



**Intel® Cluster Toolkit Compiler
Edition 3.2 for Microsoft*
Windows* Compute Cluster
Server OS Tutorial**

Intel Corporation

(Revision 20081114)

Table of Contents

1. Disclaimer and Legal Information	3
2. Introduction	5
3. Intel Software Downloads and Installation on Microsoft Windows CCS	8
3.1 Microsoft Windows CCS Installation	9
3.2 Navigating through the Intel® Cluster Tools after the Install has been Completed	10
4. Getting Started with Intel® MPI Library	15
4.1 Launching MPD Daemons	16
4.2 How to Set Up MPD Daemons on Microsoft Windows CCS	17
4.3 Compiling and Linking with Intel® MPI Library on Microsoft Windows CCS	18
4.4 Selecting a Network Fabric	20
4.5 Running an MPI Program Using Intel® MPI Library on Microsoft Windows CCS	21
4.6 Experimenting with Intel® MPI Library on Microsoft Windows CCS	23
4.7 Controlling MPI Process Placement on Microsoft Windows CCS	27
4.8 Using the Automatic Tuning Utility Called mpitune	28
4.8.1 Cluster Specific Tuning	29
4.8.2 MPI Application-Specific Tuning	30
5. Interoperability of Intel® MPI Library with the I_MPI_DEBUG Environment Variable	30
6. Instrumenting MPI Applications with Intel® Trace Analyzer and Collector	31
6.1 Instrumenting the Intel® MPI Library Test Examples	31
6.2 Instrumenting the Intel® MPI Library Test Examples in a Fail-Safe Mode ..	36
6.3 Using itcpin to Instrument an Application	38
6.4 Working with the Intel® Trace Analyzer and Collector Examples	41
6.5 Experimenting with the Message Checking Component of Intel® Trace Collector	44
7. Getting Started in Using the Intel® Math Kernel Library (Intel® MKL)	54
7.1 Experimenting with ScaLAPACK	55
7.2 Experimenting with the Cluster DFT Software	59
8. Using the Intel® MPI Benchmarks	69
9. Uninstalling the Intel® Cluster Toolkit Compiler Edition on Microsoft Windows CCS ..	74
10. Hardware Recommendations for Installation on Microsoft* Windows* CCS ..	76
11. System Administrator Checklist for Microsoft Windows CCS	76
12. User Checklist for Microsoft Windows CCS	77
13. Using the Compiler Switch /Qtcollect	77
14. Using Cluster OpenMP*	87

1.Disclaimer and Legal Information

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

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

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

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

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

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting [Intel's Web Site](#).

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

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

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.

This document as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. 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.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Developers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Improper use of reserved or undefined features or instructions may cause unpredictable behavior or failure in developer's software code when running on an Intel processor. Intel reserves these features or instructions for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from their unauthorized use.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

Document number	Revision number	Description	Revision Date
SKU – 318654-003	20081114	Updated Intel® Cluster Toolkit Compiler Edition 3.2 for Microsoft Windows Compute Cluster Server OS Tutorial to reflect changes and improvements to the software components.	11/14/2008

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

Copyright © 2007-2008, Intel Corporation. All rights reserved.

2. Introduction

In terms of the Intel Cluster Toolkit Compiler Edition software for Windows*, consider references within this document to Microsoft Windows CCS OS and Microsoft* Windows* HPC Server 2008 OS **as interchangeable**. At the time of this writing, the Intel® Cluster Toolkit Compiler Edition 3.2 release on Microsoft Windows Compute Cluster Server (Microsoft Windows CCS*) consists of:

1. Intel® C++ Compiler 11.0 Update 0xx
2. Intel® Fortran Compiler 11.0 Update 0xx
3. Intel® Math Kernel Library 10.1
4. Intel® MPI Benchmarks 3.2
5. Intel® MPI Library 3.2
6. Intel® Trace Analyzer and Collector 7.2

where 0xx might be a value such as 061 and represents a build number.

The software architecture of the Intel Cluster Toolkit Compiler Edition for Microsoft Windows CCS is illustrated in Figure 2.1:

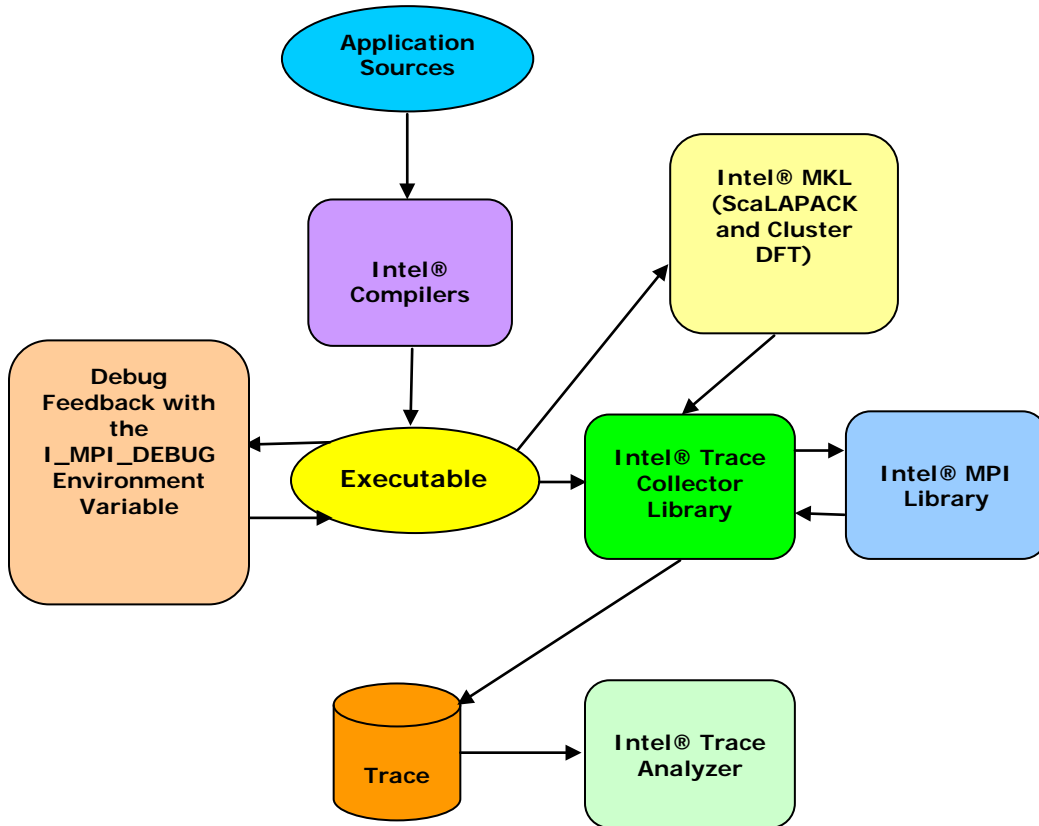


Figure 2.1 – The software architecture for the Intel Cluster Toolkit Compiler Edition for Microsoft Windows CCS

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

Acronym	Definition
ABI	Application Binary Interface – describes the low-level interface an application program and the operating system, between an application and its libraries, or between component parts of an application.
BLACS	Basic Linear Algebra Communication Subprograms – provides a linear algebra oriented message passing interface for distributed memory computing platforms.

BLAS	Basic Linear Algebra Subroutines
DAPL*	Direct Access Program Library - an Application Program Interface (API) for Remote Data Memory Access (RDMA).
DFT	Discrete Fourier Transform
Ethernet	Ethernet is the predominant local area networking technology. It transports data over a variety of electrical or optical media. It transports any of several upper layer protocols via data packet transmissions.
GB	Gigabyte
ICT	Intel® Cluster Toolkit
ICTCE	Intel® Cluster Toolkit Compiler Edition
IMB	Intel® MPI Benchmarks
IP	Internet protocol
ITA or ita	Intel® Trace Analyzer
ITAC or itac	Intel® Trace Analyzer and Collector
ITC or itc	Intel® Trace Collector
MPD	Multi-purpose daemon protocol – a daemon that runs on each node of a cluster. These MPDs configure the nodes of the cluster into a “virtual machine” that is capable of running MPI programs.
MPI	Message Passing Interface - an industry standard, message-passing protocol that typically uses a two-sided send-receive model to transfer messages between processes.
NFS	The Network File System (acronym NFS) is a client/server application that lets a computer user view and optionally store and update file on a remote computer as though they were on the user's own computer. The user's system needs to have an NFS client and the other computer needs the NFS server. Both of them require that you also have TCP/IP installed since the NFS server and client use TCP/IP as the program that sends the files and updates back and forth.
PVM*	Parallel Virtual Machine
RAM	Random Access Memory
RDMA	Remote Direct Memory Access - this capability allows processes executing on one node of a cluster to be able to "directly" access (execute reads or writes against) the memory of processes within the same user job executing on a different node of the cluster.
RDSSM	TCP + shared memory + DAPL* (for SMP clusters)

	connected via RDMA-capable fabrics)
RPM*	Red Hat Package Manager* - a system that eases installation, verification, upgrading, and uninstalling Linux packages.
ScaLAPACK	SCALable LAPACK - an acronym for Scalable Linear Algebra Package or Scalable LAPACK.
shm	Shared memory only (no sockets)
SMP	Symmetric Multi-processor
ssm	TCP + shared memory (for SMP clusters connected via Ethernet)
STF	Structured Trace Format – a trace file format used by the Intel Trace Collector for efficiently recording data, and this trace format is used by the Intel Trace Analyzer for performance analysis.
TCP	Transmission Control Protocol - a session-oriented streaming transport protocol which provides sequencing, error detection and correction, flow control, congestion control and multiplexing.
VML	Vector Math Library
VSL	Vector Statistical Library

3. Intel Software Downloads and Installation on Microsoft Windows CCS

The Intel Cluster Toolkit Compiler Edition installation process on Microsoft Windows CCS is comprised of invoking an installer wizard. The Intel Cluster Toolkit Compiler Edition 3.2 package consists of the following software components which have a folder structure which may look something like the following:

Software Component	Default Installation Directory on Intel® 64 Architecture for Microsoft Windows CCS
Intel C++ Compiler 11.0	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\icc
Intel Fortran Compiler 11.0	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\ifc
Intel MKL 10.1	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\mkl
Intel MPI Library 3.2	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\mpi
Intel MPI Benchmarks 3.2	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\imb
Intel Trace Analyzer and Collector 7.2	C:\Program Files (x86)\intel\ictce\3.2.0.0xx\itac

For the table above, references to 0xx in the folder path represents a build number such as 018. Note that the Intel Cluster Toolkit Compiler Edition installer will automatically make the appropriate selection of binaries, scripts, and text files from its installation archive based on the Intel® processor architecture of the host system where the installation process is initiated. You do not have to worry about selecting the correct software component names for the given Intel® architecture.

Note that you as a user of the Intel Cluster Toolkit Compiler Edition on Microsoft Windows CCS may need assistance from your system administrator in installing the associated software packages on your cluster system, if the installation directory requires system administrative write privileges (e.g. C:\Program Files (x86)\intel on Microsoft Windows CCS). This assumes that your login account does not have administrative capabilities.

3.1 Microsoft Windows CCS Installation

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

`w_ict_<major>.<minor>.<update>.<package_num>.exe`

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

`b_3.2.0.xxx`, where `b` is an acronym for beta

or

`p_3.2.0.xxx`, where `p` is an acronym for production

The `<package_num>` meta-symbol is a string such as 018. This string indicates the package number.

A typical name might look something like:

```
w_ict_p_3.2.0.018.exe
```

For the Intel Cluster Toolkit Compiler Edition installation process on Microsoft Windows CCS, go to the staging folder where the installation package is located and left-click on your mouse to start the installation process.

By default, the global root directory for the installation of the Intel Cluster Toolkit Compiler Edition is:

```
C:\Program Files  
(x86)\intel\ictce\<major>.<minor>.<update>.<package_num>
```

where *<major>*, *<minor>*, *<update>*, and *<package_num>* are integers. An example would be 3.2.0.018.

Within the folder path C:\Program Files (x86)\intel\ictce*<major>*.*<minor>*.*<update>*.*<package_num>* you will find the text files:

```
ictvars.bat
```

and

```
ictcesupport.txt
```

The text file:

```
ictvars.bat
```

contains environment variables for the each of the cluster tools. Sourcing this file will initialize these environment variables and will make it easy to work with the cluster tools as an aggregate. How to automatically initialize these environment variables will be explained in the next subsection.

The file called:

```
ictcesupport.txt
```

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

3.2 Navigating through the Intel® Cluster Tools after the Install has been Completed

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

- 1) Open up a DOS window by clicking on the “Build Environment for Intel Cluster Toolkit Compiler Edition” shortcut that was created by the cluster tools installer, or
- 2) Go to the Start menu and follow a path of All Programs->Intel(R) Software Development Tools->Intel(R) Cluster Toolkit Compiler Edition 3.2->Build Environment for Intel Cluster Toolkit Compiler Edition 3.2 as illustrated in Figure 3.1.

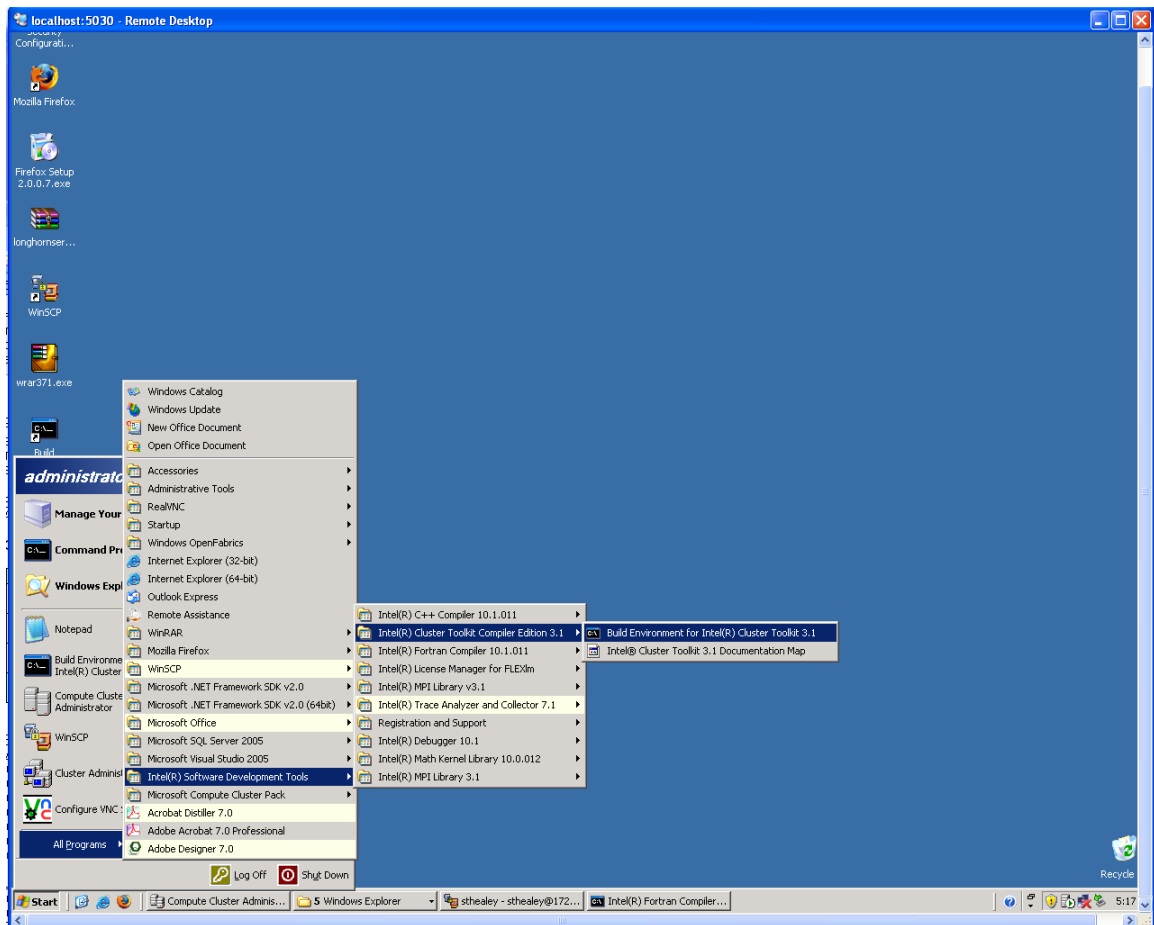


Figure 3.1 – Start->All Programs->Intel(R) Software Development Tools->Intel(R) Cluster Toolkit Compiler Edition 3.2->Build Environment for Intel(R) Cluster Toolkit Compiler Edition 3.2

If you make the selection “Build Environment for Intel Cluster Toolkit Compiler Edition 3.2”, a DOS panel similar to what is shown in Figure 3.2 will be created.

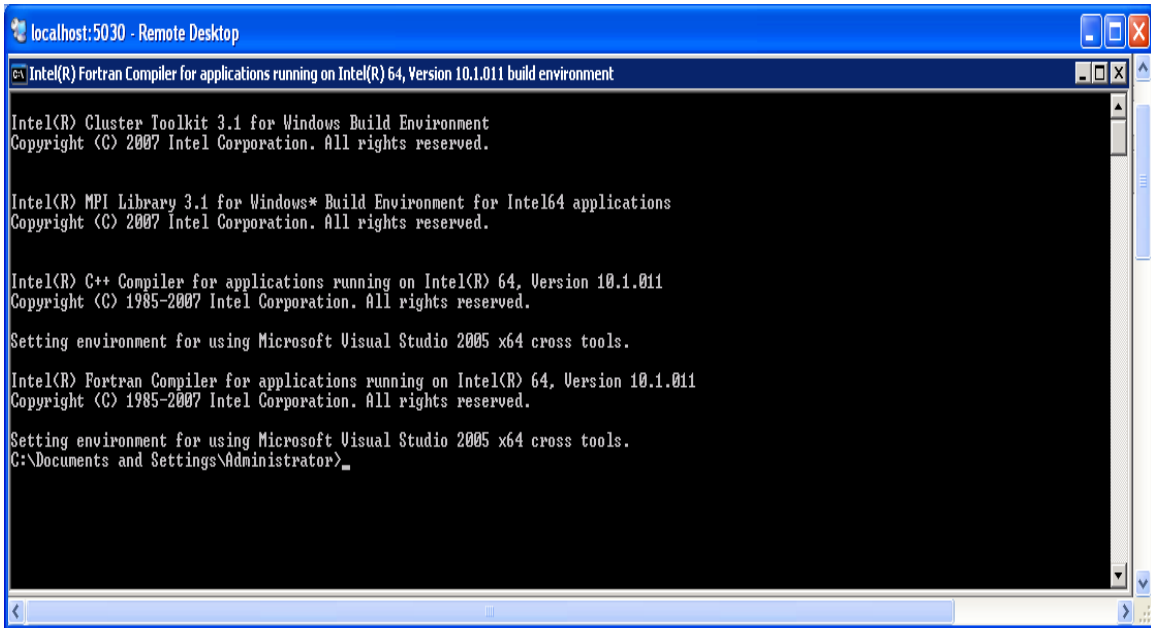


Figure 3.2 – DOS panel display that is created as a result of selecting “Build Environment for Intel Cluster Toolkit Compiler Edition 3.2”

This DOS panel sources the batch script file `ictvars.bat` that was described in subsection 3.1 for sourcing cluster tool environment variables. An example of such an environment that you might find useful is `I_MPI_ROOT`. In regards to Figure 3.2, if you type the DOS command:

```
set I_
```

the `set` command will print all defined variables that begin with the characters `I_`. Figure 3.3 illustrates the result of issuing this command where the environment variable `I_MPI_ROOT` has the value:

```
c:\Program Files (x86)\Intel\ICTCE\3.2.0.018\mpi
```

Note that this value is dependent on where the Intel Cluster Toolkit Compiler Edition package was installed. Referencing the environment variable `%I_MPI_ROOT%` might be useful during compilation phases.

