit Nodes which Share a Common /opt Directory

If there are two cluster systems (one which is 32 bit and the other which is 64 bit) and they share a common file structure, the Intel® Cluster Toolkit installer can be used so as to allow the co-existence of two versions of the cluster tools. In the staging area do the following:

1. Install the EM64T-based version of the Intel® Cluster Toolkit from host A into
   \[/opt/intel/ict/2.0/em64t\].
2. Install the IA32 version of the Intel® Cluster Toolkit from host B into
   \[/opt/intel/ict/2.0/ia32\]. If host B is an EM64T-based node, the user can use the `--arch=x86` option (Figure 1).

**Figure 1 – Installation of the Intel® Cluster Toolkit on a common file share for two cluster systems (one which is 32 bit and the other which is 64 bit)**
The --arch=x86 command-line option will install x86-specific binaries on an Intel® EM64T-based platform.

7. The Parallel Install Capability for the Intel® Cluster Toolkit Installer

Installation of the Intel® Cluster Toolkit assumes that the homogenous computing cluster has ssh connectivity. When a cluster system is configured in such a way that the cluster toolkit software is to be installed on every node of the cluster, the install script will prompt the user with a query regarding how many parallel installations should go on simultaneously. The parallel installation methodology was developed to make the installation process manageable on large clusters.

As an example, suppose the software is being installed on an eight node cluster. The installer could ask the following:

Starting cluster install
Ping test...
  clusternode2... 
    ...ok
  clusternode3... 
    ...ok
  clusternode4... 
    ...ok
  clusternode5... 
    ...ok
  clusternode6... 
    ...ok
  clusternode7... 
    ...ok
  clusternode8... 
    ...ok
Login test...
  clusternode2... 
    ...ok
  clusternode3... 
    ...ok
  clusternode4... 
    ...ok
  clusternode5... 
    ...ok
  clusternode6... 
    ...ok
  clusternode7... 
    ...ok
  clusternode8... 
    ...ok

There are 7 cluster nodes left to install.
You may select 1 to 7 parallel installations (1/2/3/4/5/6/7) [7]:

Entering a value of three, for example, will direct 3 parallel installations to occur, and will produce a message that looks something like:

Do not interrupt this install.
Waiting for cluster node installers...

At the time of this writing, in a worst case scenario, the maximum number of simultaneous parallel installations is 8. This maximum value is applicable to node counts greater than or equal to 8. When the installation completes, the user will see the following message:
8. The Software Architecture of the Intel® Cluster Toolkit

So as to understand the features of the software architecture for the Intel® Cluster Toolkit, one needs to look briefly at the profile infrastructure that is mandated by the MPI Standard. With regards to MPI profiling interfaces, an implementation of the MPI methods must⁴:

1. provide a mechanism through which all of the MPI defined functions may be accessed with a name shift. Thus, all of the MPI functions (which normally start with the prefix "MPI_") should also be accessible with the prefix "PMDI_".
2. ensure that those MPI functions which are not replaced may still be linked into an executable image without causing name clashes.
3. document the implementation of different language bindings of the MPI interface if they are layered on top of each other, so that the profiler developer knows whether they must implement the profile interface for each binding, or can economize by implementing it only for the lowest level routines.
4. where the implementation of different language bindings is done through a layered approach (e.g. the Fortran binding is a set of "wrapper" functions which call the C implementation), ensure that these wrapper functions are separable from the rest of the library. This is necessary to allow a separate profiling library to be correctly implemented, since (at least with Unix-type linker semantics) the profiling library must contain these wrapper functions if it is to perform as expected. This requirement allows the person who builds the profiling library to extract these functions from the original MPI library and add them into the profiling library without bringing along any other unnecessary code.
5. provide a no-op routine MPI_PCONTROL in the MPI library.

The above specifications for a profiling interface within MPI can be illustrated as follows in Figure 2. For Figure 2, the MPI_Send definition in the profile library is a wrapper function to the call to PMPI_Send. When the user’s application calls MPI_Send, the definition within the profile library is used. In contrast to MPI_Send, when MPI_Recv is called within the user’s executable, the definition of MPI_Recv in the MPI library is invoked.

---

⁴ For complete details, please click on the Profile Interface hyperlink at the URL: [http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html](http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html).
Figure 2 – Implementation of a Profile Library and an MPI Library that Conform to the MPI Standard

An example definition for the profile library function `MPI_Send` might look something like the following:

```c
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
{
    int return_value;

    // Start instrumentation
    return_value = PMPI_Send(buf, count, datatype, dest, tag, comm);

    // End instrumentation
}
```

The implementer of the profile library could insert instrumentation before and after the call to `PMPI_Send` so as to gather information about `PMPI_Send`.

Figure 3 takes the abstraction in Figure 2, and illustrates how Intel® MPI Library, the Intel® Trace Collector, and the Intel® Trace Analyzer interact with a user’s application. The user can compile their application with Intel® Trace Collector instrumentation, where the Intel® Trace Collector Library is analogous to the profile library in Figure 2. They can then proceed to run the executable. Optionally, a debugger can be utilized to resolve logic defects in execution for the application. The instrumentation data related to the Intel® Trace Collector will be placed into a trace file. After the executable completes its tasks, the user can proceed to do post-mortem performance analysis with the Intel® Trace Analyzer.
In Figure 3, there can be MPI calls within the executable itself, or the executable can have calls to the ScaLAPACK and Cluster DFT methods within the Intel® Cluster Math Kernel Library. The Basic Linear Algebra Communication Subprograms (BLACS) that are part of the Intel® Cluster Math Kernel Library manage the MPI communication tasks.

Following the MPI Standard, the profiling capability of the Intel® Trace Collector Library provides instrumentation wrappers for the MPI methods (Figure 3), which in turn call PMPI methods within the Intel® MPI Library. Thus, one has interoperability capability between Intel® MPI Library, Intel® Cluster Math Kernel Library, Intel® Trace Collector, and Intel® Trace Analyzer.

9. 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 *Getting Started with Intel® MPI Library* located in `<directory-path-to-Intel-MPI-`
The software architecture for Intel® MPI Library is described in Figure 4. With Intel® MPI Library, the user can choose the best interconnection fabric for running an application on an IA-32, Intel® Itanium® 2, or Intel® EM64T-based cluster. This is done at runtime by setting the 
\texttt{I\_MPI\_DEVICE} environment variable (See Section 9.4). Execution failure can be avoided even if interconnect selection fails. This is especially true for batch computing. For such situations, the sockets interface will automatically be selected (Figure 4) as a backup.

\textbf{9.1 Launching MPD Daemons}

The Intel® MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. In order to run programs compiled with \texttt{mpicc} (or related) commands, you must first set up MPD daemons. It is strongly recommended that users each start and maintain their 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.

9.1.1 How to Set Up MPD Daemons

1. Set up environment variables with appropriate values and directories, e.g., in .cshrc or .bashrc files. At a minimum, set the following environment variables: Ensure that the PATH variable includes the following:
   - The <directory-path-to-Intel-MPI-Library>/bin directory. For example, the <directory-path-to-Intel-MPI-Library>/bin directory path should be set.
   - Directory for Python* version 2.2 or greater.
   - If you are using Intel compilers, ensure that the LD_LIBRARY_PATH variable contains the directories for the compiler library. You can set this variable by using the *vars.(c)sh scripts included with the compiler. Set any additional environment variables your application uses.

2. Create a $HOME/.mpd.conf file that contains your MPD password. Your MPD password is not the same as any Linux login password, but rather is used for MPD only. It is an arbitrary password string that is used only to control access to the MPD daemons by various cluster users. To set up your MPD password:

   password=<your mpd password>

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

   chmod 600 $HOME/.mpd.conf

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

   rsh <node> env
   rsh <node> cat $HOME/.mpd.conf

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

6. Start up the MPD daemons as follows:

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

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

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

   mdptrace

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

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

```
mpdallexit
```

- Individual users should start their own set of MPD daemons. It is not recommended to start MPD as root due to setup problems and security issues.

9.2 The mpdboot Command

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

9.3 Compiling and Linking with Intel® MPI Library

This section describes the basic steps required to compile and link an MPI program, when using the 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 used for `<compiler>` appears in your `PATH`. If you are using Intel® compilers, insure that the compiler library directories appear in `LD_LIBRARY_PATH` environment variable. For example, regarding the Intel® 8.1 compilers, execution of the appropriate set up scripts will do this automatically:

```
/opt/intel_cc_81/bin/iccvars.[c]sh
```

and

```
/opt/intel_fc_81/bin/ifortvars.[c]sh
```

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

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

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

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 `mpicc` and other compiler commands, including commands for other compilers and languages.

9.4 Selecting a Network Fabric or Device

The 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. Select alternative devices via the command line using the `I_MPI_DEVICE` environment variable. The following network fabric types are supported by Intel® MPI Library:
### Possible Interconnection-Device-Fabric Values for the `I_MPI_DEVICE` Environment Variable

<table>
<thead>
<tr>
<th>Value</th>
<th>Interconnection Device Fabric Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock</td>
<td>TCP/Ethernet/sockets (default)</td>
</tr>
<tr>
<td>shm</td>
<td>Shared-memory only (no sockets)</td>
</tr>
<tr>
<td>ssm</td>
<td>TCP + shared-memory (for SMP clusters connected via Ethernet)</td>
</tr>
<tr>
<td>rdma[:&lt;provider&gt;]</td>
<td>InfiniBand*, Myrinet*, etc. (specified via the DAPL* provider)</td>
</tr>
<tr>
<td>rdssm[:&lt;provider&gt;]</td>
<td>TCP + shared-memory + DAPL* (for SMP clusters connected via RDMA-capable fabrics)</td>
</tr>
</tbody>
</table>

### 9.5 Running an MPI Program Using Intel® MPI Library

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

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

The only required option for the `mpiexec` command is the `-n` option to set the number of processes. If you are a network fabric other than the default fabric (sock), use the `-env` option to specify a value to be assigned to the `I_MPI_DEVICE` variable. For example, to run an MPI program while using the `ssm` device, use the following command:

```
mpiexec -n <# of processes> -env I_MPI_DEVICE ssm ./a.out
```

To run an MPI program while using the `rdma` device, use the following command:

```
mpiexec -n <# of processes> -env I_MPI_DEVICE rdma[:<provider>] ./a.out
```

Any supported device can be selected. See the section titled, Selecting a Network Fabric or Device in `<directory-path-to-Intel-MPI-Library>/doc/Getting_started.pdf`, or the section titled, `I_MPI_DEVICE` in `<directory-path-to-Intel-MPI-Library>/doc/Reference_Manual.pdf`.

### 9.6 Experimenting with Intel® MPI Library

For the experiments that follow, it is assumed that a computing cluster has at least 2 nodes and there are two symmetric multi-processors (SMPs) per node. Start up the MPD daemons by issuing a command such as:

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

Type the command:

```
mpdtrace
```

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

```
clusternode1
clusternode2
```

assuming that the two nodes of the cluster are called clusternode1 and clusternode2. The actual response will be a function of the user’s 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 a local user area, the patron should create a test directory called:

    test_intel_mpi/

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

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

Compile the test applications into executables using the following commands:

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

Issue the mpiexec commands:

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

The output from testcpp should look something like:

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

If one has successfully run the above applications using Intel® MPI Library, one can now run (without re-linking) the four executables on clusters that use DAPL interfaces to alternative interconnection fabrics. If one encounters problems, please see the section titled Troubleshooting within the document Getting Started with Intel® MPI Library located in <directory-path-to-Intel-MPI-Library>/doc/Getting_started.pdf for possible solutions.

Assuming that the user has an rdma device fabric installed on the cluster, the user can issue the following commands for the four executables so as to access that device fabric:

    mpiexec -env I_MPI_DEVICE rdma -n 2 ./testf
    mpiexec -env I_MPI_DEVICE rdma -n 2 ./testf90
    mpiexec -env I_MPI_DEVICE rdma -n 2 ./testc
    mpiexec -env I_MPI_DEVICE rdma -n 2 ./testcpp

The output from testf90 using the rdma device value for the I_MPI_DEVICE environment variable should look something like:

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

9.7 Controlling MPI Process Placement

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 a user’s application, particularly for clusters with SMP (symmetric multi-processor) nodes.

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

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

The results should be something like:

```
clusternode1
clusternode2
```

Since each node of the cluster is a 2-way SMP, and 4 processes are to be used for the application, the next experiment will distribute the 4 processes such that 2 of the processes will execute on clusternode1 and 2 will execute on clusternode2. For example, one might issue the following commands:

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

The following output should be produced for the executable testc:

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

In general, if there are $i$ nodes in the cluster and each node is $j$-way SMP system, 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 that the user would have to fill in appropriate host names for `<nodename-1>` through `<nodename-$i$>` with respect to their 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`.

To make inquiries about Intel® MPI Library, visit the URL: [http://premier.intel.com](http://premier.intel.com).
10. Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer

The Intel® Cluster Toolkit installer will place the Intel® Trace Collector package into a directory path on the cluster system such as demonstrated with the following example:

```
/opt/intel/ict/2.0/itc/6.0
```

In the directory where the Intel® Trace Collector was untarred\(^5\), users should see the following sub-directory and file contents:

```
bin/
  ChangeLog
doc/
eula.txt
examples/
  include/
  install
lib/
log
man/
  README.installation
relnotes.txt
SilentInstallConfigFile.ini
SilentInstallFParse.awk
slib/
sourceme.csh
sourceme.sh
support
third_party/
tmp/
```

Complete user documentation for the Intel® Trace Collector can be found within the file:

```
```

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

With respect to the Intel® Trace Analyzer, the Intel® Cluster Toolkit installer will also place the contents of the Intel® Trace Analyzer package into a directory path on the cluster system such as:

```
/opt/intel/ict/2.0/ita/6.0
```

---

\(^5\) `tar` is used to create a Tape ARchive. The resulting file is known as a tarball. Untarring is the process of getting the files out of a tarball.
In the directory where the Intel® Trace Analyzer was inserted, one should see the following sub-directory and file contents:

```
bin/
doc/
eula.txt
examples/
  ita_support.txt
lib/
  relnotes.txt
  SilentInstallConfigFile.ini
```

The user must add the `<directory-path-to-ITA>/bin` directory to their PATH environment variable. Note that `<directory-path-to-ITA>` is the absolute directory path for the Intel® Trace Analyzer. By default, the folder path might be `/opt/intel/ict/2.0/ita/6.0`.

Complete user documentation for the Intel® Trace Analyzer can be found within the file:

```
<directory-path-to-ITA>/doc/ITA-ReferenceGuide.pdf
```

In reference to the Intel® Trace Analyzer, the user will need to set environment variables. The environment variable syntax below assumes that the user’s login session is using the Bourne Shell or the Korn Shell.

```
export LD_ASSUME_KERNEL=2.4.1
```

For C Shell, the syntax should be:

```
setenv LD_ASSUME_KERNEL 2.4.1
```

The environment variable `PAL_ROOT` points to the root directory path of the Intel® Trace Analyzer installation.

Recall that `<directory-path-to-ITC>` might have an actual setting such as `/opt/intel/ict/2.0/itc/6.0` on the system where the Intel® Trace Collector is installed. The shell script files called `sourceme.sh` and `sourceme.csh` that are located in `<directory-path-to-ITC>` on the user’s system contain environment variable assignments for `VT_ROOT`, `PATH`, `MANPATH`, `CLASSPATH`, and `LD_LIBRARY_PATH`.

Note that the above environment variable settings are managed by the shell script files `ictvars.sh` and `ictvars.csh` in the directory:

```
/opt/intel/ict/<version#>.<release#>
```

As mentioned previously, if one is using Bourne Shell or Korn Shell for the login session, one should type:

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

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

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

Figure 5 shows a sample of the various panel displays of the Intel® Trace Analyzer that the user can generate when doing post-mortem performance analysis of an application that has been instrumented with the Intel® Trace Collector.
10.1 Compiling MPI Programs in conjunction with the Intel® Trace Collector Header Files

User source files without calls to the Intel® Trace Collector APIs can be compiled with the usual methods and without any special precautions. However, source files that do contain calls to the Intel® Trace Collector APIs must include the appropriate header files:

- VT.h for C and C++
- VT.inc for Fortran

that exist in the Intel® Trace Collector directory:

```
<directory-path-to-ITC>/include
```

To compile these source files, the path to the Intel® Trace Collector header files must be passed to the compiler. On most systems, this is done through the compiler command-line with the `-I` flag option. An example of this would be:

```
-I$\{VT_ROOT\}/include
```
10.2 Linking MPI Programs with the Intel® Trace Collector Libraries

The Intel® Trace Collector library, libVT.a, contains entry points for all MPI methods. The libVT.a library must be linked against the user’s application object files before listing Intel® MPI Library or an equivalent. In the section of this user’s guide regarding Intel® MPI Library, there was a discussion about creating a user directory called:

```
test_intel_mpi/
```

where the source file contents of `test_intel_mpi` should be:

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

The experimentation which follows will involve linking instrumentation into the MPI applications above. Note, that this instrumentation process is illustrated in Figure 3 with the Intel® Trace Collector and conforms to the model proposed in Figure 2. The following MPI compilation commands should be issued for the MPI source files for Itanium® 2-based systems:

```
mpiifort test.f -g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm -o testf_inst
mpiifort test.f90 -g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm -o testf90_inst
mpiicc test.c -g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm -o testc_inst
mpiicpc test.cpp -g -lmpiic -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm -o testcpp_inst
```