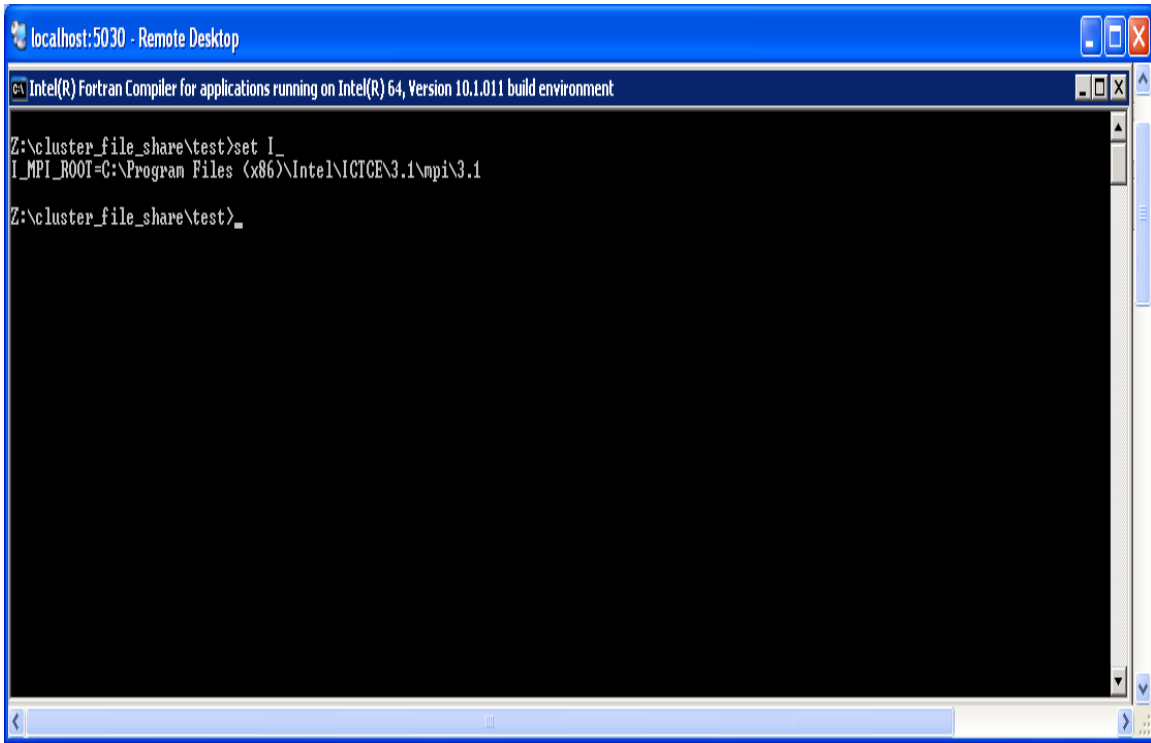


Figure 3.3 – Display of the environment variable I_MPI_ROOT which has a value such as c:\Program Files (x86)\Intel\ICTCE\3.2.0.018\mpi

Use of these environment variables will become clearer starting in Chapter 4. Compilation commands to transform a source code application into an executable can be done from such a DOS panel. Also, use of the `mpiexec` command within a DOS panel can be used to launch the MPI executable onto the nodes of the cluster.

Referring back to Figure 3.1, at the same level as the leaf menu item "Build Environment for Intel Cluster Toolkit Compiler Edition 3.2" is the menu item "Intel® Cluster Toolkit Compiler Edition 3.2 Documentation Map". If you select "Intel® Cluster Toolkit Compiler Edition 3.2 Documentation Map", a panel display something like Figure 3.4 will appear.

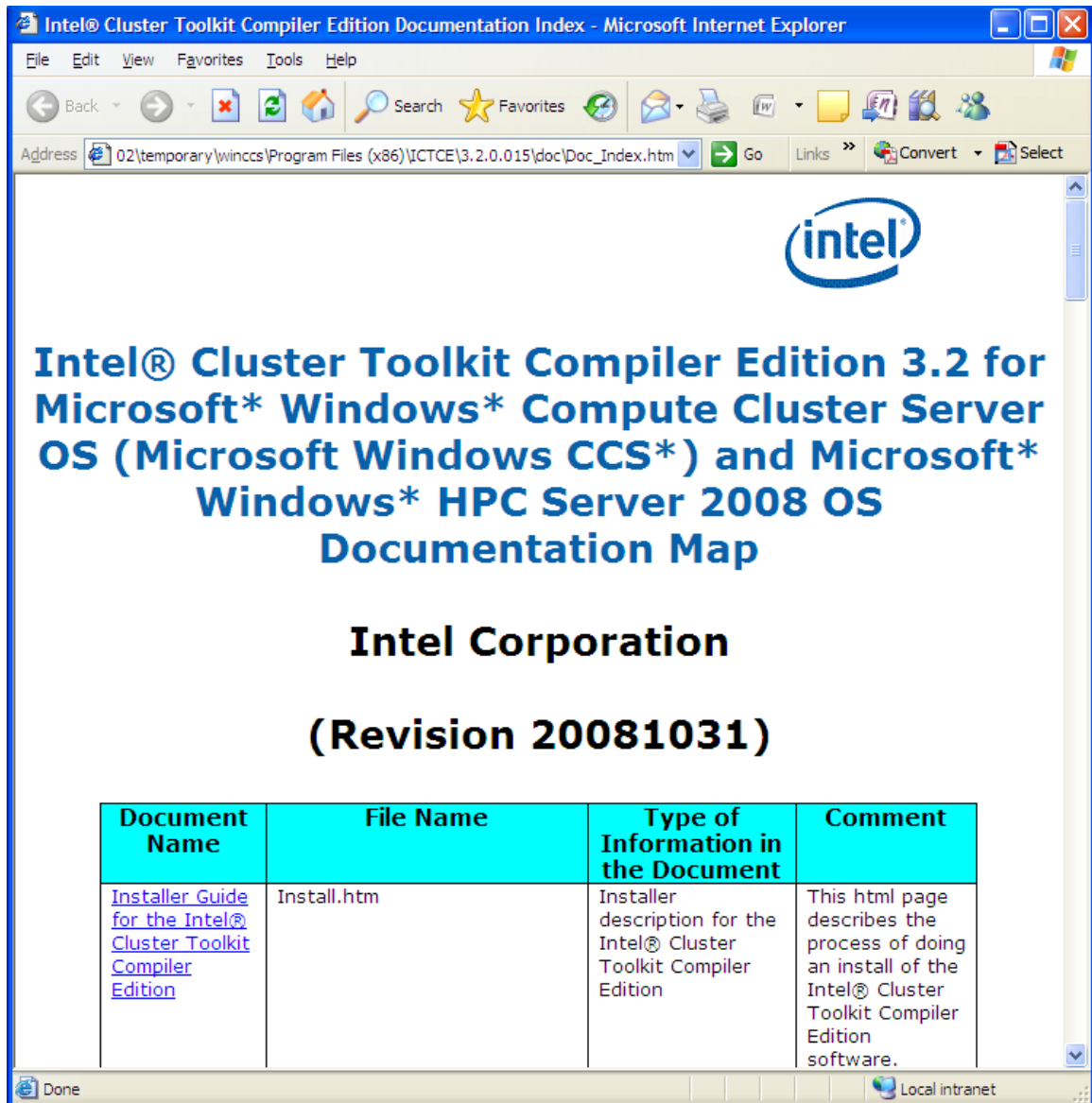


Figure 3.4 – A Rendering of the Intel Cluster Toolkit Compiler Edition Documentation Index File display

The documentation map file can be used to navigate to the FAQ, the release notes, the getting started guide, and an internet accessible [Intel Cluster Toolkit Compiler Edition Tutorial](#) which is this document. This tutorial may have information within it that is more recent than that of the *Intel® Cluster Toolkit Compiler Edition Getting Started Guide*. **Note that for Beta programs involving the Intel Cluster Toolkit Compiler Edition, there is no web based tutorial.**

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

documentation, Intel Trace Analyzer and Collector documentation, Intel MPI Library documentation, Intel MKL documentation, and Intel MPI Benchmarks documentation.

4. Getting Started with Intel® MPI Library

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

The software architecture for Intel MPI Library is described in Figure 4.1. With Intel MPI Library on Linux-based systems, you can choose the best interconnection fabric for running an application on a cluster that is based on IA-32, IA-64, or Intel® 64 architecture. This is done at runtime by setting the `I_MPI_DEVICE` environment variable (See Section 4.4). 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 4.1) as a backup.

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

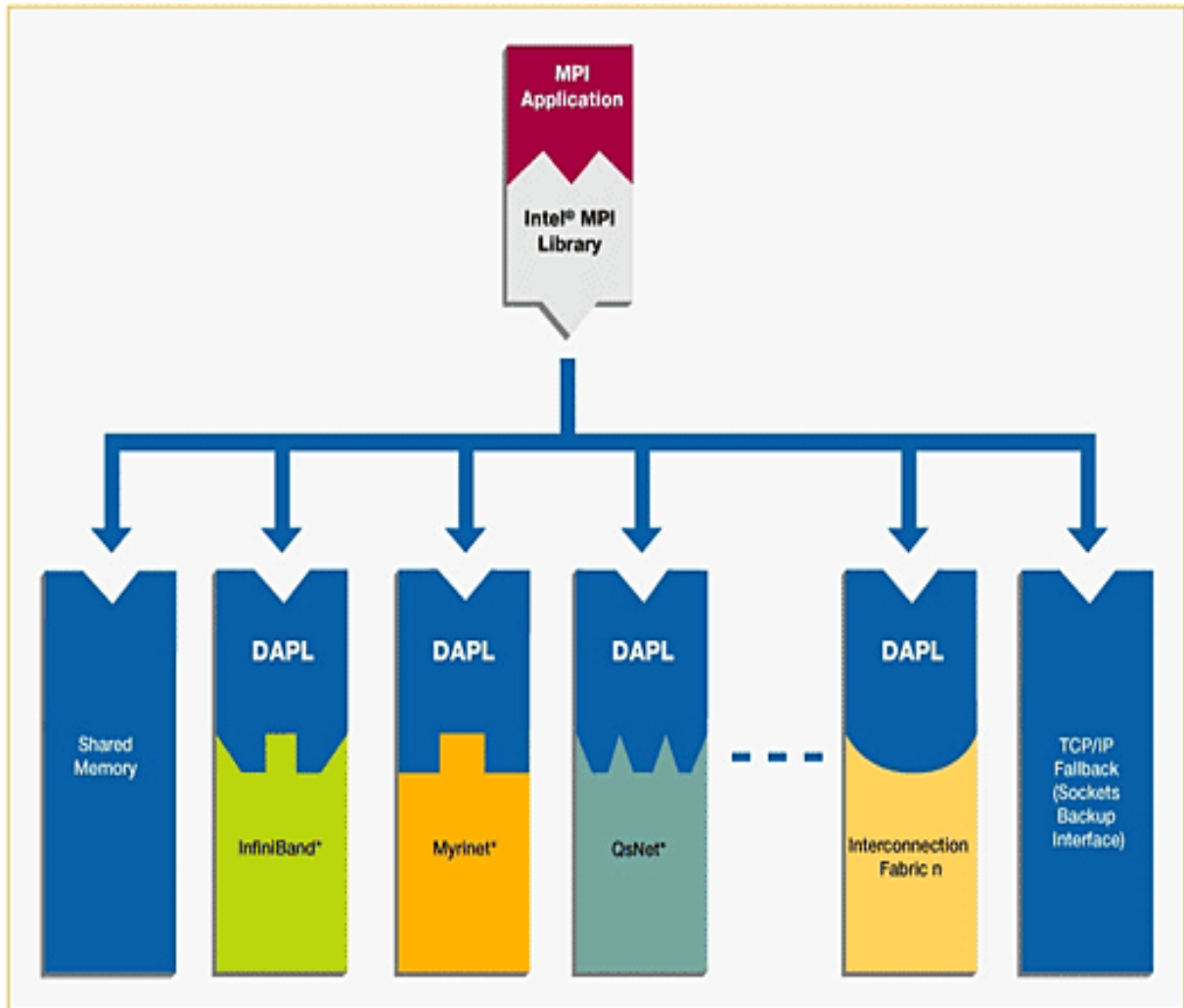


Figure 4.1 – Software architecture of the Intel® MPI Library Interface to Multiple Fast Interconnection Fabrics via shared memory, DAPL (Direct Access Programming Library), and the TCP/IP fallback

4.1 Launching MPD Daemons

The Intel MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. In order to run programs compiled with `mpic1` (or related) commands, you must first set up MPD daemons. It is strongly recommended that you start and maintain your own set of MPD daemons, as opposed to having the system administrator start up the MPD daemons once for use by all users on the system. This setup enhances system security and gives you greater flexibility in controlling your execution environment.

4.2 How to Set Up MPD Daemons on Microsoft Windows CCS

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

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

as demonstrated in Figure 4.2.

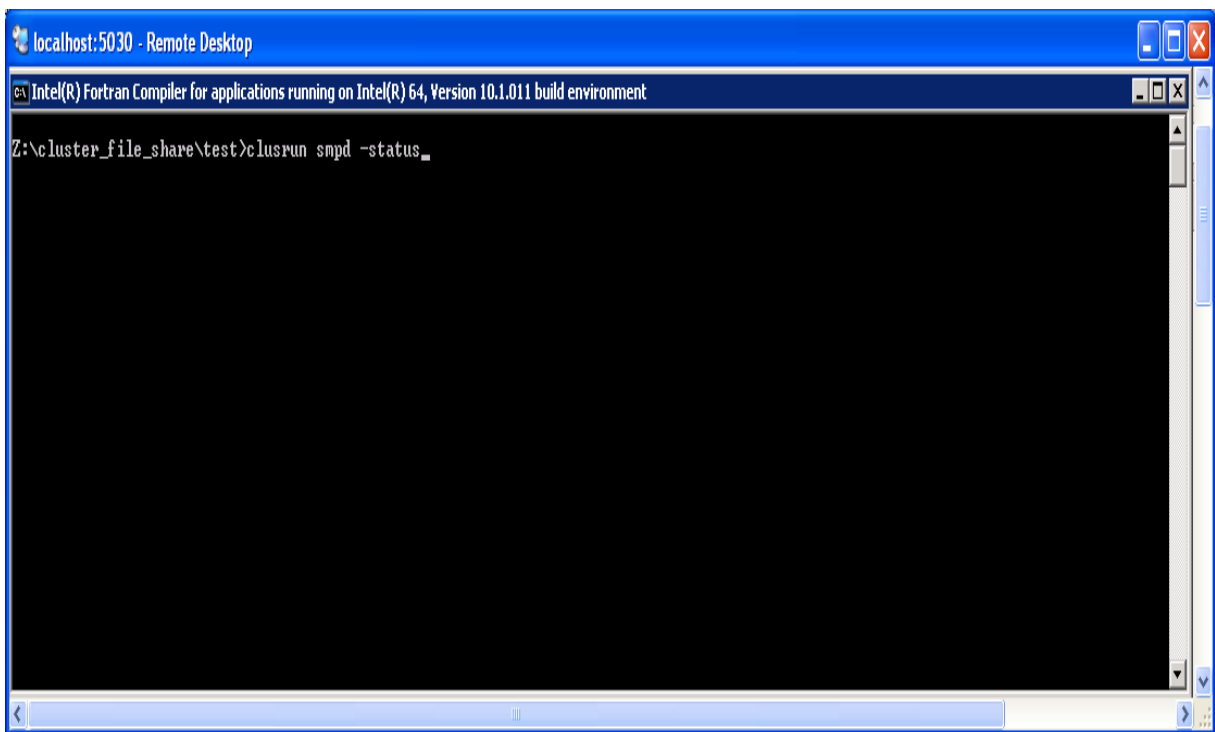


Figure 4.2 – DOS command line syntax for issuing the smpd –status query

For a four node cluster, you might see response that looks something like:

```
-----clusternode01 returns 0-----  
smpd running on clusternode01  
-----clusternode02 returns 0-----  
smpd running on clusternode02  
-----clusternode03 returns 0-----  
smpd running on clusternode03  
-----clusternode04 returns 0-----
```

smpd running on clusternode04

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

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

```
clusrun smpd -uninstall
```

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

```
clusrun smpd -install
```

or

```
clusrun smpd -regserver
```

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

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

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

```
clusrun smpd -remove
```

or

```
clusrun smpd -unregserver
```

or

```
clusrun smpd -uninstall
```

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

```
smpd -help
```

4.3 Compiling and Linking with Intel® MPI Library on Microsoft Windows CCS

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

1. Ensure that the underlying compiler and related software appear in your `PATH`. For example, regarding the Intel 10.1 compilers, execution of the appropriate set-up scripts will do this automatically:

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

and

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

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

```
mpicc <directory-path-to-Intel-MPI-Library>\test\test.c
```

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

Supplier of Core Compiler	MPI Compilation Command	Core Compiler Compilation Command	Compiler Programming Language	Support Application Binary Interface (ABI)
Microsoft Visual C++* Compiler or Intel C++ Compiler 10.1	<code>mpicc</code>	<code>cl.exe</code>	C/C++	32/64 bit
	<code>mpicl</code>	<code>cl.exe</code>	C/C++	32/64 bit
	<code>mpiicc</code>	<code>icl.exe</code>	C/C++	32/64 bit
Intel Fortran Compiler 10.1	<code>mpif77</code>	<code>ifort.exe</code>	Fortran 77 and Fortran 95	32/64 bit
	<code>mpif90</code>	<code>ifort.exe</code>	Fortran 95 and Fortran 95	32/64 bit
	<code>mpifc</code>	<code>ifort.exe</code>	Fortran 95 and Fortran 95	32/64 bit
	<code>mpiifort</code>	<code>ifort.exe</code>	Fortran 77 and Fortran 95	32/64 bit

Remarks

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

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

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

The executable will be called `testc.exe`. This is a result of using the command-line option `/Fe`. The `/I` option references the path to the MPI include files. The library path reference is for the MPI library.

```
mpiexec -machinefile machines.WINDOWS -n 4 testc.exe
```

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

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

If you have a version of the Microsoft Visual C++ Compiler that was *not* packaged with Microsoft* Visual Studio* 2008, type the following command-line:

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

If you have a version of the Microsoft Visual C++ Compiler that was packaged with Microsoft* Visual Studio* 2008, type the following command-line:

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

4.4 Selecting a Network Fabric

Intel MPI Library supports multiple, dynamically selectable network fabric device drivers to support different communication channels between MPI processes. The default communication method uses a built-in TCP (Ethernet, or sockets) device driver. 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	Interconnection Device Fabric Meaning
sock	TCP/Ethernet/sockets (default)
shm	Shared-memory only (no sockets)
ssm	TCP + shared-memory (for SMP clusters connected via Ethernet)
rdma[:<provider>]	InfiniBand*, Myrinet*, etc. (specified via the DAPL (Direct Access Program Library) provider)
rdssm[:<provider>]	TCP + shared-memory + DAPL (for SMP

	clusters connected via RDMA-capable fabrics)
--	----------------------------------------------

where *<provider>* is an optional DAPL* provider name.

4.5 Running an MPI Program Using Intel® MPI Library on Microsoft Windows CCS

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

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

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

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

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

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

```
-wdir <working directory>
-machinefile <filename>
```

You may find these command-line options useful, if the nodes of the cluster are using a file share for example.