Note that with respect to the MPI compilation commands above, the following library options were added to permit instrumentation by an MPI profiling library on Itanium® 2-based systems:

```
-L{VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm
```

Also realize that for compiling the C++ application `test.cpp` with the Intel® C/C++ compiler (mpiicpc) and with Intel® Trace Collector requires an additional library called -lmpiic. The rule for this is that if one uses the MPI-2 C++ application programming interfaces (APIs) with the Intel® C/C++ compilers, then Intel® Trace Collector cannot intercept the MPI calls as they are written. They have to be mapped to C function calls first, via an MPI implementation-specific wrapper library, which with respect to the command-line, has to be placed in front of the Intel® Trace Collector library. The name of that wrapper library for the mpiicpc compilation driver is -lmpiic. Additional compilation information regarding trace collections with C++ compilers can be found in the Intel® Trace Collector folder path:

```
```

For Intel® Pentium® 4, Intel® Xeon®, and Intel® EM64T-based architectures, the MPI compilation commands should be:

```
mpiifort test.f -g -L${VT_ROOT}/lib -lVT -ldwarf -lelf -lpthread -lm -o testf_inst
mpiifort test.f90 -g -L${VT_ROOT}/lib -lVT -ldwarf -lelf -lpthread -lm -o testf90_inst
mpiicc test.c -g -L${VT_ROOT}/lib -lVT -ldwarf -lelf -lpthread -lm -o testc_inst
mpiicpc test.cpp -g -lmpiic -L${VT_ROOT}/lib -lVT -ldwarf -lelf -lpthread -lm -o testcpp_inst
```

where the library reference -lvtunwind has been omitted. Also, remember that the special wrapper library called -lmpiic was used for the mpiicpc command-line.
Before running the MPI executable, two Intel® Trace Collector environment variables will be introduced. They are:

```plaintext
VT_LOGFILE_PREFIX
```

and

```plaintext
VT_PCTRACE
```

The `VT_LOGFILE_PREFIX` environment variable will direct the generation of instrumentation data to an area of the user’s choice. The `VT_PCTRACE` environment variable enables runtime program counter tracing. The value of 5 indicates the number of call levels of procedures that will have source location information recorded. Since the unwinding of the call stack each time a function is called can be very costly the setting of the `VT_PCTRACE` environment variable should be handled with discretion. Notice that the `-g` option was added to each of the compilation command-lines listed above. The debug profiling instrumentation that is generated during compilation will be used in conjunction with the `VT_PCTRACE` environment variable to correlate trace events with the source code in the user’s application.

Additional documentation about Intel® Trace Collector environment variables can be found in the Intel® Trace Collector folder path:

```plaintext
```

For the environment variables mentioned, the following sequence of commands could be issued if the user is running the Bourne Shell:

```bash
export VT_LOGFILE_PREFIX=${PWD}/inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
mpiexec -n 2 ./testf_inst
mpiexec -n 2 ./testf90_inst
mpiexec -n 2 ./testc_inst
mpiexec -n 2 ./testcpp_inst
```

The environment variable `VT_LOGFILE_PREFIX` will be used by the Intel® Trace Collector to place trace data into a folder other than the current directory. For the example above the subfolder is `inst` which is an acronym for instrumentation. The equivalent syntax using C Shell would be:

```bash
setenv VT_LOGFILE_PREFIX ${PWD}/inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
setenv VT_PCTRACE 5
mpiexec -n 4 ./testf_inst
mpiexec -n 4 ./testf90_inst
mpiexec -n 4 ./testc_inst
mpiexec -n 4 ./testcpp_inst
```

Using the environment variable `VT_LOGFILE_PREFIX` can help prevent working directories from being cluttered with instrumentation data. This is especially true when a user needs to do multiple experiments with Intel® Trace Collector.

The user should list the contents of the `inst` directory after completing the `mpiexec` commands above. The contents of this folder should look something like:
using the command `ls -aC -width=80`. Intel® Trace Collector by default partitions the instrumentation data into subcomponent files. This model is known as the Structured Trace Format (STF). Hence, the trace file names above have the acronym `stf` embedded within them. The Intel® Trace Collector has the capability to generate these trace file subcomponents in parallel and thus help reduce performance penalties due to instrumentation.

Please note that the display results that follow 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 cluster configuration to cluster configuration. Users may see displays that may be unique to their systems.

To begin the analysis for the executable `testf90_inst`, type the command:

```
traceanalyzer inst/testf90_inst.stf &
```

where it is assumed that `inst` is a subdirectory to the current working directory. This will produce the Function Profile Chart which is the default display panel. The view might look something like (Figure 6):
Figure 6 – The Function Profile Chart display panel for Intel® Trace Analyzer

Figure 6 contains menu items for which the user can make selections from. If one clicks on the up-arrow button (i.e. the maximize button) shown in the upper right region of Figure 6, the main display panel will fully occupy the initial white space (Figure 7).
Figure 7 – The Function Profile Chart display panel for Intel® Trace Analyzer after Pressing the Maximize Button (the up arrow)

Notice that there are 2 histograms shown. The blue histogram shows a summary of execution time that has been spent exclusively within the application. The red histogram shows a summary of execution time exclusively within the MPI library. TSelf is an acronym for Time-self. TTotal is an abbreviation for Time Total and TSelf/Call is the ratio of time divided by the number of calls. The label TSelf appears twice because the Time-self information is presented in text as well as a colored histogram. TTotal for the Group Application row is the timing sum for Group Application (exclusive time) and the time contributions for making the MPI calls.

There are also tabs called Flat Profile, Load Balance, Call Tree, and Call Graph which allow the user to view the Function Profile Chart from various perspectives. Note that the Flat Profile tab is the default. The name Flat Profile is the default tab as shown in Figure 7. With respect to Figure 7, one can press each of the tabs, and one will see a different rendering of the Function Profile.
With respect to Figure 7, if one clicks on the Charts menu item, a pop-down menu will appear as shown in Figure 8. One can select the charts titled TimeScale, Event Timeline, Qualitative Timeline, Quantitative Timeline, Function Profile, Message Profile, and Collective Operations Profile. Throughout the rest of this document, to select a chart such as Event Timeline, the menu notation that will be used is Charts->Event Timeline. If one does a selection of the Event Timeline Chart, as illustrated in Figure 8, the display shown in Figure 9 will appear.

These chart panels can also be selected with a sequence of keys such as Ctrl+Alt+E, which will display the Event Timeline chart. Note that if the Ctrl, Alt, and E keys on the left side of the keyboard do not generate an Event Timeline chart, please try using the Ctrl, Alt, keys on the right side of the keyboard along with the E key.
The top panel in Figure 9 is an example display of the Event Timeline Chart and it monitors what the individual MPI processes did during execution of the `mpiexec` command. Notice in Figure 9 that 4 MPI processes have been profiled (P0 through P3). This conforms to the `mpiexec` commands which were described earlier. The lower half panel in Figure 9 is the Function Profile Chart panel that was demonstrated in Figures 6 and 7.

For the Event Timeline Chart in Figure 9, the horizontal axis represents a time interval in the runtime of the inspected program. Notice that a process rectangle consists of the concatenation of application activities (blue rectangles) and MPI activities (red rectangles). Messages sent between processes are indicated as black lines connecting the sender and receiver processes.

With respect to the time line panel, one can zoom into a particular region by pressing the index finger mouse key and panning over the region that one wishes to zoom in on. Figure 10 illustrates the initiation of a zoom operation where yellow highlighting appears signifying the time interval that is to be zoomed-in on.
Figure 10 – Initiation of a zoom operation for the Timeline Panel

Figure 11 shows the result of doing the zoom. The keyboard key \( u \) is a hot key which will allow the user to undo the zoom operation.
Figure 11 – The resulting Timeline display as a result of zooming in on the approximate time interval 0.0028 seconds to approximately 0.0053 seconds
It was mentioned previously that black lines drawn between the activities in the Timeline view signify inter-process communication. If one takes the mouse and positions it on one of the messages (e.g. the line that starts at P1 and reaches to P0), and then clicks on the right mouse button, a context menu will appear. If one chooses the Details on Messages entry, the message panel illustrated in Figure 12 will appear.

![Figure 12 – Context menu for drilling down to application source code](image)
In Figure 13, if one clicks on one of the rows listed, say the first row, that row selection will be highlighted in black. If one proceeds to press the icon under the Show Source column, the result will be a new panel called Source View (Figure 14):
Figure 14 – Source view for an MPI message being sent from process 1

There is a Call stack sub-panel in this view which tells you the name of the file that is being viewed and the line number. The label called Process 1 (Figure 14) at the top of the display is a toggle button. The user can click on it and select the pop-up row called Process 0. This will show source code line in the application that performed the receive operation (Figure 15).
Figure 15 – Source view for an MPI message being received by process 0

For MPI applications where there may be a performance bottleneck due to inter-process communication, the use of this drill-down feature may help the user to isolate send and receive statement numbers that are causing the performance bottleneck.

The user can close the source view displays in Figures 14 or 15 by depressing the x-button in the upper right corner of these panels. For the main display panel (Figure 11) if one hits the u hot button, the user should return to the display in Figure 9. As mentioned earlier in this chapter, for the Charts menu item, one could proceed to Qualitative Timeline, Quantitative Timeline, Collective Profile, and Collective Operations Profile in specified sequence. The result display configuration is illustrated in Figure 16.
Figure 16 – An Intel® Trace Analyzer session that uses all of the view charts simultaneously

If one positions the mouse over the Qualitative Display panel (Figure 17) and presses the right mouse key to get a context sensitive menu, the context sensitive menu will allow the user to close the given chart.
Figure 17 – Context Sensitive Menu for the Chart displays

The resulting Intel® Trace Analyzer view is demonstrated in Figure 18.
Figure 18 – Result of closing the Qualitative Timeline Chart

In the lower right section of Figure 18 there are two labels involving line segments with arrows at both ends. The first double arrow line segment is colored black and it signifies sender and receiver communication. This was already discussed in regards to the Event Timeline panel. The second double arrow line segment is a crisscross and is colored blue. This blue crisscross pattern represents collective operations such as arithmetic reductions.

With respect to Figure 18, one can adjust the size of a particular chart panel. This may cause the view of another chart panel to become smaller. For example, if one wishes to better view the Message Profile Chart, one can slide the left margin of the message chart in a leftward direction (Figure 19). This will shrink the view of the Function Profile display.
Processes 1 through 3 are communicating with process 0. The color-coded legend indicates that process three took the most time in terms of inter-process communication. In terms of a temperature analogy, blue indicates a cool temperature, and the other color extreme which is red indicates a hot temperature.

The user can terminate the Intel® Trace Analyzer session by going to the Main panel and select the menu path File->Quit.

The user can proceed to issue the shell command:

```
traceanalyzer inst/testcpp_inst.stf &
```

and repeat the same Intel® Trace Analyzer commands that were used for `inst/testf90_inst.stf`.

Another alternative approach is to issue the shell command:

```
traceanalyzer &
```

and use the menu selection File->Open feature (Figure 19). This will generate an Open a Tracefile file selection panel such as that shown in Figure 20:
For Figure 20, note that the exact folder content will be a function of where the user is doing their work with the Intel® Trace Analyzer. One can toggle on the inst folder in Figure 20 and click on the open button to transition into that folder. This is illustrated in Figure 21. In Figure 21, if one clicks on a particular trace file, it will be highlighted in black and the File Name: dialog box will be filled in. One needs to simply press the Open button, and the result from selecting the trace file illustrated in Figure 21 is illustrated in Figure 22.
Figure 21 – Selection a trace file using the File-Open menu path

With respect to Figure 22, which is the Function Profile display for `test.cpp`, one can repeat the process of opening charts, and doing an Event Timeline drill-down to application source.
10.2.1 Simplifying the Compile and Link Command between Intel® MPI Library and the Intel® Trace Collector

Regarding the source file contents of test_intel_mpi, the methodology for instrumenting an MPI application with Intel® MPI Library and Intel® Trace Collector has been to use the –g debugging option along with the following link options on Itanium® 2-based systems:

```
-L$(VT_ROOT)/lib -lVT -lvttunwind -ldwarf -lelf -lpthread -lm
```

For Intel® Pentium® 4, Intel® Xeon®, and Intel® EM64T-based architectures, the MPI compilation commands should use –g option along with:

```
-L$(VT_ROOT)/lib -lVT -ldwarf -lelf -lpthread -lm
```

Also for compiling the C++ applications the library called –lmpiic needed to be included.

An alternative to the above is to use the –t or –trace option to link the resulting executable against the Intel® Trace Collector when invoking the mpiifort, mpiicc, and mpiicpc MPI-compilation commands. The shell command-line syntax for compiling the examples in test_intel_mpi with the –trace option might be:
If one wishes to capture internal Intel® MPI Library state tracing information in conjunction with Intel® Trace Analyzer and Collector one can set the environment variable VT_DETAILED_STATES to enable such trace information. By default, this environment variable is not activated. Positive integer value settings will reveal increasingly more information on the internal states. Setting the VT_DETAILED_STATES environment variable to a value of 5 might have the following respective Bourne Shell and C Shell syntax specifications:

```bash
export VT_DETAILED_STATES=5
setenv VT_DETAILED_STATES 5
```

This environment variable is used in conjunction with special Intel® MPI Library instrumentation libraries. For example, to compile and link the four test applications that have been discussed thus far, issue the following shell commands:

```bash
mpiifort test.f -g -t=log -o testf_state_inst
mpiifort test.f90 -g -t=log -o testf90_state_inst
mpiicc test.c -g -t=log -o testc_state_inst
mpiicpc test.cpp -g -t=log -o testcpp_state_inst
```

Use the `-t=log` or `-trace=log` options to link the resulting executable against the logging versions of Intel® MPI libraries and the Intel® Trace Collector. The four executables that have been compiled and linked with Intel® MPI Library state-tracing information can be run as follows:

```bash
export VT_LOGFILE_PREFIX=${PWD}/state_inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
export VT_DETAILED_STATES=5
mpiexec -n 4 ./testf_state_inst
mpiexec -n 4 ./testf90_state_inst
mpiexec -n 4 ./testc_state_inst
mpiexec -n 4 ./testcpp_state_inst
```

With regards to the VT_DETAILED_STATES environment variable, the protocol files called

```bash
state_inst/testcpp_state_inst.prot
state_inst/testc_state_inst.prot
state_inst/testf90_state_inst.prot
state_inst/testf_state_inst.prot
```

contain the following verification information about this environment variable:
When starting up the Intel® Trace Analyzer and toggling the Call Tree tab on the Function Profile Chart, one might see a call tree that looks something like:

![Figure 22 – Toggling on the Call Tree tab for Function Profile Chart](image)

If one proceeds to move the mouse over the bottom Group MPI label and right-clicks on the mouse to generate a context menu, a list of options will appear in the context menu (Figure 23). If the Ungroup Group MPI entry is selected, the call tree for the bottom Group MPI label will be expanded.
Figure 23 – Context menu for selecting the option Ungroup Group MPI

When applicable, the sequence of bringing up the context menu in Figure 23 may be invoked repeatedly on group members in the calling tree. This allows one to view the results of using the `VT_DETAILED_STATES` environment variable (Figure 24).
Figure 24 – Use of the VT_DETAILED_STATES environment variable to expose internal states of the Intel® MPI Library

![IntelliTrace Analyzer](image)

Figure 25 shows the Event Timeline Chart and how these state events are also recorded in the timeline display.
Figure 25– The Event Timeline and Function Profile Charts demonstrate internal states of the Intel® MPI Library as a result of using the VT_DETAILED_STATES environment variable.

10.2.3 How to Get Additional Help Regarding the Intel® Trace Collector
Recall that complete user documentation for the Intel® Trace Collector can be found within the file:


Also, note that <directory-path-to-ITC> is the absolute directory path to the folder such as /opt/intel/ict/2.0/ictc/6.0, which is where the doc/Intel_Trace_Collector_Users_Guide.pdf file resides, that was listed above. To make inquiries about the Intel® Trace Collector, visit the URL: http://premier.intel.com.

10.3 Experimenting with the MPI Examples in the Directory ${VT_ROOT}/examples
Recall that there exists a directory called ${VT_ROOT}/examples. The user should make a copy of this directory into a local directory partition where experimentation can take place. This local directory partition should be accessible to all nodes in the cluster. Within the sub-directory called examples, there is a script called palmake. If the user types the command:
the executables called vtjacobic, vtjacobif, vtallpair, and vtallpairc will be created. Additional information about the shell script called palmake can be found in the text file called README.examples within the examples sub-directory. Assuming that the user is working with C Shell, they might issue the following sequence of commands:

```
setenv VT_LOGFILE_PREFIX $(PWD)/inst
rm -rf $(VT_LOGFILE_PREFIX)
mkdir $(VT_LOGFILE_PREFIX)
setenv VT_PCTRACE 5
mpiexec -n 4 ./vtallpair
```

Recall that the environment variables VT_LOGFILE_PREFIX and VT_PCTRACE respectively control the directory where instrumentation data is placed and the call stack depth. The mpiexec command above uses 4 MPI processes.