If your MPI application is using a network fabric other than the default fabric (sock), use the `-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 .\myprog.exe
```

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

```
mpiexec -n <# of processes> -env I_MPI_DEVICE rdma[:<provider>] .\myprog.exe
```

where `[:<provider>]` is an optional meta-symbol which may need to be filled in by you with correct RDMA information. Note that the brackets imply an optional feature and are not part of the actual `rdma` syntax. Any supported device can be selected.

See the section titled *Selecting a Network Fabric* 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`.

To generate a `machines.Windows` text file to be used with the `-machinefile` command-line option, the following methodology might be useful.

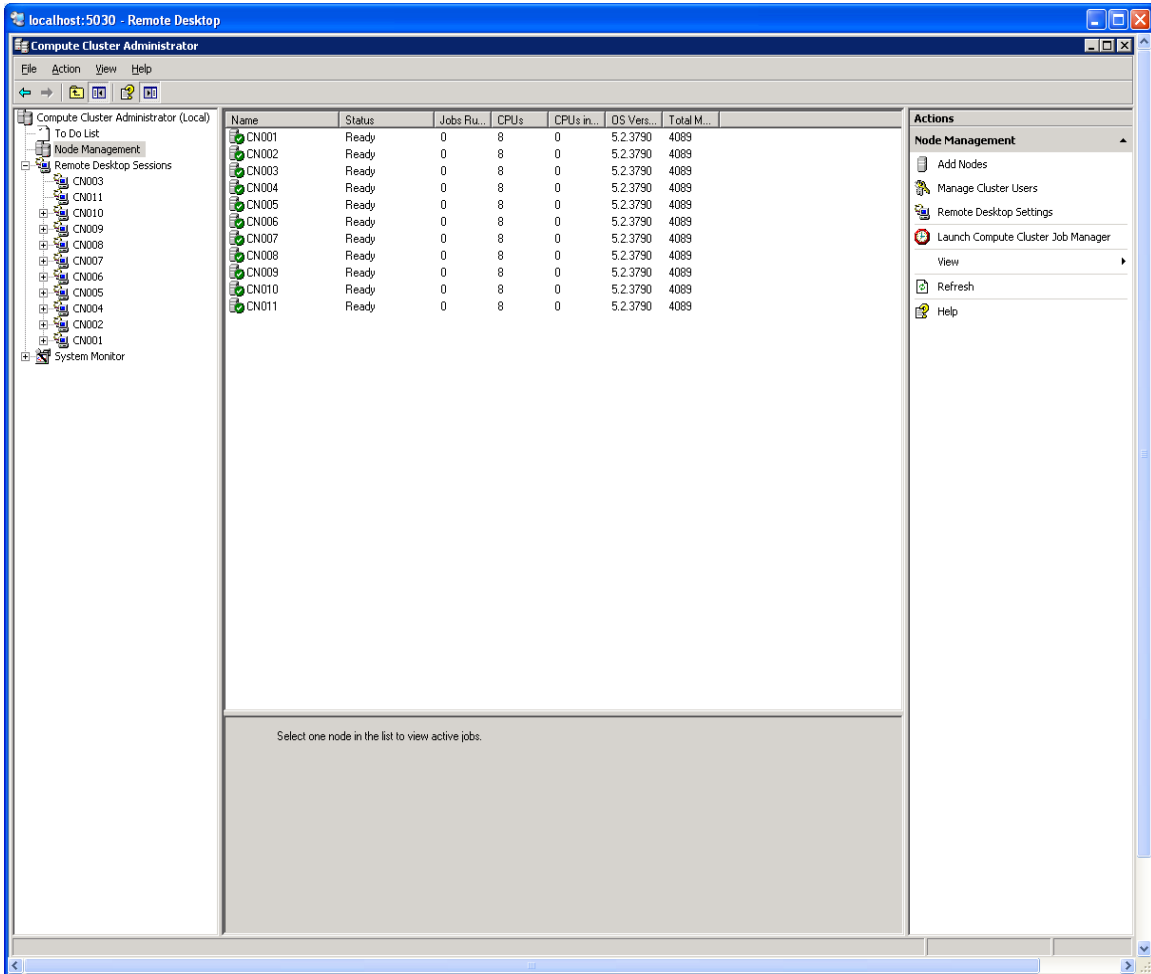


Figure 4.3 – Compute Cluster Administrator display panel within Microsoft Windows Compute Cluster Server* (CCS)

If you select the Remote Desktop Sessions link in the left sub-panel then proceed to click in the right sub-panel of Figure 4.4, you can produce a text file of node names (1 per line).

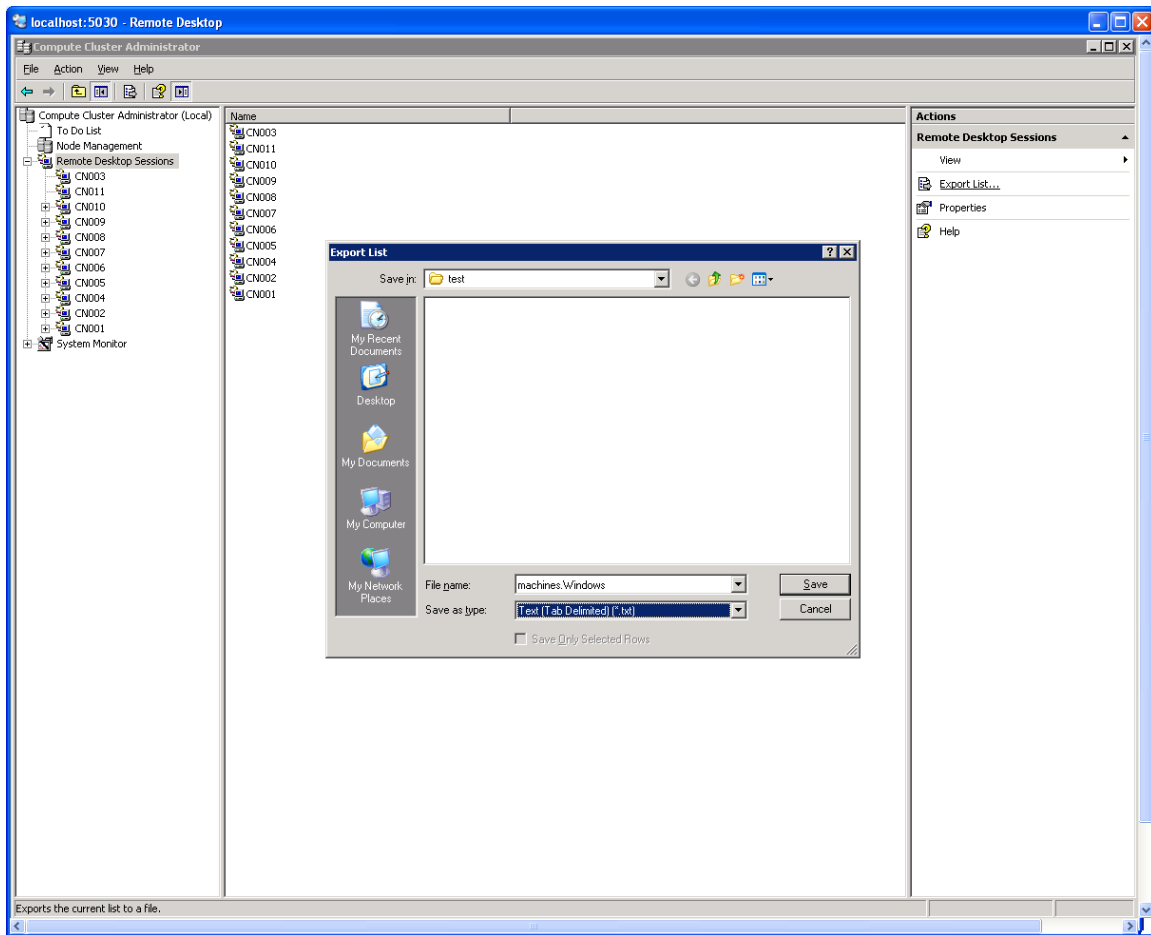


Figure 4.4 – Creation of the `machines.Windows` file for the `mpirun` command-line option `-machinefile`

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

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

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

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

4.6 Experimenting with Intel® MPI Library on Microsoft Windows CCS

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.

Recall that in section 4.2 that the command for launching multi-purpose daemons on Microsoft Windows is called "smpd", which is an acronym for simple multi-purpose daemons. Also note that when Intel MPI Library is installed on a cluster, the smpd service is automatically started. In part 4.2 it was mentioned that you could type the command:

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

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

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

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

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

```
test_intel_mpi
```

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

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

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

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

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

If you have a version of the Microsoft Visual C++ Compiler that was *not* packaged with Microsoft* Visual Studio* 2008, type the following respective command-lines for the C and C++ test applications:

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

and

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

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

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

and

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

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

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

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

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

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

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

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

and for the Microsoft Visual C++ executables:

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

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

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

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

```
z:\cluster_file_share
```

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

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

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

```
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2 testf.exe
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2 testf90.exe
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2 testc.exe
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2 testcpp.exe
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2 testc_vc.exe
mpiexec -machinefile machines.Windows -env I_MPI_DEVICE rdma -n 2
testcpp_vc.exe
```

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
```

4.7 Controlling MPI Process Placement on Microsoft Windows CCS

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

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

```
type machines.Windows
```

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, you might issue the following commands:

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

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

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

In general, if there are i nodes in the cluster and each node is j -way SMP system, 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 you would have to fill in appropriate host names for `<nodename-1>` through `<nodename-i>` with respect to your cluster system. For a complete discussion on how to control process placement through the `mpiexec` command, see the *Local Options*

section of the *Intel MPI Library Reference Manual* located in `<directory-path-to-Intel-MPI-Library>\doc\Reference_Manual.pdf`.

4.8 Using the Automatic Tuning Utility Called `mpitune`

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

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

```
mpitune -f machines.Linux -o .\ --app mpiexec -n 4 .\testc
```

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

Command-line Option	Semantic Meaning
<code>-h --help</code>	Display a help message
<code>-V --version</code>	Display the Intel® MPI Library version information
<code>-e <envfile> --env <envfile></code>	Specify path/name of the file with hardware and software environment information. <code><installdir>/etc/env.xml</code> or <code><installdir>/etc64/env.xml</code> are used by default
<code>-r <rulesfile> --rules <rulesfile></code>	Specify path/name of the file with the tuning rules. <code><installdir>/etc/rules.xml</code> or <code><installdir>/etc64/rules.xml</code> are used by default
<code>-f <hostsfile> --file <hostsfile></code>	Specify path/name of the file that has a list of machine names to be used in the tuning process. <code>%CD%\mpd.hosts</code> is used by default. If the host file list is omitted, availability of a suitable MPD ring is expected <code>-w <workdir> --wdir <workdir></code>
<code>-w <workdir> --wdir <workdir></code>	Specify the location of the benchmarking program(s). <code><installdir>/bin</code> or <code><installdir>/bin64</code> are used by default <code>-o <outputdir> --outdir <outputdir></code>
<code>-o <outputdir> --outdir <outputdir></code>	Specify the output directory for the <code>mpiexec</code> configuration files. <code><installdir>/etc</code> or <code><installdir>/etc64</code> are used by default

	<i>Intel® MPI Library for Windows* OS Reference Manual</i> Document number: 315399-004 25
-d --debug	Print debug information
-i <count> --iterations <count>	Define how many times to run each tuning step. One iteration is the default value. Higher iteration counts increase tuning time but may also increase the accuracy of the results
-v --verbose	Print detailed information on the progress of the tuning process
-s --strict	Stop execution if any of the test units failed
-c <name> --configfile <name>	Set the name of the tuned configuration file. The default name for the application tuning is <code>app.conf</code> . A configuration name for the cluster-specific tuning is selected automatically. A configuration file will be stored in <code><outputdir></code>
--silent	Run tuner silently, dumping output of a single iteration at the end
--logs	Save application output at each iteration for debugging reasons
--app<application command line>	Switch on application tuning mode. Default mode is the cluster specific tuning. The rest of the arguments list beyond the --app flag is treated as the application command line to be used for tuning

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

4.8.1 Cluster Specific Tuning

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

```
mpitune -f machines.LINUX -o .\
```

could be used, where `machines.LINUX` contains a list of the nodes in the cluster. Completion of this command may take some time. The `mpitune` utility will generate a configuration file that might have a name such as `mpiexec_shm_nn_1_np_4_ppn_4.conf`. You can then proceed to run the `mpiexec` command on an application using the `-tune` option. For example, the `mpiexec`

command-line syntax for the `testc` executable might look something like the following:

```
mpiexec -tune -n 4 .\testc
```

4.8.2 MPI Application-Specific Tuning

The `mpitune` invocation:

```
mpitune -f machines.Linux -o .\ --app mpiexec -n 4 .\testf90
```

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

```
mpiexec -tune app.config -n 4 .\testf90
```

where the `mpiexec` command will load the configuration options recorded in `app.config`.

You might want to use `mpitune` utility on each of the test applications `testc`, `testf`, and `testf90`. For a complete discussion on how to use the `mpitune` utility, see the *Tuning Reference* 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>.

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

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

```
I_MPI_DEBUG=<level>
```

where `<level>` can have the values:

Value	Debug Level Semantics
Not set	Print no debugging information
1	Print warnings if specified <code>I_MPI_DEVICE</code> could not be used
2	Confirm which <code>I_MPI_DEVICE</code> was used
> 2	Add extra levels of debugging information

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

```
mpiexec -n <# of processes> -env I_MPI_DEBUG +2 my_prog.exe
```

produces output debug messages in the following format:

```
I_MPI: [rank#pid@hostname]Debug message
```

You can also compile the MPI the application with the `/zi` or `/z7` compiler options.

6. Instrumenting MPI Applications with Intel® Trace Analyzer and Collector

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

6.1 Instrumenting the Intel® MPI Library Test Examples

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

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

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

```
set VT_LOGFILE_PREFIX=test_inst
```

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

```
mkdir %VT_LOGFILE_PREFIX%
```

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

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

and

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

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

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

and

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

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

Alternatively, to compile and instrument the respective C and C++ files `test.c` and `test.cpp` using a Microsoft* Visual Studio* C++ compiler that was *not* packaged with Microsoft* Visual Studio* 2008, you can issue the DOS commands:

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

and

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

Note that when compiling and linking with a Microsoft Visual Studio C++ compiler that was *not* packaged with Microsoft* Visual Studio* 2008, the library `bufferoverflowu.lib` has been added as demonstrated above for the C and C++ test cases.

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

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

and

```
cl /Fetestcpp_vc /I"%I_MPI_ROOT%\em64t\include test.cpp /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impicxx.lib impi.lib
/NODEFAULTLIB:LIBCMD.lib
```

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

```
testc.exe
testcpp.exe
testcpp_vc.exe
testc_vc.exe
testf.exe
testf90.exe
```

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

```
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_vc
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_vc
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf90
```

For the executables above, 4 MPI processes are created via the `mpiexec` command. These `mpiexec` commands will produce the STF files:

```
testc.stf testcpp.stf testcpp_vc.stf testc_vc.stf testf.stf testf90.stf
```

within the directory `test_inst`.

Issuing the `traceanalyzer` command on the STF file `test_inst\testcpp_vc.stf` as follows:

```
traceanalyzer test_inst\testcpp_vc.stf
```

will generate a profile panel which looks something like the following:

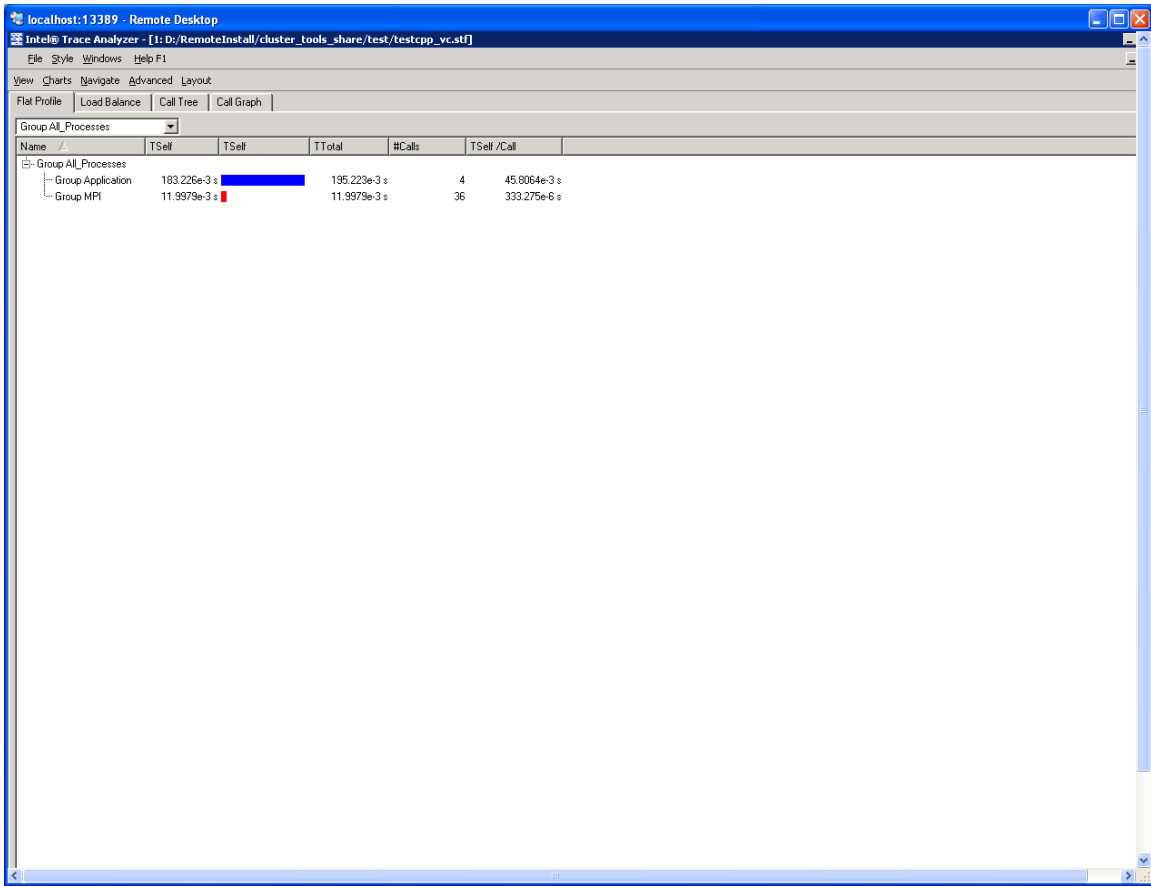


Figure 6.1 – The Profile display for testcpp_vc.stf

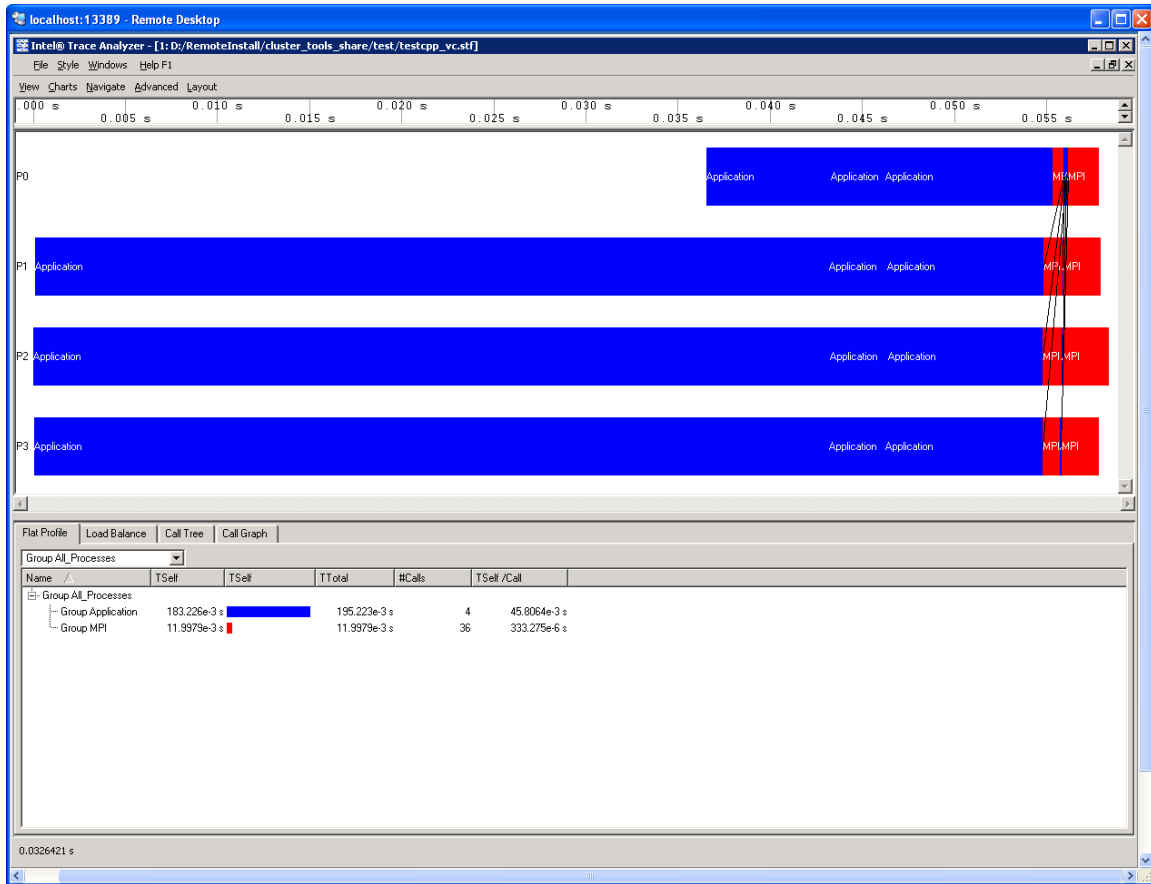


Figure 6.2 – The Profile and Timeline display for testcpp_vc.stf

An alternative to the above `mpirun` commands is to create a trace collector configuration file such as `vtconfig.txt` which could have the contents, beginning in column 1, of:

```
logfile-prefix test_inst
```

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

```
mpirun -n 4 -env VT_CONFIG vtconfig.txt testc
mpirun -n 4 -env VT_CONFIG vtconfig.txt testcpp
mpirun -n 4 -env VT_CONFIG vtconfig.txt testcpp_vc
mpirun -n 4 -env VT_CONFIG vtconfig.txt testc_vc
mpirun -n 4 -env VT_CONFIG vtconfig.txt testf
```

```
mpiexec -n 4 -env VT_CONFIG vtconfig.txt testf90
```

6.2 Instrumenting the Intel® MPI Library Test Examples in a Fail-Safe Mode

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

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

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

and

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

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

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

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

and

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

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

Alternatively, to compile and instrument the respective C and C++ files `test.c` and `test.cpp` using a Microsoft* Visual Studio* C++ compiler that was *not* packaged with Microsoft* Visual Studio* 2008, you can issue the DOS commands:

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

and

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

Again note that when compiling and linking with a Microsoft Visual Studio C++ compiler that was *not* packaged with Microsoft* Visual Studio* 2008, the library `bufferoverflowu.lib` has been added as demonstrated above for the C and C++ test cases.