After this MPI application completes execution, there will be a collection of analysis files in the inst subdirectory that look something like:

```
vtallpair.prot
vtallpair.stf
vtallpair.stf.dcl
vtallpair.stf.frm
vtallpair.stf.gop
vtallpair.stf.gop.anc
vtallpair.stf.msg
vtallpair.stf.msg.anc
vtallpair.stf.pr.0
vtallpair.stf.pr.0.anc
vtallpair.stf.sts
```

The user can proceed to visualize the analysis data with a command such as:

```
traceanalyzer inst/vtallpair.stf &
```
As with the previous tracing examples, the Intel® Trace Analyzer main panel along with the Function Profile Chart will immediately appear. Figure 26 shows the maximized chart displays for the Event Timeline and the Function Profile. Notice in Figure 26 for the Event Timeline Chart, that there are two types of line segments drawn between the process activities. Recall from the explanation regarding Figure 18 that the blue line segments describe message collective operations, and the black line segments in the right portion of Figure 26 describe inter-process communication.

If one moves the mouse over one of the blue line segments in the left section of the Event Timeline Chart in Figure 26, and right clicks on the mouse, a context menu will appear with the Detail on Function, Collective selection item (Figure 27). If one proceeds to select the row called Details on Function, Collective, a new display panel will appear. For the given example, the display panel will refer to the collective operation.

Figure 26 - Intel® Trace Analyzer panel showing the Timeline and Function Profile Charts
Figure 27 – Context Sensitive Pop-up Menu for getting detailed information on collective operations

Figure 28 shows the Details on Collective MPI_Barrier display panel. If one clicks on row with the notation MPI_Barrier in Figure 28, that row will be highlighted in black. One can proceed to click on arrow icon that is the left of the label MPI_Barrier in Figure 28. The result is a display of the MPI processes that are involved in the MPI_Barrier operation. This is illustrated in Figure 29.
Figure 28 – Display Panel for the associated collective operation that was selected in Figure 27

Figure 29 – Display of the MPI processes that are involved in the MPI_Barrier operation

One can proceed to click on any of the Show Source icons that reside in the Show Source column label in Figure 29.

The result will be the source listing shown in Figure 30. The particular MPI collective operation that the blue line segments represent in Figures 26 and 27 is an MPI_Barrier call.
Figure 30 – MPI collective operation that was selected using the context menu selection in Figure 26 and Figure 27

In Figure 30, the Process 0 label is part of a push button bar. When it is depressed, a pop-down menu appears showing the other MPI processes that are involved. The source code for each of the other MPI processes that are involved in the MPI barrier operation can also be displayed. Figure 31 shows the intermediate state of toggling from the source code line for process 0 to that of process 1.
10.4 Experimenting with Intel® Trace Analyzer Examples in the Directory <directory-path-to-ITA>/examples

If one goes to the directory <directory-path-to-ITA>/examples, one will find the following key trace analysis files:

- poisson.icomm.single.stf
- poisson_sendrecv.single.stf

Figure 31 – The intermediate state of toggling from the source code line for process 0 to that of process 1
To begin the analysis of the trace data one can simply type a command such as:

```
traceanalyzer poisson_icomm.single.stf &
```

A maximized view for the Event Timeline chart and the Function Profile chart are shown in Figure 32.

![Figure 32 - Intel® Trace Analyzer Display for 16 MPI Processes](image)

The given application that is being analyzed used 16 MPI processes during execution.
10.4.1 How to Get Additional Help Regarding the Intel® Trace Analyzer

Recall that complete user documentation for the Intel® Trace Analyzer can be found within the file:

<directory-path-to-ITA>/doc/ITA-ReferenceGuide.pdf

Also, note that <directory-path-to-ITA> is the absolute directory path to a folder such as /opt/intel/ict/2.0/ita/6.0, which is where the doc/ITA-ReferenceGuide.pdf file resides, that was listed above.

10.5 Experimenting with Intel® Trace Analyzer and Collector 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. An example shell command-line syntax for linking such a library is:

mpiicc test.c -o testc_fs -L${VT_ROOT}/lib -lVTfs -ldwarf -lelf -lnsl -lpthread -lm

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

In case execution failure by the application, the fail-safe library freezes all MPI processes and then writes out the trace file. Figure 33 shows an Intel® Trace Analyzer display for test.c.
Figure 33 – Intel® Trace Analyzer Display of Fail-Safe Trace Collection by Intel® Trace Collector

Figure 34 shows the Intel® Trace Analyzer display for the same example with fail-safe instrumentation removed. Note that the execution display in Figure 33 indicates that fail-safe tracing is a bit slower than that of standard tracing by Intel® Trace Collector.
Recall that complete user documentation regarding fail-safe tracing for the Intel® Trace Collector can be found within the file:


on the system where the Intel® Trace Collector is installed. To make inquiries about the Intel® Trace Analyzer, visit the URL: http://premier.intel.com.

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

The installation process for Intel® Cluster MKL on the cluster system will produce a sub-directory called .../cmkl/8.0.1. The default directory path for the library installation process is:

/opt/intel/ict/2.0/cmkl/8.0.1

The contents of the .../cmkl/8.0.1 sub-directory should be:
Complete user documentation for the Intel® Cluster Math Kernel Library 8.0.1 can be found within the directory path:

\(<\text{directory-path-to-mkl}>/\text{doc}\)

where \(<\text{directory-path-to-mkl}>\) is the absolute directory path to where the Intel® Cluster MKL files and sub-directories are installed on the cluster system.

In order to ensure that the correct support libraries are linked on Red Hat* Enterprise Linux 3.0, the environment variable \(\text{LD_ASSUME_KERNEL}\) must be set. This environment variable was referenced in the installation section for Intel® Trace Collector and Intel® Trace Analyzer. The syntax for this environment variable might be:

\(\text{export LD_ASSUME_KERNEL}=2.4.1\)

In the directory path:

\(<\text{directory-path-to-mkl}>/\text{tests/scalapack}\)

the user can type the command:

\(\text{gmake lib64 F=intel80 mpi=intelmpi20 LIBdir=<directory-path-to-mkl>/lib/64}\)

Note that the \texttt{make} command above is applicable to Itanium® 2-based systems. This makefile creates and runs executables for the ScaLAPACK (SCAlable LAPACK) examples.

\(<\text{directory-path-to-mkl}>/\text{tests/scalapack/source/TESTING}\)

For Intel® Pentium® 4, and Intel® Xeon® architectures, the \texttt{make} command might be:

\(\text{gmake lib32 F=intel80 mpi=intelmpi20 LIBdir=<directory-path-to-mkl>/lib/32}\)

Finally, for the Intel® EM64T-based architecture, the \texttt{make} command could be:

\(\text{gmake libem64t F=intel80 mpi=intelmpi20 LIBdir=<directory-path-to-mkl>/lib/em64t}\)

The user can invoke an editor to view the results in each of the "*.txt" files that have been created. These "*.txt" files also reside in the directory path:
As an example result, the file "cbrd_ia32_intelmpi20_intel_noopt.txt" might have something like the following in terms of contents for a run on a cluster using 4 MPI processes. The cluster that generated this sample output consisted of 4 nodes. The text file was generated by the corresponding executable xcbrd_ia32_intelmpi20_intel_noopt.

SCALAPACK Bidiagonal reduction
'MPI machine'

Tests of the parallel complex single precision bidiagonal reduction routines.
The following scaled residual checks will be computed:

\[ \frac{\|A - QBP\|}{\|A\| \cdot \text{eps} \cdot N} \]

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.
M : The number of rows of the matrix A.
N : The number of columns of the matrix A.
NB : The size of the square blocks the matrix A is split into.
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
BRD time : Time in seconds to reduce the matrix
MFLOPS : Rate of execution for the bidiagonal reduction.