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

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

and

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

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

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

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

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

In case of execution failure by the application, the fail-safe library freezes all MPI processes and then writes out the trace file. Figure 6.3 shows an Intel Trace Analyzer display for `test.c`.

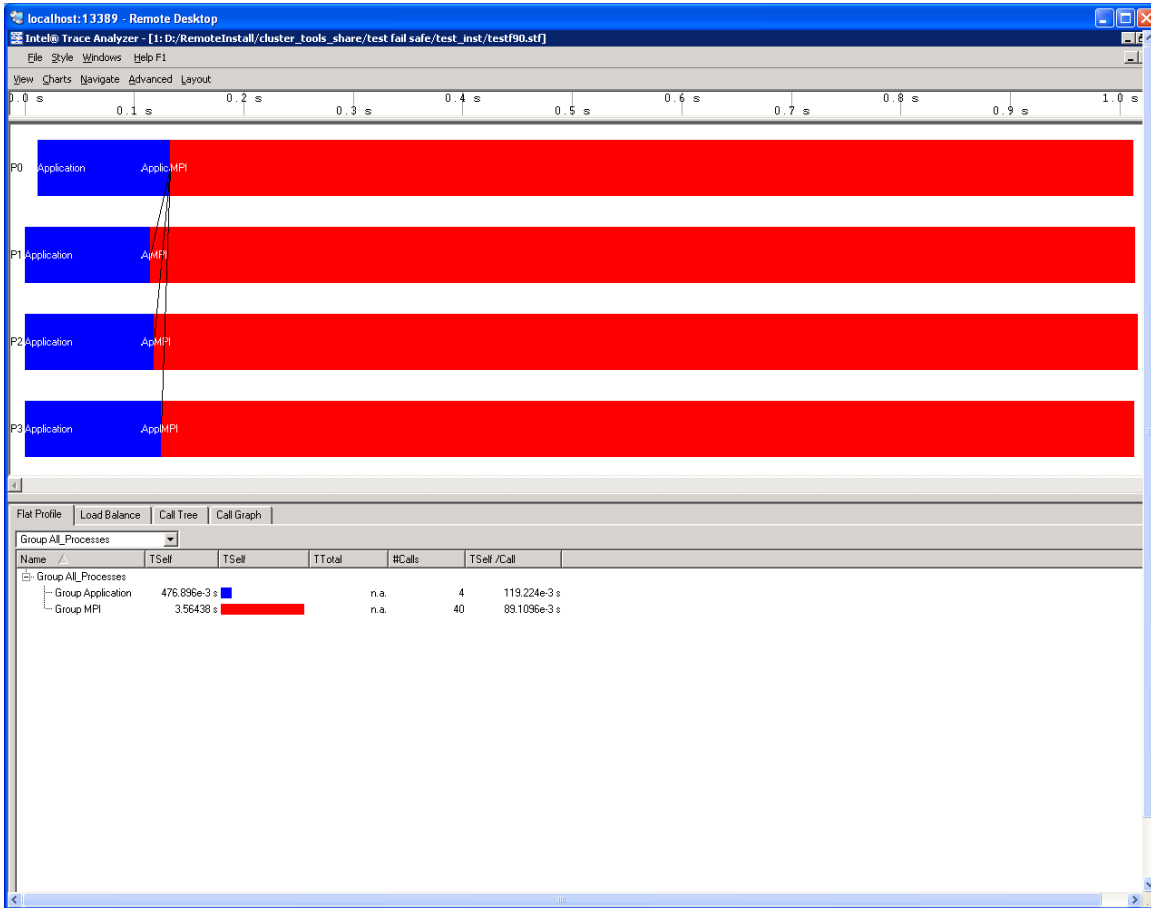


Figure 6.3 – Intel Trace Analyzer display of Fail-Safe Trace Collection by Intel Trace Collector

Complete user documentation regarding `vtfs.lib` for the Intel Trace Collector can be found within the file:

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

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

6.3 Using `itcpin` to Instrument an Application

The `itcpin` utility is a binary instrumentation tool that comes with Intel Trace Analyzer and Collector. The Intel® architecture for `itcpin` on Microsoft Windows must be Intel® 64.

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

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

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

The `<ITC options>` that will be used here are:

```
--run (off)
```

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

```
--insert
```

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

To obtain a list of all of the `itcpin` options simply type:

```
itcpin --help
```

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

http://rac.uits.iu.edu/hpc/mpi_tutorial/s2_computing_pi_parallel.html

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

```
icl /Fepi /I"%I_MPI_ROOT%" \em64t\include pi.c /link
/LIBPATH:"%VT_LIB_DIR%" VT.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
set VT_LOGFILE_PREFIX=itcpin_inst
```

```
rmdir /S /Q %VT_LOGFILE_PREFIX%
```

```
mkdir %VT_LOGFILE_PREFIX%
```

```
mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL
"%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env
```

```
VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin
--run --insert VT -- pi.exe 1000000
```

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

```
-env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env
VT_LOGFILE_FORMAT STF -env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX
"%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON"
```

An explanation for these *instrumentation* environment variables can be found in the Intel Trace Collector Users' Guide under the search topic "ITC Configuration".

The DOS shell commands above could be packaged into a DOS batch script. The value of 1,000,000 after the executable called `pi.exe` indicates the number of intervals that will be used in the calculation of "pi".

The output from the above sequence or DOS Shell commands looks something like the following:

```
[0] Intel(R) Trace Collector INFO: Writing tracefile pi.stf in
C:\WINDOWS\Temp\cluster_share\itcpin\itcpin_inst
[0] Intel(R) Trace Collector INFO: Writing tracefile pi.stf in
C:\WINDOWS\Temp\cluster_share\itcpin\itcpin_inst
The computed value of the integral is 3.141592653589903
The exact value of the integral is    3.141592653589793
```

Figure 6.4 shows the timeline and function panel displays that were generated from the instrumentation data that was stored into the directory `itcpin_inst` as indicated by the environment variable `VT_LOGFILE_PREFIX`. The command that initiated the Intel Trace Analyzer with respect to the current directory was:

```
traceanalyzer itcpin_inst\pi.exe.stf &
```

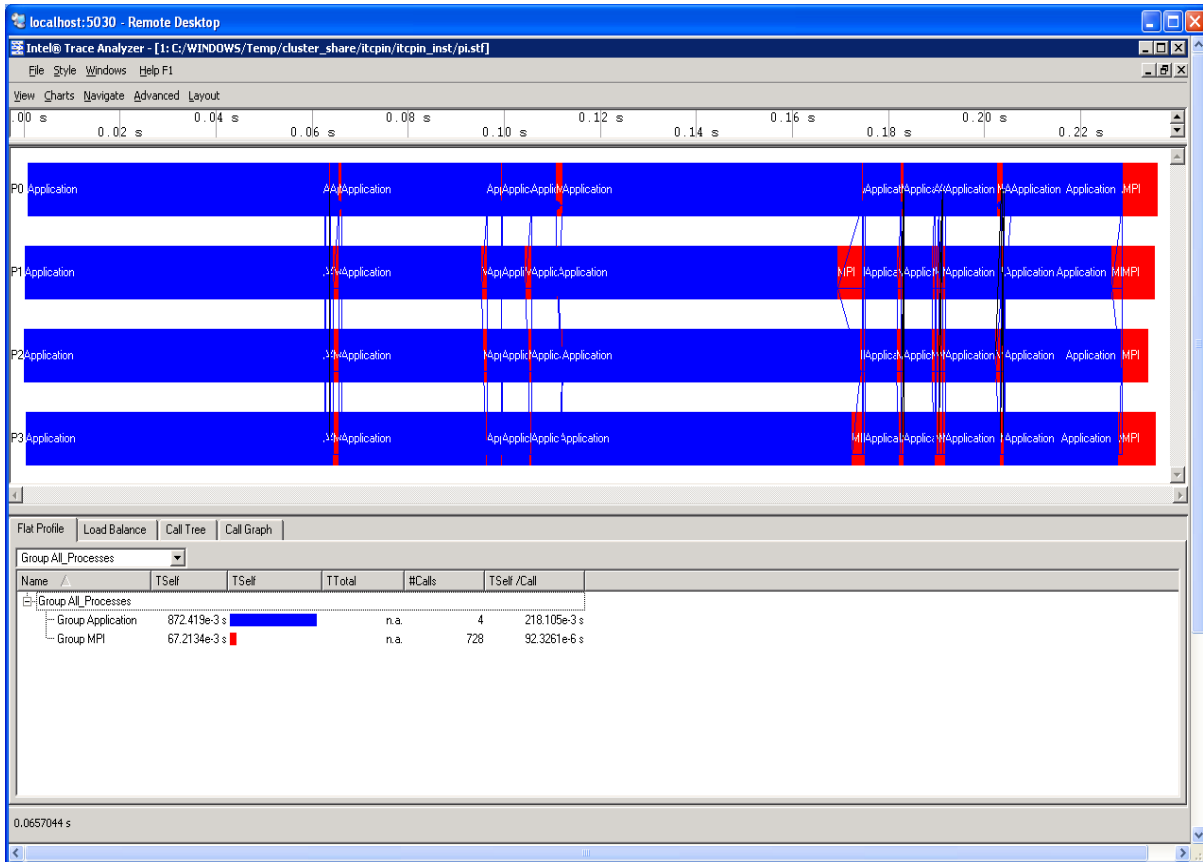


Figure 6.4 – Intel Trace Analyzer display of the “pi” integration application that has been binary instrumented with itcpin

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

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

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

6.4 Working with the Intel® Trace Analyzer and Collector Examples

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

`C:\Program Files (x86)\Intel\ictce\3.2.0.018\itac\examples`

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

```
nmake distclean
```

```
nmake all MPIDIR="c:\Program Files (x86)\Intel\ictce\3.2.0.018\MPI\em64t"
```

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

```
mpiconstants.exe  
vncallpair.exe  
vncallpairc.exe  
vnjacobic.exe  
vnjacobicif.exe  
vtallpair.exe  
vtallpairc.exe  
vtcounterscopec.exe  
vtjacobic.exe  
vtjacobicif.exe  
vttimertest.exe
```

of which the following STF files are created:

```
timertest.stf  
vtallpair.stf  
vtallpairc.stf  
vtcounterscopec.stf  
vtjacobic.stf  
vtjacobicif.stf
```

If one invokes Intel Trace Analyzer with the command:

```
traceanalyzer vtjacobic.stf
```

the following display panel will appear (Figure 6.5):

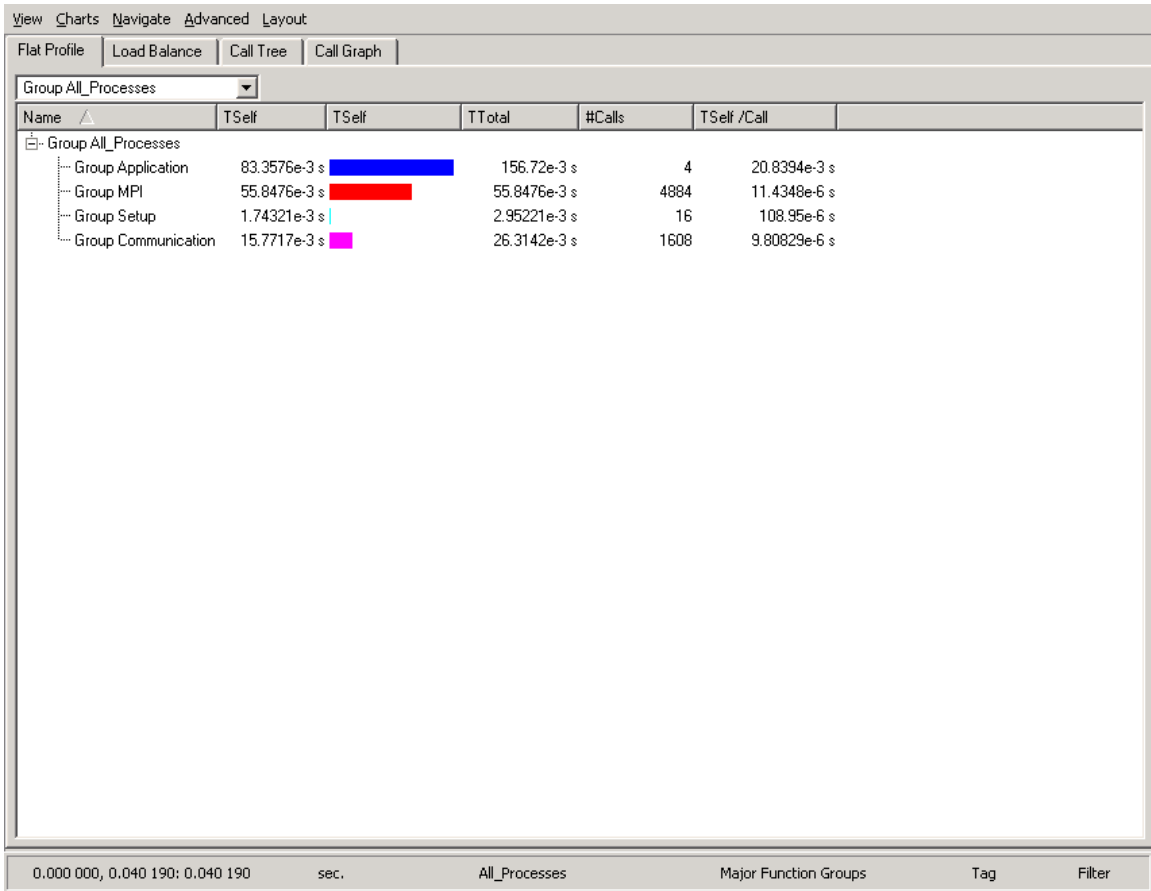


Figure 6.5 - Intel Trace Analyzer Display for vtjacobic.stf

Figure 6.6 shows the Event Timeline display which results when following the menu path Charts->Event Timeline within Figure 6.5.

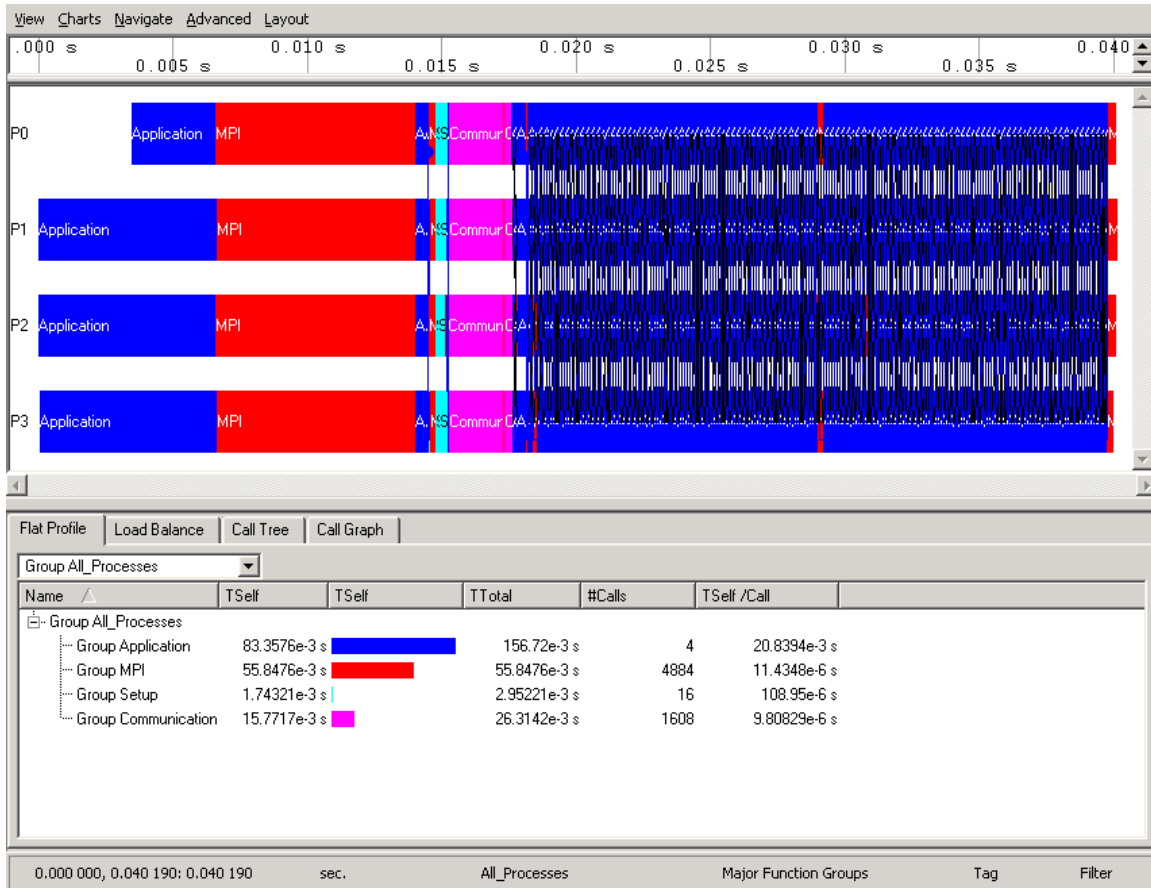


Figure 6.6 - Intel Trace Analyzer Display for vtjacobic.stf using Charts->Event Timeline

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

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

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

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

`VT_DEADLOCK_WARNING <delay>` where `<delay>` is a time value. The default value is 5 minutes and the notation for the meta-symbol `<delay>` could be 5m. If on average

the MPI processes are stuck in their last MPI call for more than this threshold, then a GLOBAL:DEADLOCK:NO PROGRESS warning is generated. This is a sign of a load imbalance or a deadlock which cannot be detected because at least one process polls for progress instead of blocking inside an MPI call.

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

Complete user documentation regarding message checking for the Intel Trace Collector can be found within the file:

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

The chapter title is called "Correctness Checking".

At the URL:

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

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

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

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

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

```
cl /D_CRT_SECURE_NO_DEPRECATED /Fedeadlock_vc
/I"%I_MPI_ROOT%\em64t\include /Zi deadlock.c /link /stack:8000000
/LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

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

```
cl /D_CRT_SECURE_NO_DEPRECATED /Fedeadlock
/I"%I_MPI_ROOT%\em64t\include /Zi deadlock.c /link /stack:8000000
/LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib
```

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

Also recall that the prior to issuing an `mpiexec` command, a scratch folder for trace information called `test_inst`, can be created with the `VT_LOGFILE_PREFIX` environment variable by using the following process:

```
set VT_LOGFILE_PREFIX=test_inst
```

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

```
mkdir %VT_LOGFILE_PREFIX%
```

When issuing the `mpiexec` command with settings for the `VT_DEADLOCK_TIMEOUT`, `VT_DEADLOCK_WARNING`, and `VT_CHECK_TRACING` environment variables:

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

diagnostics that look something like the following are generated:

```
...  
[0] ERROR: no progress observed in any process for over 0:29 minutes, aborting  
application  
[0] WARNING: starting emergency trace file writing  
  
[0] ERROR: GLOBAL:DEADLOCK:HARD: fatal error  
[0] ERROR: Application aborted because no progress was observed for over 0:29 minutes,  
[0] ERROR: check for real deadlock (cycle of processes waiting for data) or  
[0] ERROR: potential deadlock (processes sending data to each other and getting  
blocked  
[0] ERROR: because the MPI might wait for the corresponding receive).  
[0] ERROR: [0] no progress observed for over 0:29 minutes, process is currently in MPI  
call:  
[0] ERROR: MPI_Recv(*buf=00000000001C5680, count=800000, datatype=MPI_INT,  
source=1, tag=999, comm=MPI_COMM_WORLD, *status=00000000007DFE80)  
[0] ERROR: main (Z:\message_checking\deadlock.c:49)  
[0] ERROR: __tmainCRTStartup  
(f:\dd\vctools\crt_bld\self_64_amd64\crt\src\crt0.c:266)  
[0] ERROR: BaseThreadInitThunk (kernel32)  
[0] ERROR: RtlUserThreadStart (ntdll)  
[0] ERROR: (  
[0] ERROR: [1] no progress observed for over 0:29 minutes, process is currently in MPI  
call:  
[0] ERROR: MPI_Recv(*buf=00000000001C5680, count=800000, datatype=MPI_INT,  
source=0, tag=999, comm=MPI_COMM_WORLD, *status=00000000007DFE80)  
[0] ERROR: main (Z:\message_checking\deadlock.c:49)  
[0] ERROR: __tmainCRTStartup  
(f:\dd\vctools\crt_bld\self_64_amd64\crt\src\crt0.c:266)  
[0] ERROR: BaseThreadInitThunk (kernel32)  
[0] ERROR: RtlUserThreadStart (ntdll)  
[0] ERROR: (  
[0] INFO: Writing tracefile deadlock.stf in Z:\message_checking\test_inst  
  
[0] INFO: GLOBAL:DEADLOCK:HARD: found 1 time (1 error + 0 warnings), 0 reports were  
suppressed  
[0] INFO: Found 1 problem (1 error + 0 warnings), 0 reports were suppressed.
```

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

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

One can use the Intel® Trace Analyzer to view the deadlock problem that was reported in the output listing above. Here is what the trace information might look like (Figure 6.7):

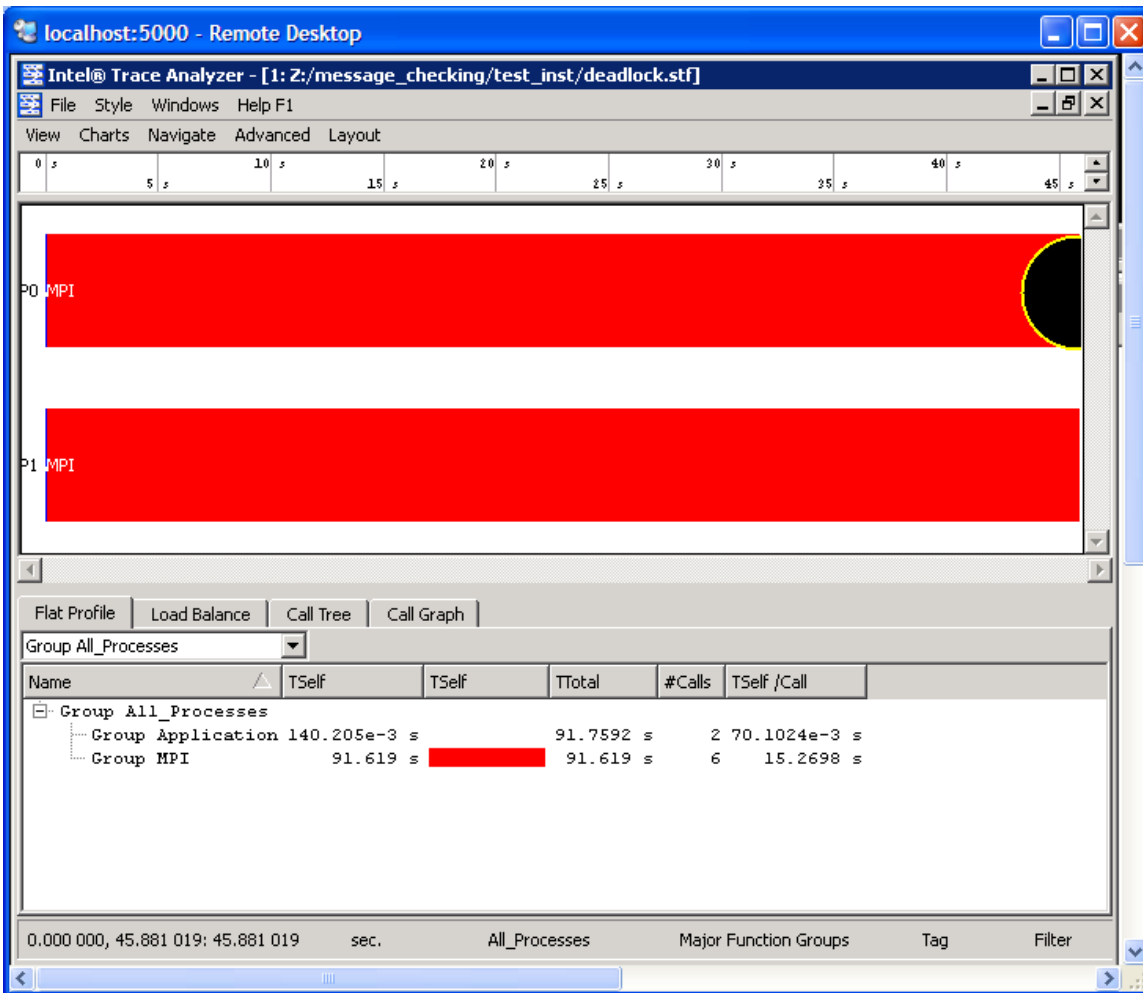


Figure 6.7 – Event Timeline illustrating an error as signified by the black circle

For the event timeline chart, errors and warnings are represented by yellow-bordered circles (Figure 6.7). The color of each circle depends on the type of the particular diagnostic. If there is an error the circle will be filled in with a black coloring. If there is a warning, the circle will be filled in with a gray coloring.

For Figure 6.7, error messages and warnings can be suppressed by using a context menu. A context menu will appear if you right click the mouse as shown in Figure 6.8 and follow the path Show->Issues. If you uncheck the Issues item, the black and gray circles will clear.

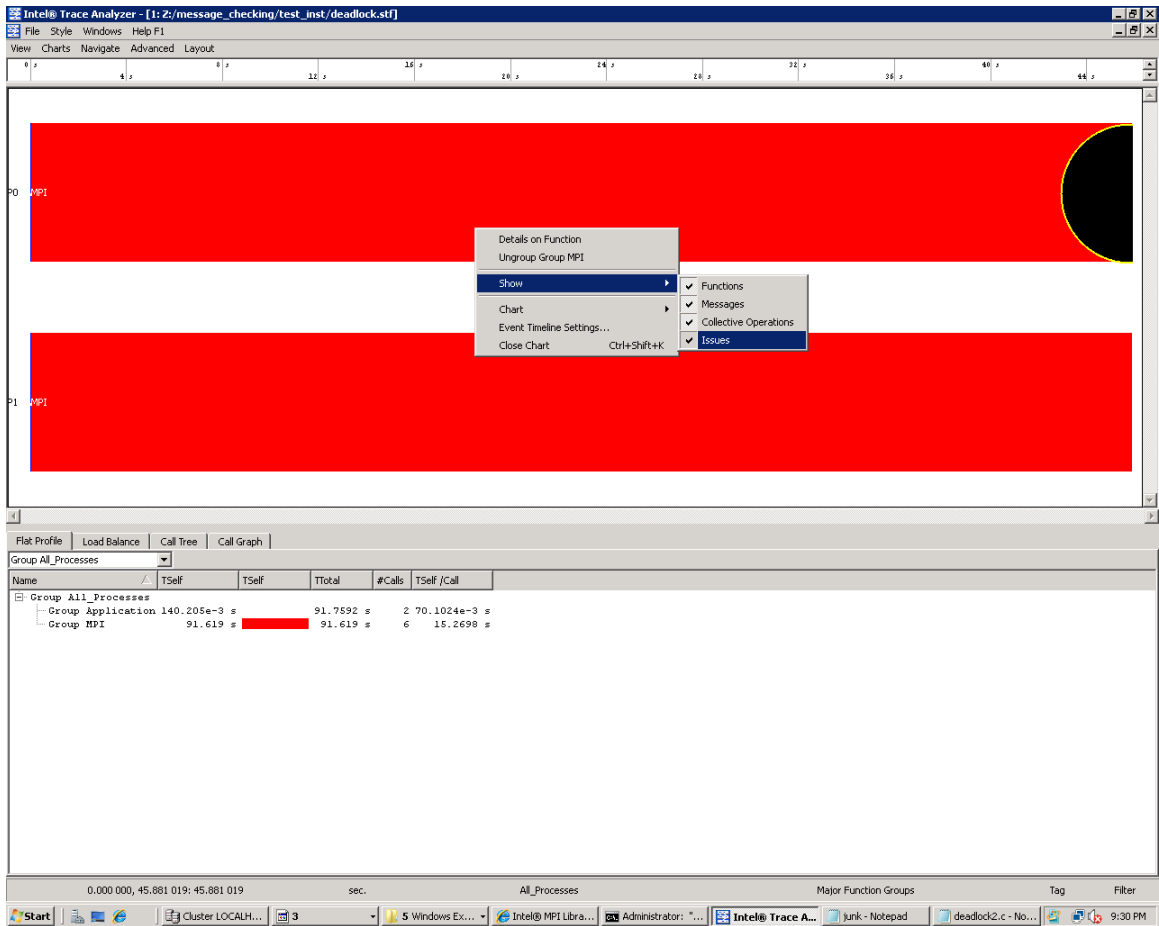


Figure 6.8 – Context menu that can be used to suppress “Issues”. This is done by un-checking the “Issues” item

One can determine what source line is associated with an error message by using the context menu and selecting Details on Function. This will generate the following Details on Function panel (Figure 6.9):

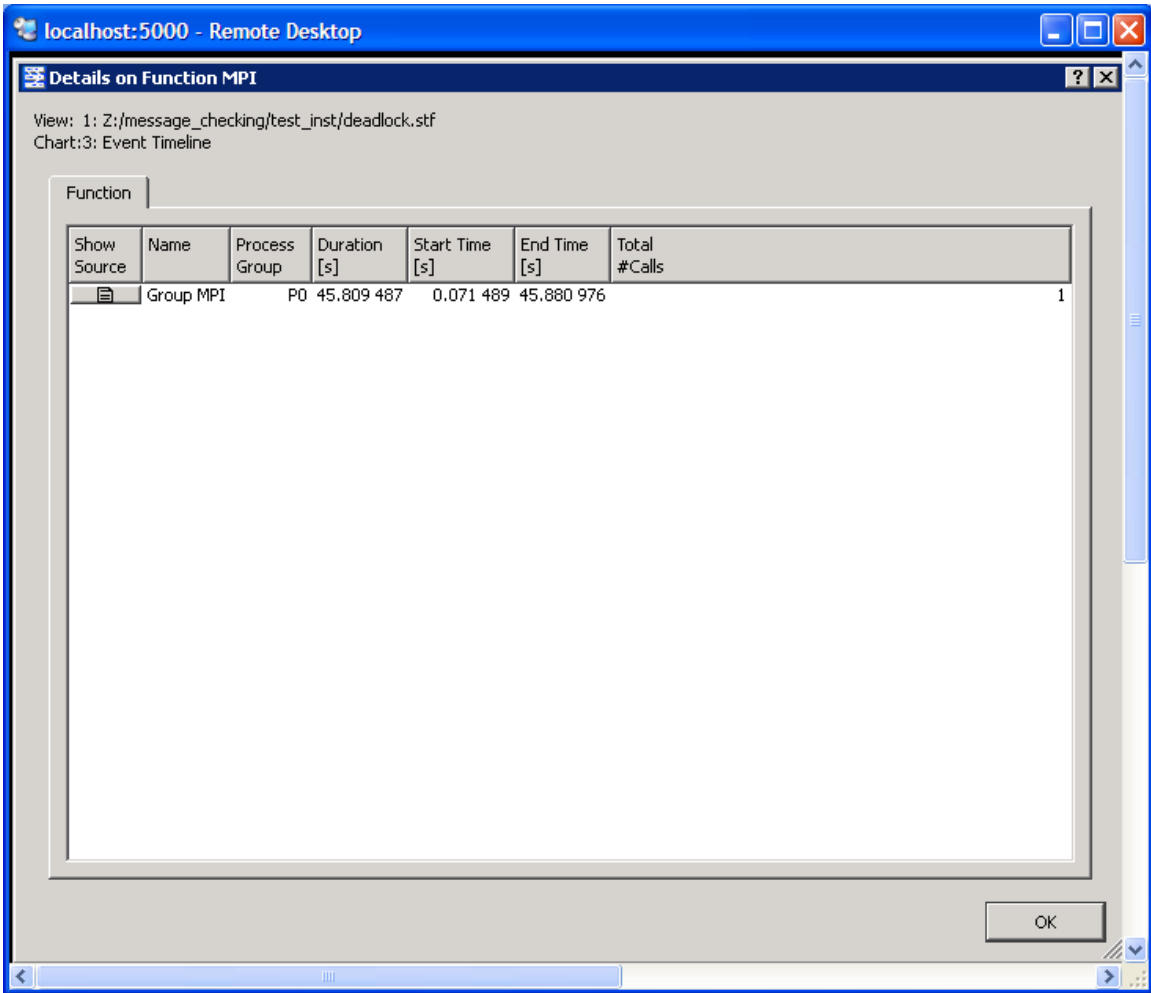


Figure 6.9 – Illustration of the Detail on Function panel. The Show Source tab is the first item on the left

If you click on the Show Source tab in Figure 6.9, you will ultimately reach a source file panel such as what is demonstrated in Figure 6.10.



Figure 6.10 – The source panel display which shows the line in the user’s source where deadlock has taken place.

The diagnostic text messages and the illustration in Figure 6.10 reference line 49 of `deadlock.c` which looks something like the following:

```

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

```

This is illustrated in Figure 6.11. To avoid deadlock situations, one might be able to

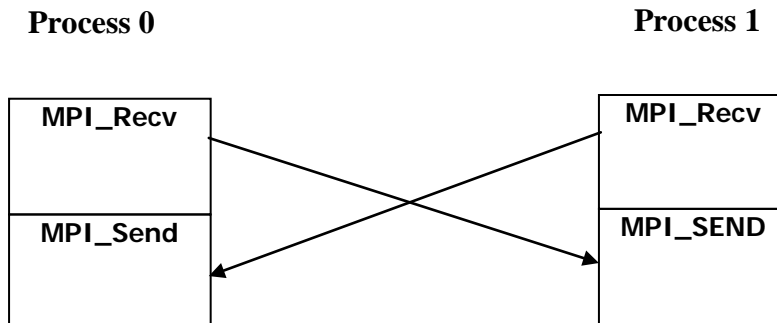


Figure 6.11 – Cycle illustration for processes 0 and 1 when executing source lines 49 and 43 within application deadlock.c

resort to the following solutions:

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

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

```

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

```

```

52             MPI_COMM_WORLD);
53     printf ("\n%d/%d: sendt %d\n", rank, size, messagelength);
54 }

```

...

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

```

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

```

and save this information into a file called deadlock2.c, and proceed to compile the modified application with the Microsoft Visual C++ compiler:

```

cl /D_CRT_SECURE_NO_DEPRECATED /Fedeadlock2_vc
/I"%I_MPI_ROOT%\em64t\include /Zi deadlock2.c /link /stack:8000000
/LIBPATH:"%VT_LIB_DIR%" VTmc.lib Ws2_32.lib bufferoverflowu.lib
/LIBPATH:"%I_MPI_ROOT%\em64t\lib" impi.lib /NODEFAULTLIB:LIBCMTD.lib

```

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

```

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

```

is the following:

...

```

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

```

```

1/2: receiving 80000

```

```

1/2: sendt 80000

```

```

0/2: receiving 80000

```

```

0/2: sendt 80000

```

This indicates the deadlock errors that were originally encountered have been eliminated for this example. Using the Intel® Trace Analyzer to view the instrumentation results, we see that the deadlock issues have been resolved (Figure 6.12).

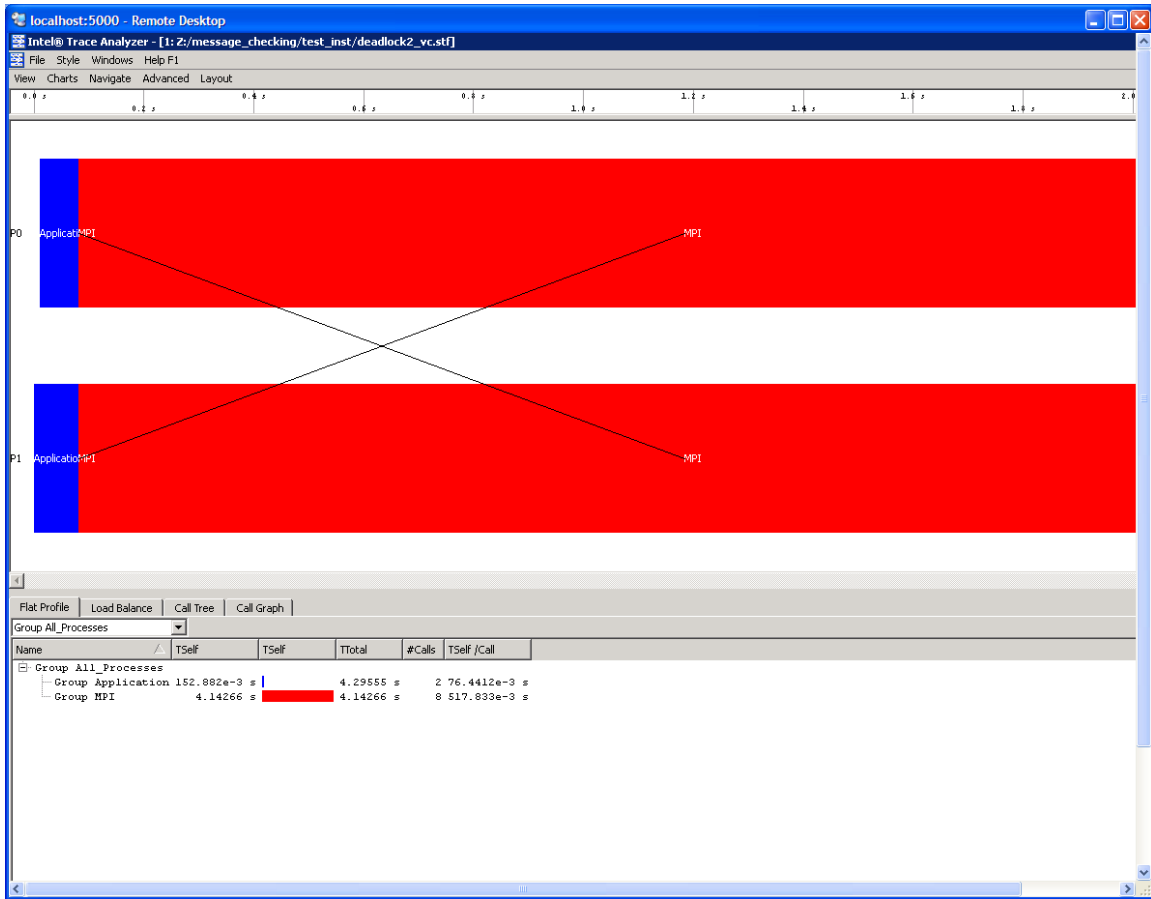


Figure 6.12 – Illustration of deadlock removal by using MPI_Sendrecv in the original source file called deadlock.c

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

If you encounter the following link error message:

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

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

```
vcvarsx86_amd64.bat
```

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

```
"C:\Program Files (x86)\Microsoft Visual Studio
8\VC\bin\x86_amd64\vcvarsx86_amd64.bat"
```

where the line above is contiguous.

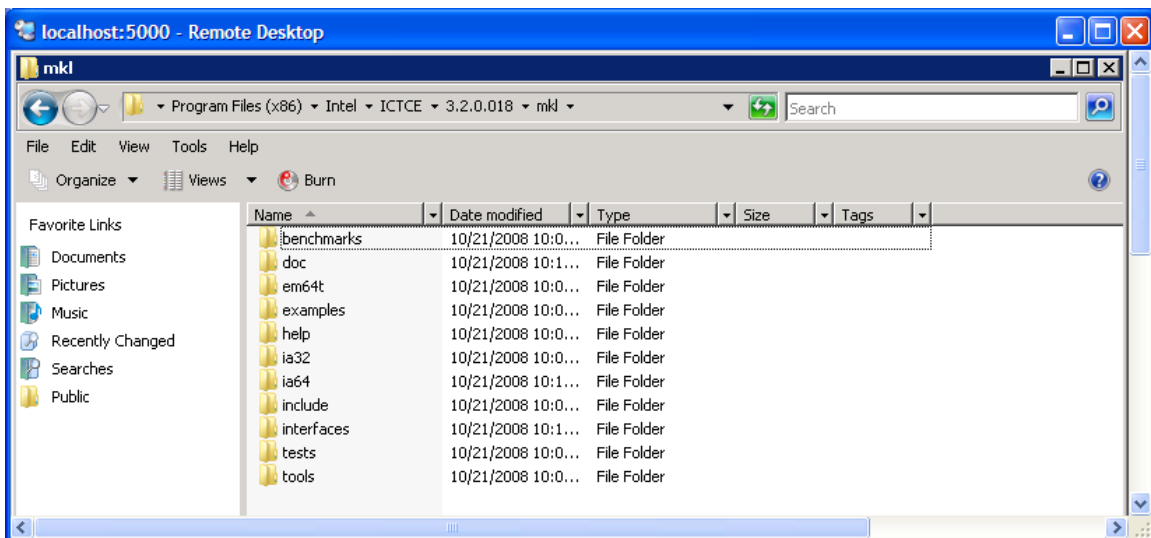
If for the ScaLAPACK experiments in this chapter, you are using a version of `nmake` from Microsoft Visual Studio 2008, then you can use the ScaLAPACK makefile variable `msvs=2008` to prevent the link error referenced above. The setting of `msvs=2008` will instruction the ScaLAPACK makefile to not use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

7.1 Experimenting with ScaLAPACK

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