The following parameter values will be used:

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>NB</th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>17</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Relative machine precision (eps) is taken to be 0.596046E-07

Routines pass computational tests if scaled residual is less than 10.000

<table>
<thead>
<tr>
<th>TIME</th>
<th>M</th>
<th>N</th>
<th>NB</th>
<th>P</th>
<th>Q</th>
<th>BRD Time</th>
<th>MFLOPS</th>
<th>Residual</th>
<th>CHECK</th>
</tr>
</thead>
<tbody>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>0.37</td>
<td>0.86</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>1.35</td>
<td>0.69</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>3.52</td>
<td>1.18</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>3.47</td>
<td>1.18</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>12.04</td>
<td>0.51</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>11.42</td>
<td>0.49</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>11.87</td>
<td>0.49</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>14.12</td>
<td>0.55</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>21.66</td>
<td>0.34</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>21.09</td>
<td>0.42</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>20.50</td>
<td>0.39</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>21.92</td>
<td>0.35</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>15.07</td>
<td>0.54</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>14.75</td>
<td>0.52</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>14.33</td>
<td>0.44</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>15.70</td>
<td>0.41</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.18</td>
<td>0.00</td>
<td>0.83</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>0.01</td>
<td>0.78</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0.00</td>
<td>0.26</td>
<td>1.18</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0.00</td>
<td>0.27</td>
<td>1.18</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.10</td>
<td>0.14</td>
<td>0.41</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.05</td>
<td>0.27</td>
<td>0.46</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0.03</td>
<td>0.47</td>
<td>0.59</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0.02</td>
<td>0.60</td>
<td>0.46</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.85</td>
<td>0.33</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.95</td>
<td>0.41</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.81</td>
<td>0.43</td>
<td>PASSED</td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.92</td>
<td>0.41 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.67</td>
<td>0.62 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.03</td>
<td>0.74</td>
<td>0.56 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>0.65</td>
<td>0.44 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>0.35</td>
<td>0.50 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.00</td>
<td>0.15</td>
<td>0.86 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0.00</td>
<td>0.15</td>
<td>0.64 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.00</td>
<td>0.28</td>
<td>1.18 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0.00</td>
<td>0.28</td>
<td>1.18 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.17</td>
<td>0.08</td>
<td>0.40 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0.05</td>
<td>0.27</td>
<td>0.47 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.02</td>
<td>0.81</td>
<td>0.58 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0.02</td>
<td>0.61</td>
<td>0.47 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.03</td>
<td>1.34</td>
<td>0.37 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0.03</td>
<td>1.29</td>
<td>0.42 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.03</td>
<td>0.89</td>
<td>0.44 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.41 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.02</td>
<td>0.97</td>
<td>0.47 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0.02</td>
<td>1.05</td>
<td>0.45 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.03</td>
<td>0.76</td>
<td>0.49 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0.04</td>
<td>0.63</td>
<td>0.65 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.01</td>
<td>0.12</td>
<td>0.83 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.01</td>
<td>0.10</td>
<td>1.03 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.00</td>
<td>0.25</td>
<td>1.18 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0.00</td>
<td>0.25</td>
<td>1.18 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.03</td>
<td>0.46</td>
<td>0.50 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.03</td>
<td>0.42</td>
<td>0.53 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.04</td>
<td>0.34</td>
<td>0.57 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0.03</td>
<td>0.43</td>
<td>0.52 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.05</td>
<td>0.62</td>
<td>0.39 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.26</td>
<td>0.13</td>
<td>0.39 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.06</td>
<td>0.59</td>
<td>0.49 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>17</td>
<td>13</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0.07</td>
<td>0.48</td>
<td>0.52 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.05</td>
<td>0.43</td>
<td>0.44 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.05</td>
<td>0.47</td>
<td>0.47 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.05</td>
<td>0.44</td>
<td>0.56 PASSED</td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0.07</td>
<td>0.35</td>
<td>0.47 PASSED</td>
<td></td>
</tr>
</tbody>
</table>

Finished 64 tests, with the following results:
64 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 a user will see.

Please 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 cluster configuration to cluster configuration.

If one proceeds to load the cbrd_ia32_intelmpi20_intel_noopt.txt table above into a Microsoft* Excel Spreadsheet, and builds a chart to compare the Bidiagonal Reduction (BRD) Time and the Megaflop values, one might see something like the following (Figure 35):
11.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK Examples with the Intel® Trace Collector and Intel® Trace Analyzer

In the chapter titled Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer, cursory explanations were provided in gathering trace data and opening various analyzer panels for view trace file content. Analysis of the ScaLAPACK examples with Intel® Trace Collector and Intel® Trace Analyzer can also be done easily. This subsection will dwell further on the instrumentation and analysis process. The discussion will focus on how to alter the command-line options for the ScaLAPACK make command so that performance data collection will be possible. Note however, that the user will want to have plenty of disk storage available for collecting trace information on all of the examples because there are approximately 68 ScaLAPACK executables. To instrument the ScaLAPACK examples on an Intel® Itanium® 2-based cluster, one could use the following make command:

```
gmake lib64 F=intel80 mpi=intelmpi20 LIBdir=/opt/intel/ict/2.0/cmkl/8.0.1/lib/64 MPILIB="-L${VT_ROOT}/lib -lVT -ldwarf -lelf -lvtunwind"
```

where the above shell command should appear on one line. For Intel® Pentium® 4, and Intel® Xeon® architectures, a make command to instrument the ScaLAPACK examples might be:

```
gmake lib32 F=intel80 mpi=intelmpi20 LIBdir=/opt/intel/ict/2.0/cmkl/8.0.1/lib/32 MPILIB="-L${VT_ROOT}/lib -lVT -ldwarf -lelf"
```

Finally, for the Intel® EM64T-based architecture, the make command for gathering ScaLAPACK instrumentation data could possibly be:
Recall the instrumentation processes discussed in Figure 1 and Figure 2. The recommended amount of disk storage for collecting trace data on all of the ScALAPACK test cases is about 5 gigabytes. For an executable such as xslu_ipf_gnu_noopt that has been instrumented with the Intel® Trace Collector, a trace file called xslu_ipf_gnu_noopt.stf will be generated. Recalling the protocol that was discussed in the chapter for using Intel® Trace Analyzer, the user can proceed to analyze the content of xslu_ipf_gnu_noopt.stf with the following shell command:

```
traceanalyzer xslu_ipf_gnu_noopt.stf &
```

This command for invoking the Intel® Trace Analyzer will cause the Timeline chart and the Function Profile chart (Figure 36) to be produced as described previously:

![Timeline chart and Function Profile chart](image)

**Figure 36 – Event Timeline Chart and the Function Profile Chart for the executable xslu_ipf_gnu_noopt**

The experiment was conducted on a 4 node cluster. The user should again realize that the contents of a trace file such as xslu_ipf_gnu_noopt.stf will vary from cluster configuration.
to cluster configuration due to factors such as the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster.

Figure 37– The Message Profile Chart (lower right) for the executable \texttt{xslu_ipf_gnu_noopt}

If one proceeds to select Charts->Message Profile, one will generate the Message Profile Chart shown in Figure 37. Subsequently, if Charts->Collective Operations Profile is selected, then the chart shown in Figure 38 will be produced.
Figure 38 – Display of the Collective Operations Profile Chart (lower right) for xslu_ipf_gnu_noopt

The user can zoom in on a particular time interval for the Event Timeline Chart in Figure 38. Clicking on the left-most mouse button and panning across the desired time interval will cause the zoom in function. For example, Figure 39 shows zooming in to the time interval which spans from approximately 49 seconds to approximately 50 seconds. Notice that the number of message lines that are shown in black are significantly reduced with respect to Figure 38.

Figure 39 – Zooming in on the Event Timeline Chart for example xslu_ipf_gnu_noopt

For Figure 39, the send and receive message lines can be “drilled-down-to” so as to view the source by using the context menu as shown in Figure 40.
11.2 Experimenting with the Cluster DFT Software

In the directory path:

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

The user will find a set of subdirectories:

```
blas    cblas  cdftc  dftc  fftc      interval  lapack95  spblas        vmlc  vslc
blas95  cdftf  dftf   fftf  fftw2mkl  lapack    solver    versionquery  vmlf  vslf
```

The two that will be discussed in here are `cdftc` and `cdftf`. These two directories respectively contain C and Fortran programming language examples of the Cluster Discrete Fourier Transform (CDFT). To do experimentation with the contents of these two folders, the following sequence of commands could be used to create instrumented executables and result information.

For the C language version of the CDFT, the Bourne shell commands might look something like:

```
export CWD=${PWD}
export VT_LOGFILE_PREFIX=${CWD}/cdftc_inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
export VT_DETAILED_STATES=5
cd /usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples/cdftc
make lib32
PWD=/usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples/cdftc
mpi=Intel2 comp=intel8 workdir=${VT_LOGFILE_PREFIX}
mpidir=/usr/local/opt/intel/ict/2.0/mpi/2.0 CS="mpiicc -t=log"
```
where `<directory-path-to-mkl>/examples` in the shell command-sequence above is

/usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples

Note that the folder path above will vary depending on where the Intel® Cluster Toolkit was installed on the users system. The change directory command above (i.e. `cd ...`) transfers the Bourne shell session to:

/usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples/cdftc

The `gmake` command for the target `lib32` is one contiguous line that ends with `CS="mpiicc -t=log"`. This command references the makefile variables `lib32`, `PWD`, `mpi`, `comp`, `workdir`, `mpidir`, and `CS`. As mentioned above, the target for the `gmake` command is `lib32`. Two other targets of this type are `lib64` and `libem64t`. The target `lib64` is used for Itanium® 2-based systems. The user can get complete information about this makefile by looking at its contents. There is also a `help` target built within the makefile, and therefore the user can type:

```
gmake help
```

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

```
export VT_LOGFILE_PREFIX=${CWD}/cdftf_inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
export VT_DETAILED_STATES=5
cd /usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples/cdftf
gmake lib32
PWD=/usr/local/opt/intel/ict/2.0/cmkl/8.0.1/examples/cdftf mpi=intel2 comp=intel8 workdir=${VT_LOGFILE_PREFIX}
mpidir=/usr/local/opt/intel/ict/2.0/mpi/2.0 CS="mpiifort -t=log"
```

Within the `cdftc_inst` and `cdftf_inst` folders, there is a folder path called `_results/lib32_intel2mpi_intel8comp` which contains the executables and the output results. The contents of this folder path look something like:

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

and

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

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

```
Program is running on 2 processors

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

70
Configuration parameters:

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

INPUT X, 4 columns

| Row 0: |   1.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 1: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 2: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 3: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 4: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 5: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 6: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 7: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 8: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 9: |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 10: |  0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 11: |  0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 12: |  0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 13: |  0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
| Row 14: |  0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |   0.000, 0.000 |
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.exe.stf
```
```
cdftc_inst/dm_complex_2d_double_ex2.exe.stf
```
```
cdftc_inst/dm_complex_2d_single_ex1.exe.stf
```
```
cdftc_inst/dm_complex_2d_single_ex2.exe.stf
```

and

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

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

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

Figure 41 shows result of simultaneously displaying the Function Profile Chart and the Event Timeline Chart.
12. Experimenting with the Intel® MPI Benchmarks

As mentioned in the general product capabilities section of the Intel® Cluster Toolkit URL, the user should run Intel® MPI Library Benchmarks after installing the Intel® Cluster Tools to verify that the tools have been installed properly. Intel® MPI Benchmarks will also tell the user how fast and efficient messaging is on their Intel® Pentium® 4, or Intel® Xeon®, or Intel® Itanium® 2, or Intel® EM64T-based cluster.

The Intel® MPI Benchmarks directory has 4 subfolders and a ReadMe_first file:

```
doc/ license/ ReadMe_first src/ versions_news/
```

One should make a copy of the installed directory structure .../imb/2.3 for the purpose of making modifications and for running the benchmarks. If one changes their current working directory to that of src, one will find several files with the prefix of make_... where these files serve as include files to Makefile. If for example, one looks at the contents of make_ia64, one will see the default settings of the following makefile variables:
MPI_HOME    = ${MPICH}
MPI_INCLUDE = $(MPI_HOME)/include
LIB_PATH    =
LIBS        =
CC          = ${MPI_HOME}/bin/mpicc
OPTFLAGS    = -O
CLINKER     = $(CC)
LDFLAGS     =
CPPFLAGS    =

At the top of the file called `Makefile`, one will see:

```
##### User configurable options #####
#include make_ia32
#include make_ia64
#include make_sun
#include make_solaris
#include make_dec
#include make_ibm_sp
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

For compilation on Itanium® architectures, for example, the user will need to remove the leading sharp symbol on the phrase `#include make_ia64` listed above. Upon doing this, the beginning lines of `Makefile` should look something like the following:

```
##### User configurable options #####
#include make_ia32
#include make_ia64
#include make_sun
#include make_solaris
#include make_dec
#include make_ibm_sp
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

That directory path for the Intel® MPI Benchmarks could be titled something like .../imb/2.3. From that directory change to the child directory called `src`. In the child directory type the command:

```
gmake all MPI_HOME=/opt/intel/ict/2.0/mpi/2.0 CC=mpiicc
```

The target called `all` will create the executables `IMB-MPI1`, `IMB-EXT`, and `IMB-IO`. Recall that `MPI_HOME` and `CC` are two of many makefile variables defined in the include file `make_ia64`. Thus, for the command line above, the default settings for `MPI_HOME` and `CC` are being overridden via the shell command-line. Remember that specifying a folder path such as:

```
/opt/intel/ict/2.0/mpi/2.0
```

will be dependent on where Intel® MPI Library 2.0 is installed on the user’s cluster.
When compilation completes and the executables are built, one can then proceed to run the applications. For example, if the user wishes to run IMB-MPI1 with 4 MPI processes, the following mpiexec command can be used:

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

The above command-line syntax assumes that a user is running Bourne Shell or Korn Shell. A partial list of the file IMB-MPI1.report might look something like:

```
# Benchmarking PingPong
# ( 2 additional processes waiting in MPI_Barrier)

#---------------------------------------------------
# bytes #repetitions t[usec] Mbytes/sec
#---------------------------------------------------
# 0 1000 67.58 0.00
# 1 1000 71.26 0.01
# 2 1000 71.36 0.03
# 4 1000 71.35 0.05
# 8 1000 71.84 0.11
# 16 1000 72.48 0.21
# 32 1000 611.47 0.05
# 64 1000 612.03 0.10
# 128 1000 617.66 0.20
# 256 1000 628.05 0.39
# 512 1000 650.54 0.75
# 1024 1000 691.77 1.41
# 2048 1000 771.36 2.53
# 4096 1000 1185.50 3.30
# 8192 1000 1830.50 4.27
# 16384 1000 1879.36 8.31
# 32768 1000 3247.16 9.62
# 65536 640 6406.77 9.76
# 131072 320 12256.16 10.20
# 262144 160 23525.30 10.63
# 524288 80 45723.85 10.94
# 1048576 40 89865.24 11.13
# 2097152 20 179383.90 11.15
# 4194304 10 357146.39 11.20
```

The table above can be loaded into Microsoft Excel and the first and third columns can be used to build a two-dimensional Cartesian plot (Figure 42).
Figure 42 - Intel® MPI Benchmark Display for the Ping Pong Application using 4 MPI processes

Please note that the above results are a function of the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster.

The user can also run the executables for IMB-EXT and IMB-IO in a similar fashion:

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

Four MPI processes are used. An example segment of output for IMB-EXT.report might look something like:
# Benchmarking Accumulate

# processes = 4

## MODE: AGGREGATE

<table>
<thead>
<tr>
<th>bytes</th>
<th>#repetitions</th>
<th>t_min[usec]</th>
<th>t_max[usec]</th>
<th>t_avg[usec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>0.35</td>
<td>0.56</td>
<td>0.47</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>6.53</td>
<td>60.37</td>
<td>20.15</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>6.44</td>
<td>54.55</td>
<td>18.57</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>6.47</td>
<td>55.19</td>
<td>18.82</td>
</tr>
<tr>
<td>32</td>
<td>1000</td>
<td>6.63</td>
<td>57.01</td>
<td>19.37</td>
</tr>
<tr>
<td>64</td>
<td>1000</td>
<td>7.50</td>
<td>74.88</td>
<td>47.39</td>
</tr>
<tr>
<td>128</td>
<td>1000</td>
<td>29.60</td>
<td>87.21</td>
<td>56.17</td>
</tr>
<tr>
<td>256</td>
<td>1000</td>
<td>59.94</td>
<td>138.16</td>
<td>95.45</td>
</tr>
<tr>
<td>512</td>
<td>1000</td>
<td>104.60</td>
<td>246.95</td>
<td>171.98</td>
</tr>
<tr>
<td>1024</td>
<td>1000</td>
<td>188.04</td>
<td>475.42</td>
<td>327.74</td>
</tr>
<tr>
<td>2048</td>
<td>1000</td>
<td>376.32</td>
<td>933.19</td>
<td>651.73</td>
</tr>
<tr>
<td>4096</td>
<td>1000</td>
<td>733.76</td>
<td>1848.82</td>
<td>1289.00</td>
</tr>
<tr>
<td>8192</td>
<td>1000</td>
<td>1475.49</td>
<td>3687.01</td>
<td>2576.86</td>
</tr>
<tr>
<td>16384</td>
<td>1000</td>
<td>2953.40</td>
<td>7355.67</td>
<td>5149.97</td>
</tr>
<tr>
<td>32768</td>
<td>1000</td>
<td>5897.64</td>
<td>14692.51</td>
<td>10291.14</td>
</tr>
<tr>
<td>65536</td>
<td>640</td>
<td>11787.41</td>
<td>29376.90</td>
<td>20578.30</td>
</tr>
<tr>
<td>131072</td>
<td>320</td>
<td>23580.53</td>
<td>58731.23</td>
<td>41143.32</td>
</tr>
<tr>
<td>262144</td>
<td>160</td>
<td>47140.90</td>
<td>117496.79</td>
<td>82303.60</td>
</tr>
<tr>
<td>524288</td>
<td>80</td>
<td>94233.52</td>
<td>234842.73</td>
<td>164509.69</td>
</tr>
<tr>
<td>1048576</td>
<td>40</td>
<td>188976.13</td>
<td>469388.68</td>
<td>329079.15</td>
</tr>
<tr>
<td>2097152</td>
<td>20</td>
<td>386117.74</td>
<td>939044.40</td>
<td>662263.24</td>
</tr>
<tr>
<td>4194304</td>
<td>10</td>
<td>787810.78</td>
<td>1892752.00</td>
<td>1340101.52</td>
</tr>
</tbody>
</table>
The partial contents of IMB-IO.report might look something like:

```
# Benchmarking P_Write_Shared
# #processes = 4
#------------------------------------------
# MODE: AGGREGATE
#
#    #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]   Mbytes/sec
#0           50         0.42       178.86        45.13         0.00
#1           50         0.36       213.82        53.73         0.00
#2           50         0.36       303.08        76.04         0.01
#4           50         0.34       301.54        75.65         0.01
#8           50         0.34     32548.78      8204.50         0.00
#16           50       192.30   1231042.60    473953.90         0.00
#32           50       385.14   1231399.70    474027.37         0.00
#64           50       402.20   1231042.60    473953.90         0.00
#128          50       344.62   1231466.86    473561.15         0.00
#256          50       285.10   1231414.20    473478.35         0.00
#512          50       328.22   1231229.82    473761.23         0.00
#1024         50       337.70   1231474.40    473659.83         0.00
#2048         50       294.62   1233236.52    475120.33         0.00
#4096         50       355.26   1231414.20    473561.15         0.00
#8192         50       387.26   1232010.18    473946.82         0.01
#16384        50       573.94   1231196.76    474204.87         0.01
#32768        50       545.36   1232419.32    474679.69         0.03
#65536        50       767.24   1232294.18    474992.15         0.05
#131072       50      1107.88   1235153.40    477082.85         0.10
#262144       50      2628.58   1238987.30    480433.41         0.20
#524288       32      6078.94   1921242.59    740190.35         0.26
#1048576      16      9489.12   3809780.69   11476210.39        0.26
#2097152      8      17663.12   7587811.62   2884954.81         0.26
#4194304      4      38477.72  15139749.23   5745178.37         0.26
#8388608      2      73649.53  30264719.01  11476210.39        0.26
#16777216     1     109252.93  60503872.16  22922102.27        0.26
```