```
C:\Program Files (x86)\intel\ictce\3.2.0.xxx\mkl\
```

where `xxx` is the build number of the Intel® Cluster Toolkit Compiler Edition 3.2 package. The contents of `...\mkl` sub-folder should be:



To experiment with the ScaLAPACK (SCALable LAPACK) test suite, recursively copy the contents of the directory path:

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

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

```
cd scalapack
```

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

```
nmake msvs=2008 arch=em64t mpi=intelmpi MPIdir="%I_MPI_ROOT%\em64t"  
libtype=static run
```

if you are using an `nmake` from Microsoft* Visual Studio* 2008. Otherwise use the command:

```
nmake arch=em64t mpi=intelmpi MPIdir="%I_MPI_ROOT%\em64t"  
libtype=static run
```

In the `scalapack` working directory where the `nmake` command was issued, the ScaLAPACK executables can be found in `source\TESTING`, and the results of the computation will also be placed into this same sub-directory. The results will be placed into `"*.txt"` files. You can invoke an editor to view the results in each of the `"*.txt"` files that have been created.

As an example result, the file `"cdtlu_em64t_static_intelmpi_lp64.exe.txt"` might have something like the following in terms of contents for an execution 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 `xcdtlu_em64t_static_intelmpi_lp64.exe`.

SCALAPACK banded linear systems.
 'MPI machine'

Tests of the parallel complex single precision band matrix solve

The following scaled residual checks will be computed:

Solve residual = $\|Ax - b\| / (\|x\| * \|A\| * \text{eps} * N)$

Factorization residual = $\|A - LU\| / (\|A\| * \text{eps} * N)$

The matrix A is randomly generated for each test.

An explanation of the input/output parameters follows:

TIME : Indicates whether WALL or CPU time was used.

N : The number of rows and columns in the matrix A.

bwl, bwu : The number of diagonals in the matrix A.

NB : The size of the column panels the matrix A is split into. [-1 for default]

NRHS : The total number of RHS to solve for.

NBRHS : The number of RHS to be put on a column of processes before going on to the next column of processes.

P : The number of process rows.

Q : The number of process columns.

THRESH : If a residual value is less than THRESH, CHECK is flagged as PASSED

Fact time: Time in seconds to factor the matrix

Sol Time: Time in seconds to solve the system.

MFLOPS : Rate of execution for factor and solve using sequential operation count.

MFLOP2 : Rough estimate of speed using actual op count (accurate big P,N).

The following parameter values will be used:

```

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

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

Routines pass computational tests if scaled residual is less than 3.0000

TIME	TR	N	BWL	BWU	NB	NRHS	P	Q	L*U Time	Slv Time	MFLOPS	MFLOP2	CHECK
WALL	N	3	1	1	3	4	1	1	0.000	0.0003	0.45	0.43	PASSED
WALL	N	5	1	1	5	4	1	1	0.000	0.0003	0.82	0.77	PASSED
WALL	N	17	1	1	17	4	1	1	0.000	0.0003	2.77	2.63	PASSED
WALL	N	3	1	1	2	4	1	2	0.000	0.0047	0.05	0.07	PASSED
WALL	N	5	1	1	3	4	1	2	0.000	0.0004	0.56	0.84	PASSED
WALL	N	17	1	1	9	4	1	2	0.000	0.0004	1.96	2.97	PASSED
WALL	N	3	1	1	2	4	1	3	0.000	0.0005	0.24	0.35	PASSED
WALL	N	5	1	1	2	4	1	3	0.000	0.0038	0.09	0.16	PASSED
WALL	N	17	1	1	6	4	1	3	0.001	0.0009	0.82	1.25	PASSED
WALL	N	3	1	1	2	4	1	4	0.001	0.0011	0.13	0.19	PASSED
WALL	N	5	1	1	2	4	1	4	0.001	0.0011	0.19	0.33	PASSED
WALL	N	17	1	1	5	4	1	4	0.001	0.0041	0.27	0.42	PASSED

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

END OF TESTS.

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

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 one cluster configuration to another.

If you proceed to load the `cdtlu_em64t_static_intelmpi_lp64.exe.txt` table above into a Microsoft Excel* Spreadsheet, and build a chart to compare the Time in Seconds to Solve the System (SLV) and the Megaflop values, you might see something like the following (Figure 7.1):

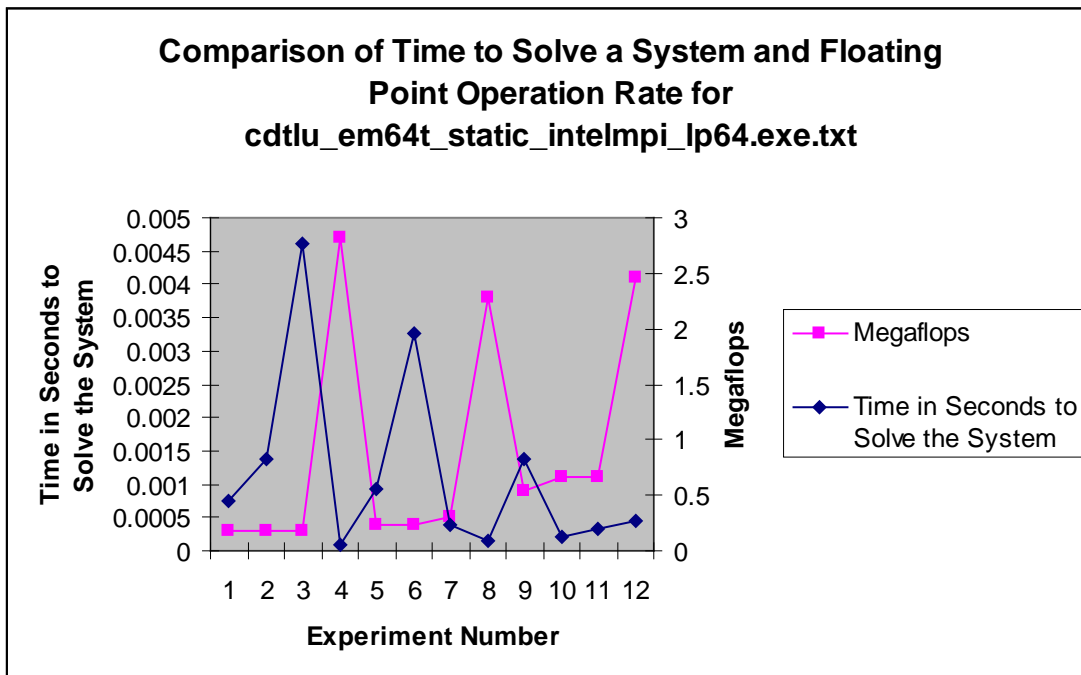


Figure 7.1 – Display of ScaLAPACK DATA from the executable `xcdtlu_em64t_static_intelmpi_lp64.exe`

You can also link the libraries dynamically. The following command will provide for this if `nmake` is part of Microsoft Visual Studio 2008:

```
nmake msvs=2008 arch=em64t mpi=intelmpi MPIdir="%I_MPI_ROOT%\em64t"
libtype=dynamic run
```

Otherwise, use the command:

```
nmake arch=em64t mpi=intelmpi MPIdir="%I_MPI_ROOT%\em64t"
libtype=dynamic run
```

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

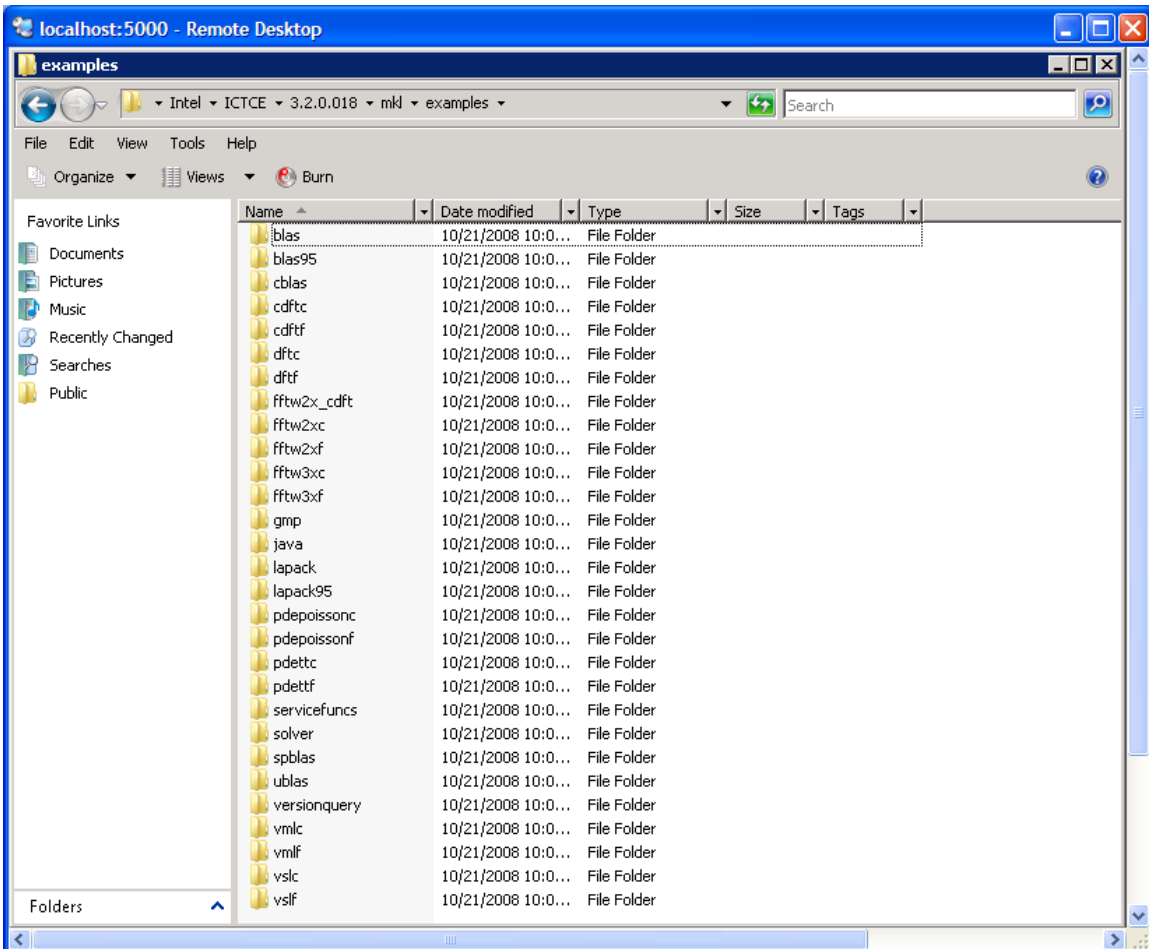
```
nmake cleanall
```

7.2 Experimenting with the Cluster DFT Software

On Microsoft Windows CCS, in the folder path:

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

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



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

nmake help

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

```
nmake libem64t mpi=intelmpi mpidir="%I_MPI_ROOT%"  
workdir="c:\MPI_Share_Area\cdftc_test"
```

and

```
nmake libem64t mpi=intelmpi mpidir="%I_MPI_ROOT%"  
workdir="c:\MPI_Share_Area\cdftf_test"
```

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

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

```
_results\lp64_em64t_intelmpi
```

The executable and result contents of each of the subfolder paths `_results\lp64_em64t_intelmpi` might respectively look something like:

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

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 look something like:

Program is running on 2 processes

DM_COMPLEX_2D_DOUBLE_EX1

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

Configuration parameters:

DFTI_FORWARD_DOMAIN = DFTI_COMPLEX
DFTI_PRECISION = DFTI_DOUBLE
DFTI_DIMENSION = 2
DFTI_LENGTHS (MxN) = {19,12}
DFTI_FORWARD_SCALE = 1.0
DFTI_BACKWARD_SCALE = 1.0/(m*n)

INPUT Global vector X, n columns

Row 0:
(1.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 2:
(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 4:
(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 6:
(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 8:
(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 10:
(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
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)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(
0.000, 0.000)(0.000, 0.000)(0.000, 0.000)(0.000, 0.000)
Row 12:

```

( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 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)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
Row 14:
( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
Row 15:
( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
Row 16:
( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
Row 17:
( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
Row 18:
( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000,
0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)(
0.000, 0.000)( 0.000, 0.000)( 0.000, 0.000)
...

```

Note that the output results that you obtain will be a function of your cluster configuration.

Recall the `itcpin` discussion in [Section 6.3 Using itcpin to Instrument an Application](#), where `itcpin` is the instrumentation tool used to insert Intel Trace Collector calls into the executables. Using `itcpin` technology, the following sequence of shell commands could be used to create instrumented executables and generate result information for the executables located in `_results\lp64_em64t_intelmpi`.

For the C language version of the Cluster Discrete Fourier Transform, the DOS Shell commands might look something like:

Intel® Processor Architecture	Command-line Sequence for Microsoft Windows	Trace Results are Located In	Execution Results are Located In
Intel® 64 (formerly EM64T)	<pre> set VT_LOGFILE_PREFIX=cdftc_inst rmdir /S /Q %VT_LOGFILE_PREFIX% mkdir %VT_LOGFILE_PREFIX% mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_double_ex1.dat" > "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex2.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_double_ex2.dat" > "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex2.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_single_ex1.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_single_ex1.dat" > "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_single_ex1.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_single_ex2.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_single_ex2.dat" > "%CD%_results\lp64_em64t_intelmpi\dm_complex_2d_single_ex2.res" </pre>	%CD%\cdfc_inst	%CD%_results\lp64_em64t_intelmpi

The "xxx" in the folder path 3.2.0.xxx needs to be replaced with the appropriate build number that is associated with the Intel® Cluster Toolkit Compiler Edition installation on your system.

In this table, the four executables are supplemented with instrumentation calls to the Intel Trace Collector. These DOS commands could be copied from the table above and pasted into a .bat file.

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

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

```
mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL
"%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF -env VT_PCTTRACE 5 -env
VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin
--run --insert VT --
"%CD%\_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.exe" <
"C:\Program Files
(x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_doubl
e_ex1.dat" >
"%CD%\_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.res"
```

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

```
-env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env
VT_LOGFILE_FORMAT STF -env VT_PCTTRACE 5 -env VT_LOGFILE_PREFIX
"%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON"
```

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

In continuing the itcpin review, the itcpin component as part of the overall mpiexec command-line is:

```
itcpin --run --insert VT --
"%CD%\_results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.exe" <
"C:\Program Files
(x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_doubl
e_ex1.dat"
```

The data input file for executable dm_complex_2d_double_ex1.exe is:

```
"C:\Program Files
(x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdftc\data\dm_complex_2d_doubl
e_ex1.dat"
```

Remember that the *actual reference* to the Intel Math Kernel Library folder path for the data input file on *your* cluster system will be dependent on where you installed the Intel® Cluster Toolkit Compiler Edition software package and the value of `xxx` in the subfolder name `3.2.0.xxx`.

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

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

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

```
--run --insert VT
```

The switch called `--run` instructs `itcpin` to run the application executable. The `--insert` option is followed by an instrumentation library value of `VT`. In this case, `VT` refers to the instrumentation library is `VT.lib` which is for the Intel Trace Collector. Also, remember that you can find out additional information about `itcpin` in the Intel Trace Collector User's Guide under the search topic `itcpin`.

With regards to the test area referenced by the folder path `c:\MPI_Share_Area\cdftf_test`, the Fortran language version of Cluster Discrete Fourier Transform could be instrumented with `itcpin` as follows:

Intel® Processor Architecture	Command-line Sequence for Microsoft Windows	Trace Results are Located In	Execution Results are Located In
Intel® 64 (formerly EM64T)	<pre> set VT_LOGFILE_PREFIX=cdf_Inst rmdir /S /Q %VT_LOGFILE_PREFIX% mkdir %VT_LOGFILE_PREFIX% mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdfc\data\dm_complex_2d_double_ex1.dat" > "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_double_ex1.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_double_ex2.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdfc\data\dm_complex_2d_double_ex2.dat" > "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_double_ex2.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_single_ex1.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdfc\data\dm_complex_2d_single_ex1.dat" > "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_single_ex1.res" mpiexec -n 4 -env VT_DLL_DIR "%VT_DLL_DIR%" -env VT_MPI_DLL "%VT_MPI_DLL%" -env VT_LOGFILE_FORMAT STF - env VT_PCTTRACE 5 -env VT_LOGFILE_PREFIX "%VT_LOGFILE_PREFIX%" -env VT_PROCESS "0:N ON" itcpin --run --insert VT -- "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_single_ex2.exe" < "C:\Program Files (x86)\Intel\ICTCE\3.2.0.xxx\mkl\examples\cdfc\data\dm_complex_2d_single_ex2.dat" > "%CD%\results\lp64_em64t_intelmpi\dm_complex_2d_single_ex2.res" </pre>	%CD%\cdf_Inst	%CD%\results\lp64_em64t_intelmpi

As was mentioned previously for the C version of the Cluster Discrete Fourier Transform examples, the "xxx" in the folder path 3.2.0.xxx needs to be replaced

with the appropriate build number that is associated with the Intel® Cluster Toolkit Compiler Edition installation on your system.

The DOS commands above could be copied and pasted into a .bat file within a test area such as c:\MPI_Share_Area\cdftf_test. In regards to the test areas c:\MPI_Share_Area\cdftc_test and c:\MPI_Share_Area\cdftf_test, the tracing data for the executables should be deposited respectively into the folders cdftc_inst and cdftf_inst. Note that in the two tables above, the setting of the environment variable VT_LOGFILE_PREFIX resulted in the deposit of trace information into the directories cdftc_inst and cdftf_inst as demonstrated with a listing of the Structured Trace Format (STF) index files:

```
cdftc_inst\dm_complex_2d_double_ex1.exe.stf
cdftc_inst\dm_complex_2d_double_ex2.exe.stf
cdftc_inst\dm_complex_2d_single_ex1.exe.stf
cdftc_inst\dm_complex_2d_single_ex2.exe.stf
```

and

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

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

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

Figure 7.2 shows the result of simultaneously displaying the Function Profile Chart and the Event Timeline Chart.

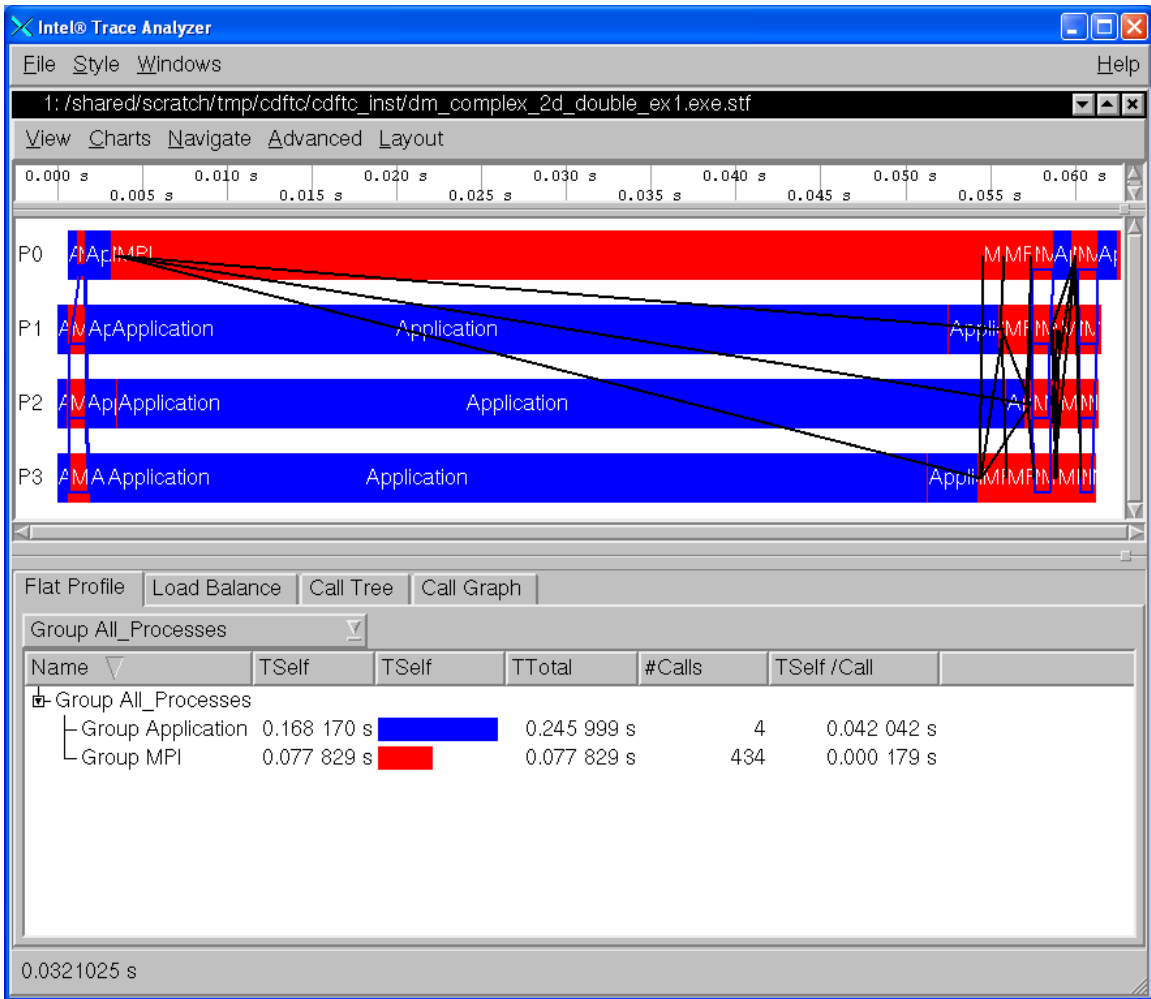


Figure 7.2 – The Event Timeline Chart and the Function Profile Chart for a Cluster Discrete Fourier Transform Example

The file <directory-path-to-mkl>\doc\mkl_documentation.htm contains a landing page linking various documentation files associated with Intel MKL 10.1. To make inquiries about Intel Math Kernel Library 10.1, visit the URL: <http://premier.intel.com>.

8. Using the Intel® MPI Benchmarks

The Intel MPI Benchmarks have been ported to Microsoft Windows*. The folder structure for the Intel® MPI Benchmarks 3.2 looks something like the following where the parenthesized text contains descriptive information:

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

The WINDOWS folder as noted above contains Microsoft* Visual Studio* 2005 and 2008 project folders which allow you to use a pre-existing ".vcproj" project file in conjunction with Microsoft* Visual Studio* to build and run the associated Intel® MPI Benchmark application.

Within Microsoft Windows Explorer and starting at the Windows folder, you can go to one of the subfolders IMB-EXT_VS_2005, IMB-EXT_VS_2008, IMB-IO_VS_2005, IMB-IO_VS_2008, IMB-MPI1_VS_2005, or IMB-MPI1_VS_2008 and click on the corresponding ".vcproj" file and open up Microsoft Visual Studio* (Figure 8.1).

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

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

In Figure 8.1 Microsoft Visual Studio* 2008 is being used.

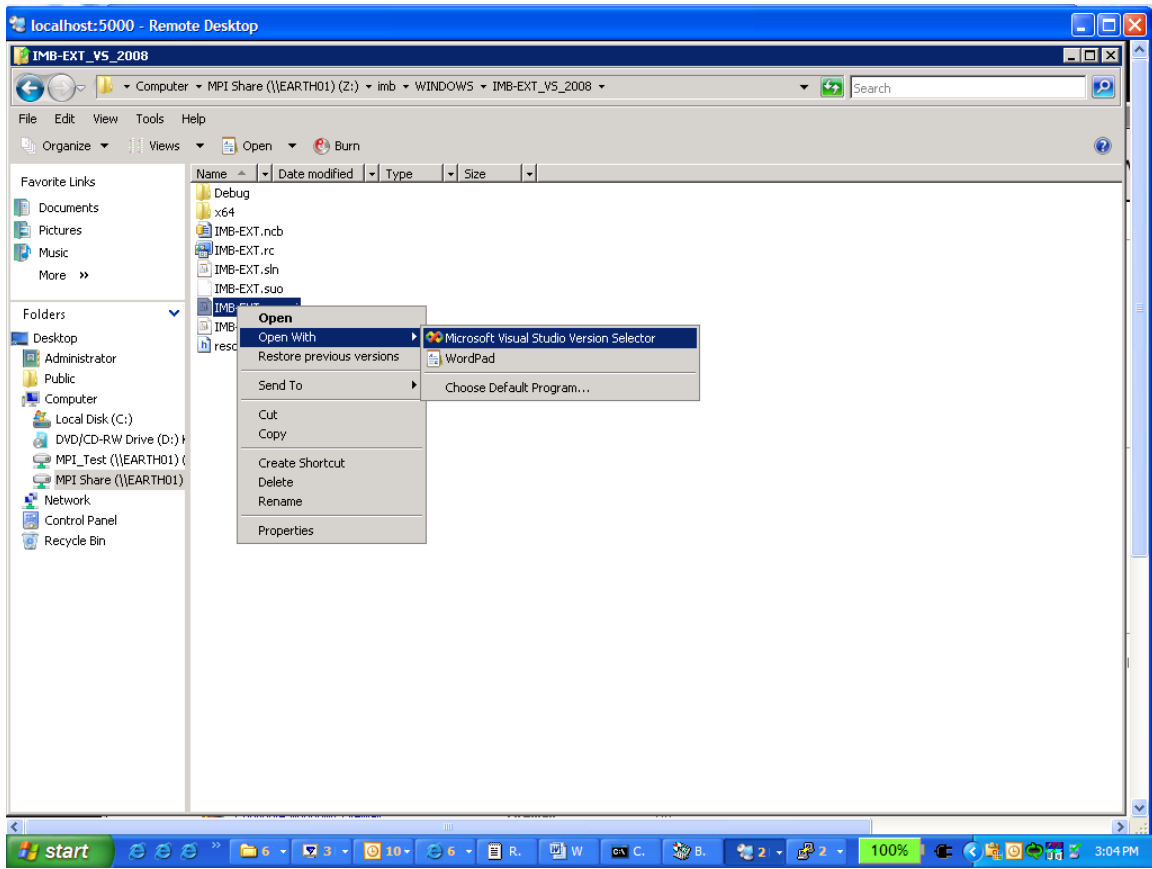


Figure 8.1 – Illustration for Starting Microsoft Visual Studio* 2008 on the project file IMB-EXT.vcproj

From the Visual Studio Project panel:

- 1) Change the "Solution Platforms" dialog box to "x64". See Figure 8.2.
- 2) Change the "Solution Configurations" dialog box to "Release". See Figure 8.2

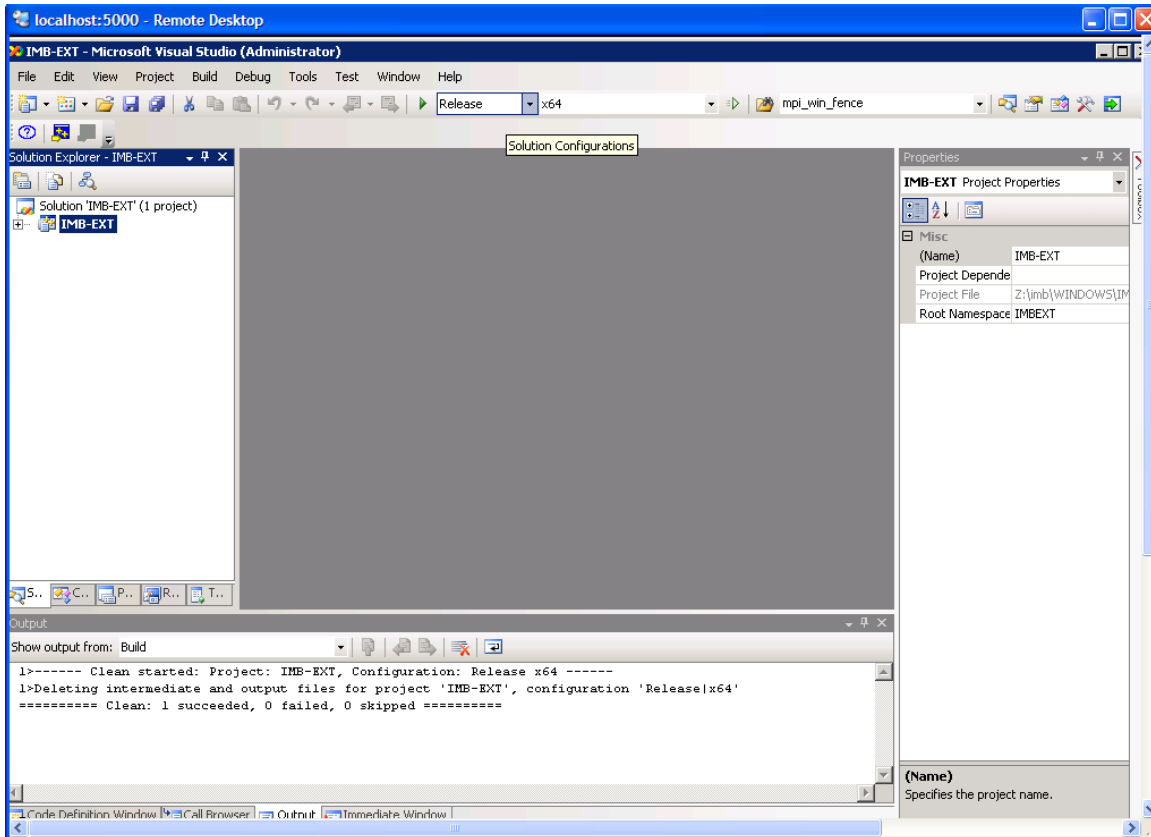


Figure 8.2 – The Solution Configuration is set to “ReleasE” and Solution Platforms is set to “x64”. Also note that IMB-EXT is highlighted in the Solution Explorer panel on the left in preparation for the context sensitive operations outlined in step 3

- 3) Follow the menu path Project->Properties or Alt+F7 and check to make sure that the following are set by expanding "Configuration Properties":
 - a) General->Project Defaults - Change "Character Set" to "Use Multi-Byte Character Set"
 - b) Debugging
 - i) Set the "Debugger to launch" value to "Local Windows Debugger", for example. Note that "Local Windows Debugger" is one possible setting. See Figure 8.3.
 - ii) For the row "Command" add "\$ (I_MPI_ROOT)\em64t\bin\mpiexec.exe". Be sure to include the quotes.
 - iii) For the row "Command Arguments" add "-n 2 \$(TargetPath)"
 - c) C/C++->General
 - i) For the row "Additional Include Directories", add "\$ (I_MPI_ROOT)\em64t\include".
 - ii) For the row "Warning Level", set the warning level to "Level 1 (/W1)"
 - d) Linker->Input

i) For the row "Additional Dependencies" add
" $\$(I_MPI_ROOT)\backslash\text{em64t}\backslash\text{lib}\backslash\text{impi.lib}$ "
If items "a" through "d" are already set, then proceed to step 4.

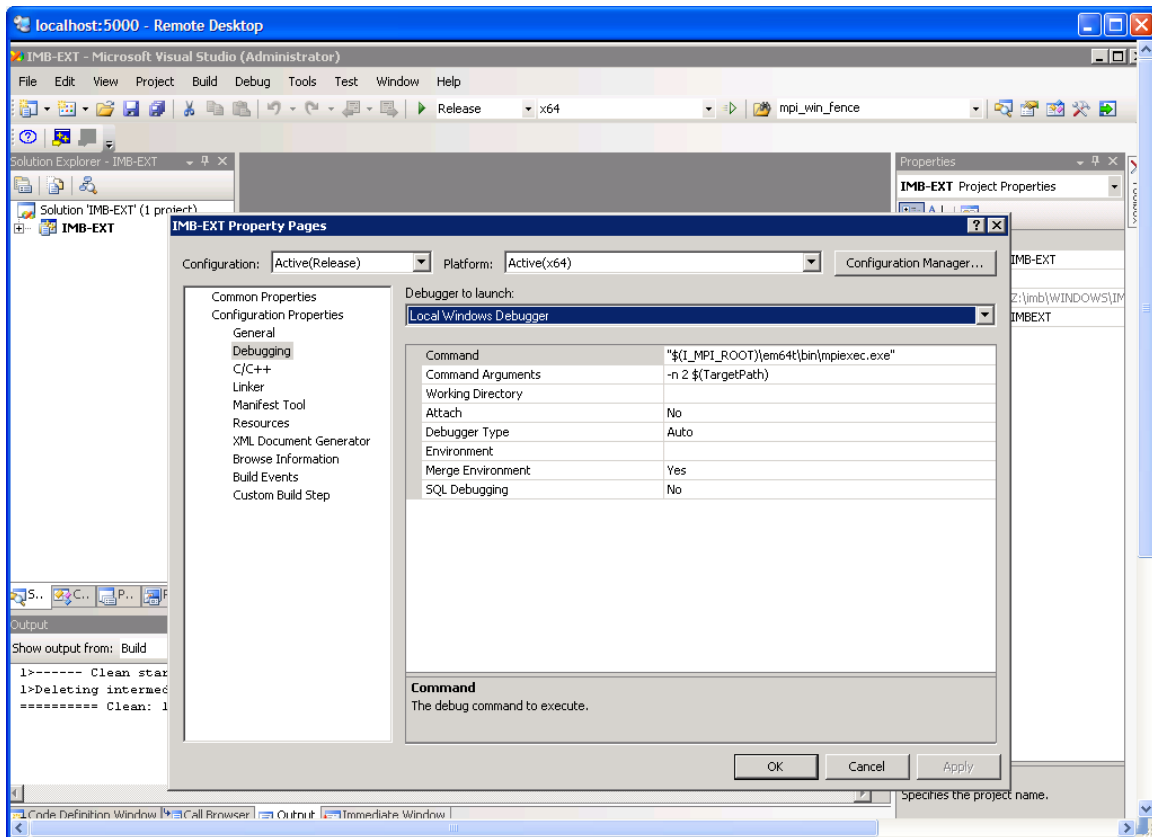


Figure 8.3 – Setting the Command and Command Arguments for Debugging under Configuration Properties

3) Use F7 or Build->Build Solution to create an executable. See Figure 8.4.

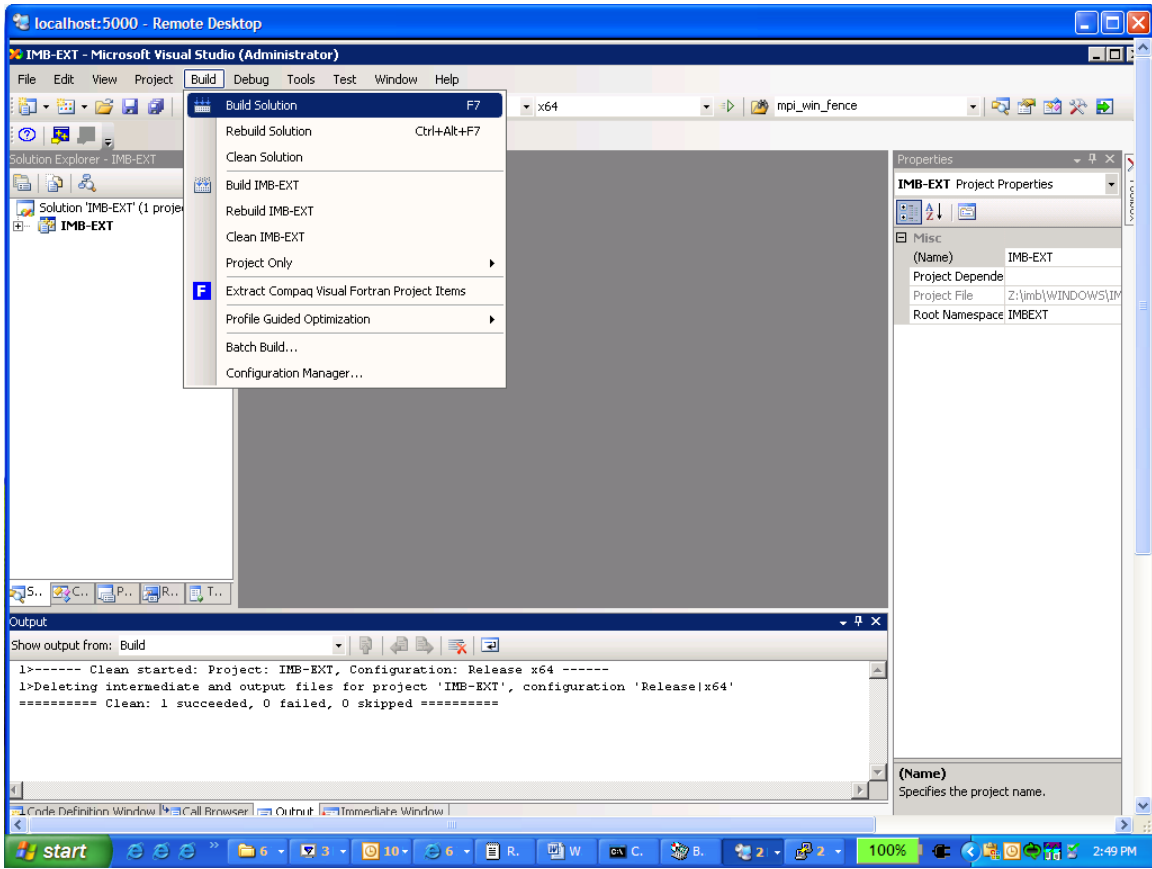


Figure 8.4 – Microsoft* Visual Studio* 2008 Illustration for building a solution for IMB-EXT

5) Use Debug->Start Without Debugging or Ctrl+F5 to run the executable. See Figure 8.5

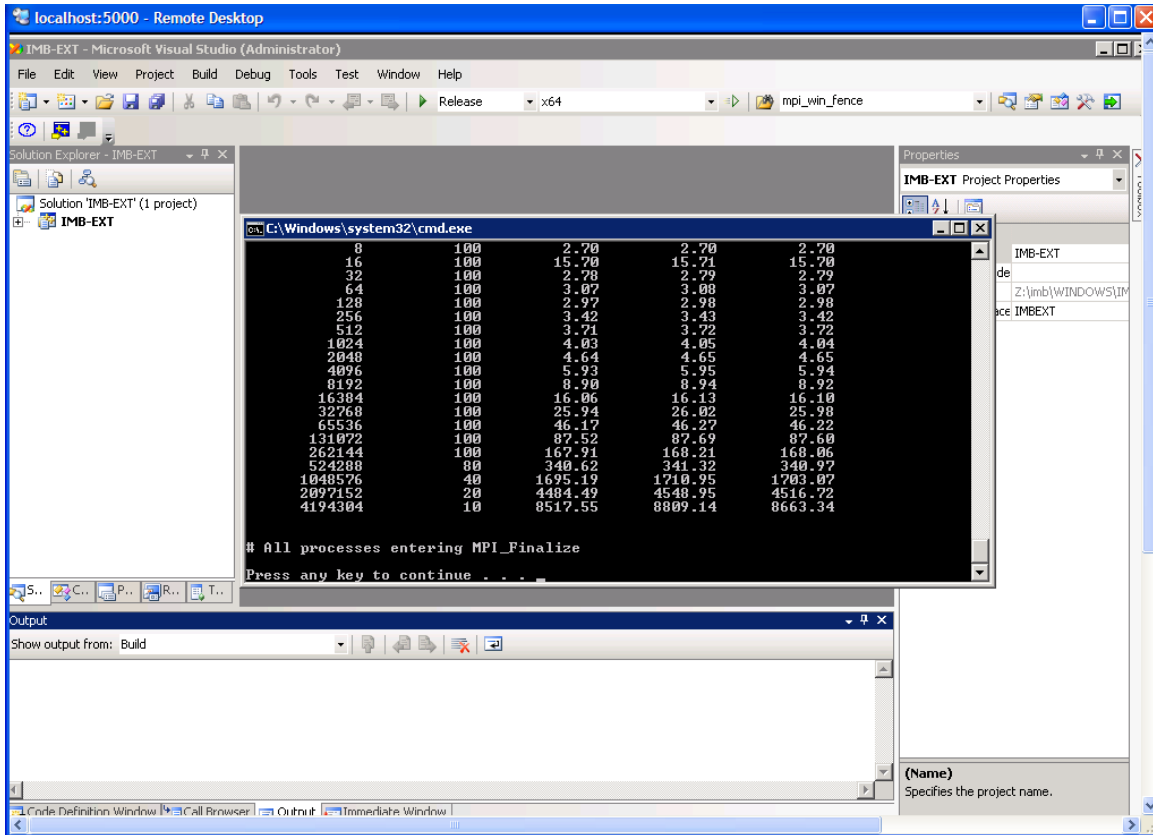


Figure 8.5 – Generation of the command-line panel using the keys Ctrl+F5. The command-line panel shows the execution results for IMB-EXT.exe within Microsoft Visual Studio* 2008

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

9. Uninstalling the Intel® Cluster Toolkit Compiler Edition on Microsoft Windows CCS

For Microsoft Windows CCS, if you wish to uninstall the Intel Cluster Toolkit Compiler Edition, follow the menu path Start->Control Panel->Add or Remove Programs. For this display panel find the appropriate version of the Intel Cluster Toolkit Compiler Edition that you wish to remove, highlight it, and then click on the corresponding Remove button.