Assuming that the user has an rdma device fabric installed on the cluster, the user can issue the following commands for the three Intel® MPI Benchmark executables so as to access that device fabric:

```
mpiexec -n 4 -env I_MPI_DEVICE rdma ./IMB-MPI1 > IMB-MPI1.rdma.report 2>&1
mpiexec -n 4 -env I_MPI_DEVICE rdma ./IMB-EXT > IMB-EXT.rdma.report 2>&1
mpiexec -n 4 -env I_MPI_DEVICE rdma ./IMB-IO > IMB-IO.rdma.report 2>&1
```

To instrument the Intel® MPI Benchmarks with the Intel® Trace Collector, the make command that was used previously will have to be extended. On Itanium® 2-based systems, recall that the basic link options needed to instrument an MPI application are:

```
-L$(VT_ROOT)/lib -lVT -lvtunwind -ldwarf -lelf -lpthread -lm
```

This options need to be assimilated into the Intel MPI® Benchmarks makefile variables. This can be done as follows:

```
gmake clean
gmake all MPI_HOME=/opt/intel/ict/2.0/mpi/2.0 CC=mpiicc OPTFLAGS="-O -g" LIB_PATH="-L $(VT_ROOT)/lib" LIBS="-lVT -ldwarf -lelf -lvtunwind -lpthread -lm"
```
For Intel® Pentium® 4, Intel® Xeon®, and Intel® EM64T-based architectures, the `make` command for the Intel MPI® Benchmarks might be:

```
gmake all MPI_HOME=/opt/intel/ict/2.0/mpi/2.0 CC=mpiicc OPTFLAGS="-O -g -fp" LIB_PATH="-L ${VT_ROOT}/lib" LIBS="-lVT -ldwarf -lelf -lpthread -lm"
```

where the `LIBS` option `-lvtunwind` is omitted. An alternative to the `gmake` commands for the `all` target listed above is to use the `-trace` option that is associated with the `mpiicc` command. Recall that this option will instruct the compilation driver to link in the Intel® Trace Collector. The simplified `gmake` command for the `all` makefile target might look something like the following:

```
gmake all MPI_HOME=/opt/intel/ict/2.0/mpi/2.0 CC="mpiicc" LDFLAGS="-t" OPTFLAGS="-O -g"
```

The makefile variable called `LDFLAGS` is used to specify the `-trace` option. Note that the `gmake` `all` ... commands listed above are single-line shell commands. Assuming use of the Bourne Shell, the following environment variable settings, and commands could be used to gather instrumentation data.

```
export VT_LOGFILE_PREFIX=${PWD}/inst
export VT_PCTRACE=5
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
mpiexec -n 4 ./IMB-MPI1 1> IMB-MPI1.inst.report 2>&1
mpiexec -n 4 ./IMB-EXT 1> IMB-EXT.inst.report 2>&1
```

As a reminder, the `VT_LOGFILE_PREFIX` environment variable will direct the instrumentation data to a sub-directory called `inst`. Also, the `VT_PCTRACE` environment variable enables runtime program counter tracing. The value of 5 indicates the number of call levels of procedures that will have source location information recorded. Since the unwinding of the call stack each time a function is called can be very costly, the setting of the `VT_PCTRACE` environment variable should be handled with discretion. Additional documentation about Intel® Trace Collector environment variables can be found in the Intel® Trace Collector folder path:

```
```

If one proceeds with the process of issuing the command:

```
traceanalyzer inst/IMB-MPI1.stf &
```

and one proceeds to use the context menu to load the trace file from the Frame Display and generate an Event Timeline Display, one might see something like the following (Figure 43):
Figure 43 – Event Timeline and Function Profile displays for the IMB-MPI1 executable

One can zoom in by clicking on the index button of the mouse while simultaneously panning across the timeline (Figure 43). Figure 44 was produced by doing this panning operating on the left portion of Figure 43.
Figure 44 – Event Timeline and Function Profile displays for the IMB-MPI1 executable where a zoom operation has taken place for the left portion of the Event Timeline.

The black lines signify message events that were sent from one process to another over the course of time. The blue lines describe collective operations. If one positions the cross-hairs of one of these black lines in Figure 44, and left-clicks on the mouse, one will see a pop-up window panel indicating that the message has been identified (Figure 45).
Figure 45 – Generation of the Context Window Panel for the Event Timeline Chart

Figure 46, shows a close-up of the Message panel that was generated by selecting the Details on Message category displayed in Figure 45.
If one clicks on the icon under the Show Source Column in the Details on 1 Message window in Figure 46, a Source View panel will be generated. This is illustrated in Figure 47.
Recall that pressing on the Process 1 button within the panel shown in Figure 47, will cause a toggle to the second process that is involved in the message communication. Completing this process selection will produce the source view for the code fragment that received the message (Figure 48).
Figure 48 – Source View of code fragment that is receiving a message

To make inquiries about the Intel® MPI Benchmarks, visit the URL: http://premier.intel.com.
13. Hardware and Recommendations for Installation

**Processor System Requirements**
Intel® Pentium® 4 processor, or
Intel® Xeon® processor, or
Intel® Itanium® 2 processor, or
Intel® EM64T-based processor

Note that it is assumed that the processors listed above are configured into homogeneous clusters.

**Disk Space Requirements**
10 GBs of disk space (minimum)

**Operating System Requirements**

<table>
<thead>
<tr>
<th>Linux Distributions</th>
<th>IA32</th>
<th>EM64T</th>
<th>Itanium® 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux 3.0</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux 4.0</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server 9</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SUSE Linux 9.0</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SUSE Linux 9.1</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

S = Supported

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

**Intel® Compilers**
For all of the Intel® processor architectures, the version number on the Intel® compilers should be 8.1 or greater.

14. System Administrator Checklist

1. Intel license keys should be place in a common repository for access by the software components of the Intel® Cluster Toolkit. An example license directory path might be:

```
/opt/intel/licenses
```
15. User Checklist

1. Configure the environment variables. For the ~/.bashrc file, an example of setting environment variables might be:

   export INTEL_LICENSE_FILE=/opt/intel/licenses
   . /opt/intel/ict/2.0/ictvars.sh

Alternatively, for ~/.cshrc one needs to set:

   setenv INTEL_LICENSE_FILE /opt/intel/licenses
   source /opt/intel/ict/2.0/ictvars.csh

The above example-checklist items assume that /opt/intel is the default installation area.