The Add or Remove Display panel might look something like the following:

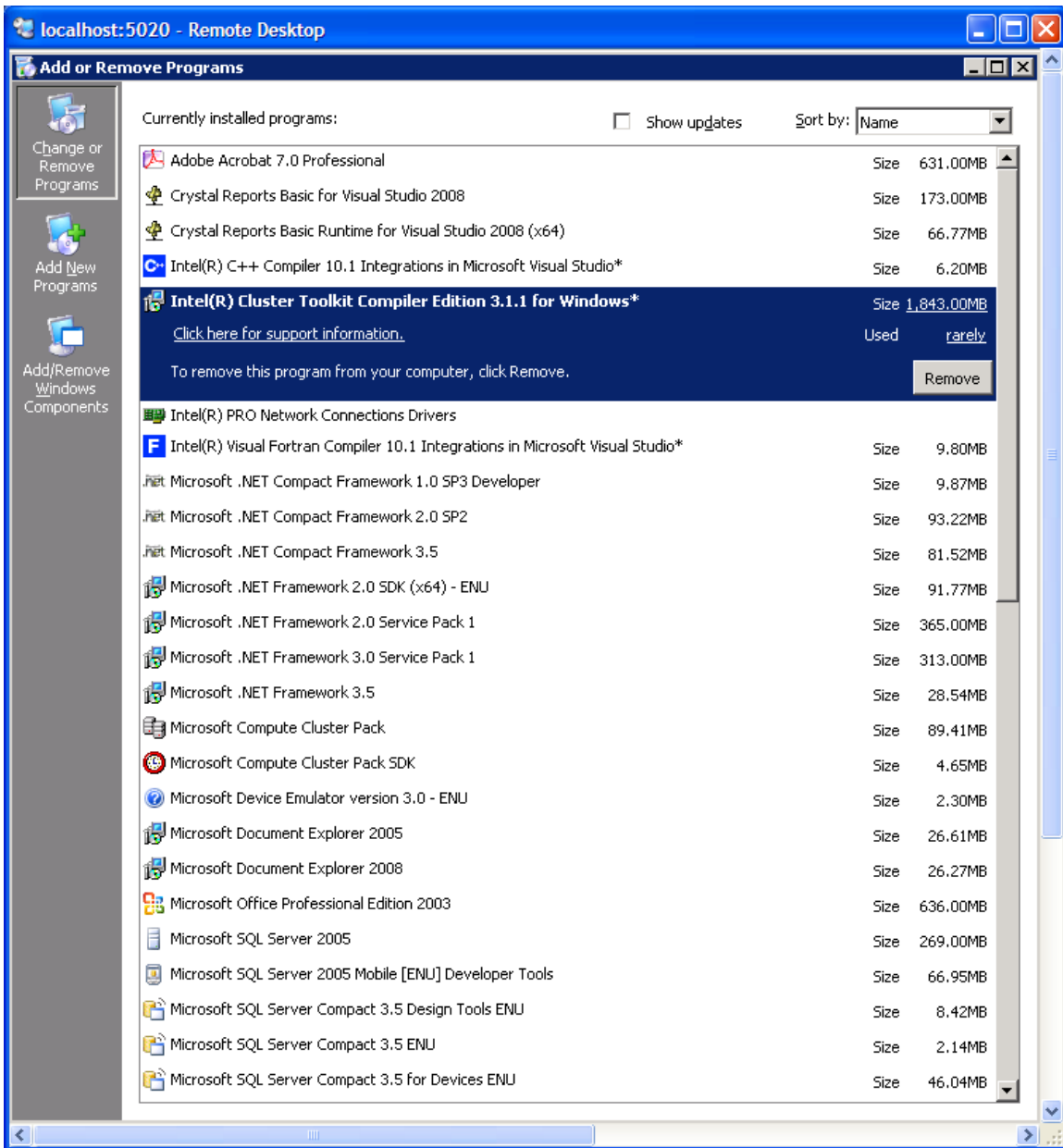


Figure 9.1 – The Add or Remove Programs Panel where Intel® Cluster Toolkit Compiler Edition has been highlighted

Figure 9.1 shows the row for the Intel® Cluster Toolkit Compiler Edition application as being highlighted. If you click on the “Remove” button that is shown in the right portion of the screen, the software components of the Intel® Cluster Toolkit Compiler Edition will be removed from all of the nodes of the cluster. Note that after completion of the uninstall, the row entry for Intel® Cluster Toolkit Compiler Edition will be absent, but the individual components (e.g. Intel® MPI Library) will still appear on the panel. If you press the F5 key on your keyboard the Add or Remove

Programs panel will be refreshed, and this should verify that the individual software subcomponents have been removed from the cluster.

10. Hardware Recommendations for Installation on Microsoft* Windows* CCS

Processor System Requirements

Intel® Pentium® 4 processor, or
 Intel® Xeon® processor, or
 Intel® Core™2 Duo processor (example of Intel® 64 (formerly Intel EM64T) architecture)

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

Disk-Space Requirements

20 GBs of disk space (minimum)

Note that during the installation process the installer may need approximately 4 gigabytes of temporary disk storage to manage the intermediate installation files.

Operating System Requirements for Microsoft Windows

OS Distributions	IA-32 Architecture	Intel® 64 Architecture		IA-64 Architecture
		32-Bit Applications	64-Bit Applications	
Microsoft* Windows* Compute Cluster Server (Microsoft Windows CCS*)	N/A	S	S	N/A
Microsoft* Windows* HPC Server 2008	N/A	S	S	N/A

S = Supported

Memory Requirements

2 GB of RAM (minimum)

11. System Administrator Checklist for Microsoft Windows CCS

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

12. User Checklist for Microsoft Windows CCS

Locate the .bat file called `ictvars.bat`, and issue the command:

```
ictvars.bat
```

within a DOS command-line session.

13. Using the Compiler Switch /Qtcollect

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

Library Selection	Meaning	How to Request
VT.lib	Default library	<code>/Qtcollect</code>
VTcs.lib	Client-server trace collection library	<code>/Qtcollect=VTcs</code>
VTfs.lib	Fail-safe trace collection library	<code>/Qtcollect=VTfs</code>

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

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

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

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

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

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

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

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

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

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

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

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

```
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testc_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testcpp_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf_Qtcollect
mpiexec -n 4 -env VT_LOGFILE_PREFIX test_inst testf90_Qtcollect
```

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

```
traceanalyzer
```

Figure 13.1 shows the base panel for the Intel Trace Analyzer as a result of invoking the command above from a DOS window.

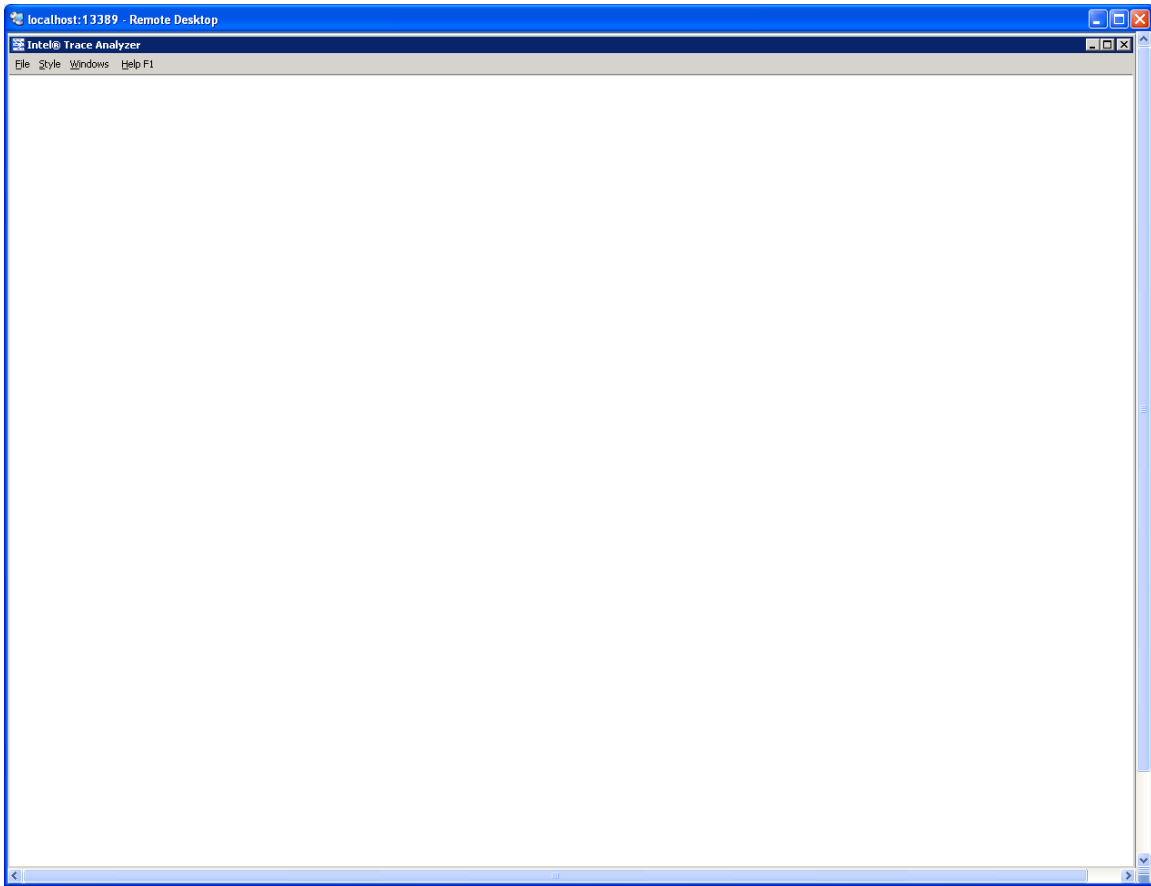


Figure 13.1 – Base panel for the Intel Trace Analyzer when invoking the DOS Command: `traceanalyzer` without any arguments

If you select the menu path `File->Open` and click on the `test_inst` folder, the following panel will appear (Figure 13.2):

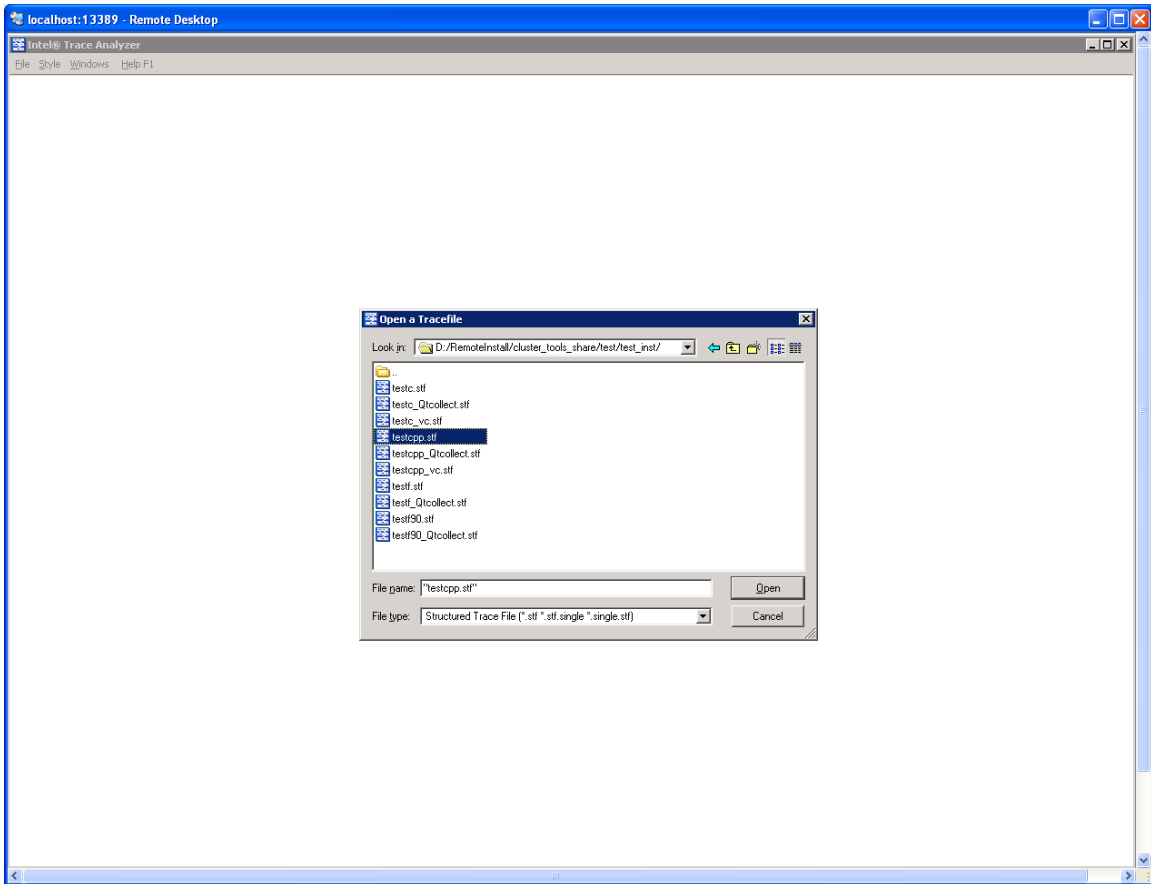


Figure 13.2 – Open a Tracefile Rendering for the test_inst Folder where testcpp.stf has been Highlighted

Selecting testcpp.stf will generate a Flat Profile panel within the Intel Trace Analyzer session that might look something like the following (Figure 13.3).

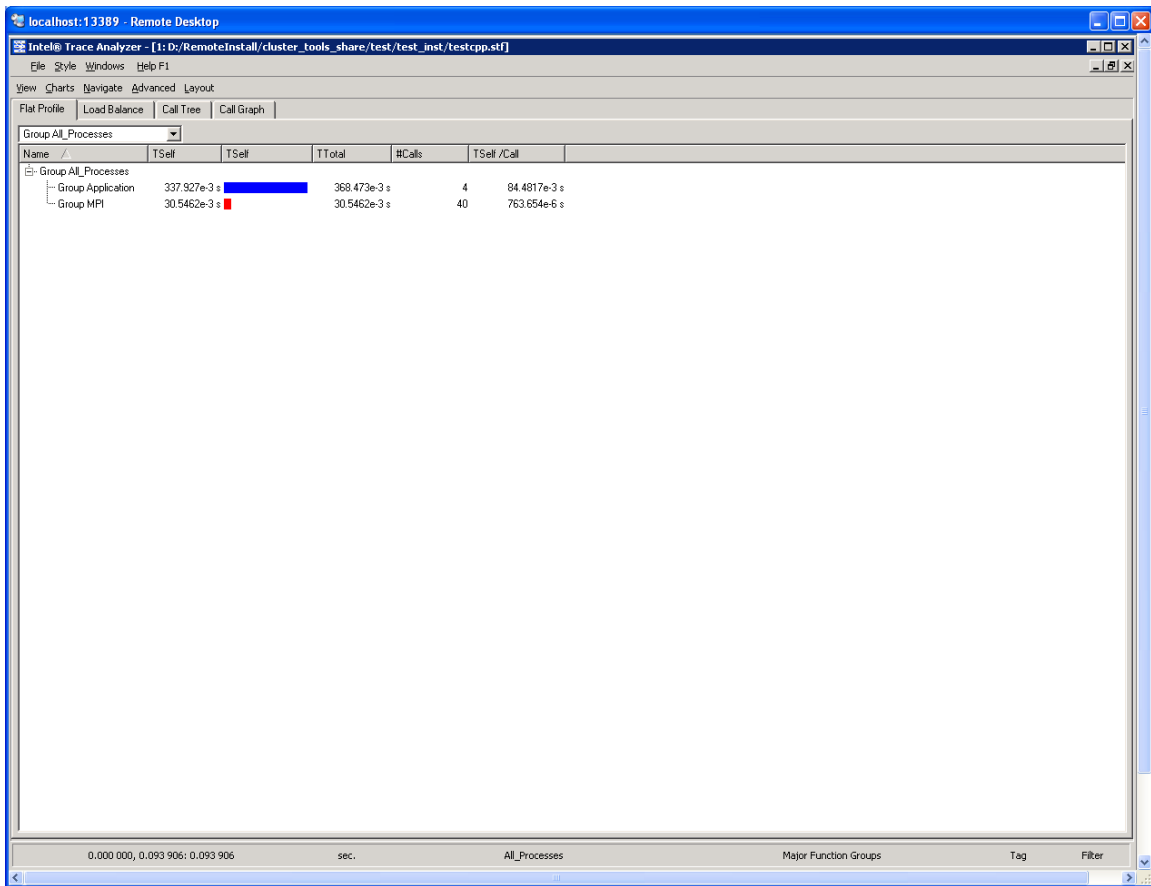


Figure 13.3 – Flat Panel Display for test_inst\testcpp.stf

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

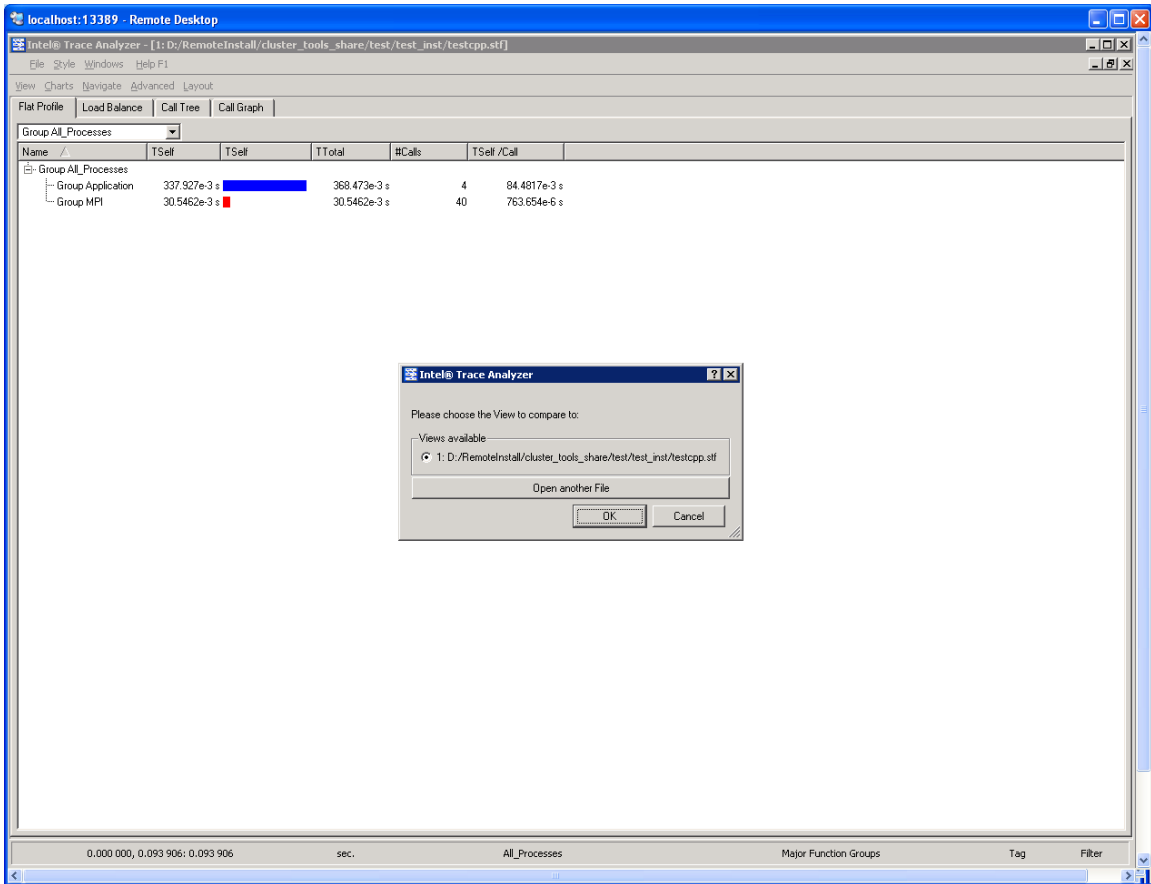


Figure 13.4 – Sub-panel Display for Adding a Comparison STF File

Click on the “Open another file” button and select `testcpp_Qtcollect.stf` and then proceed to push on the Open button with your mouse.

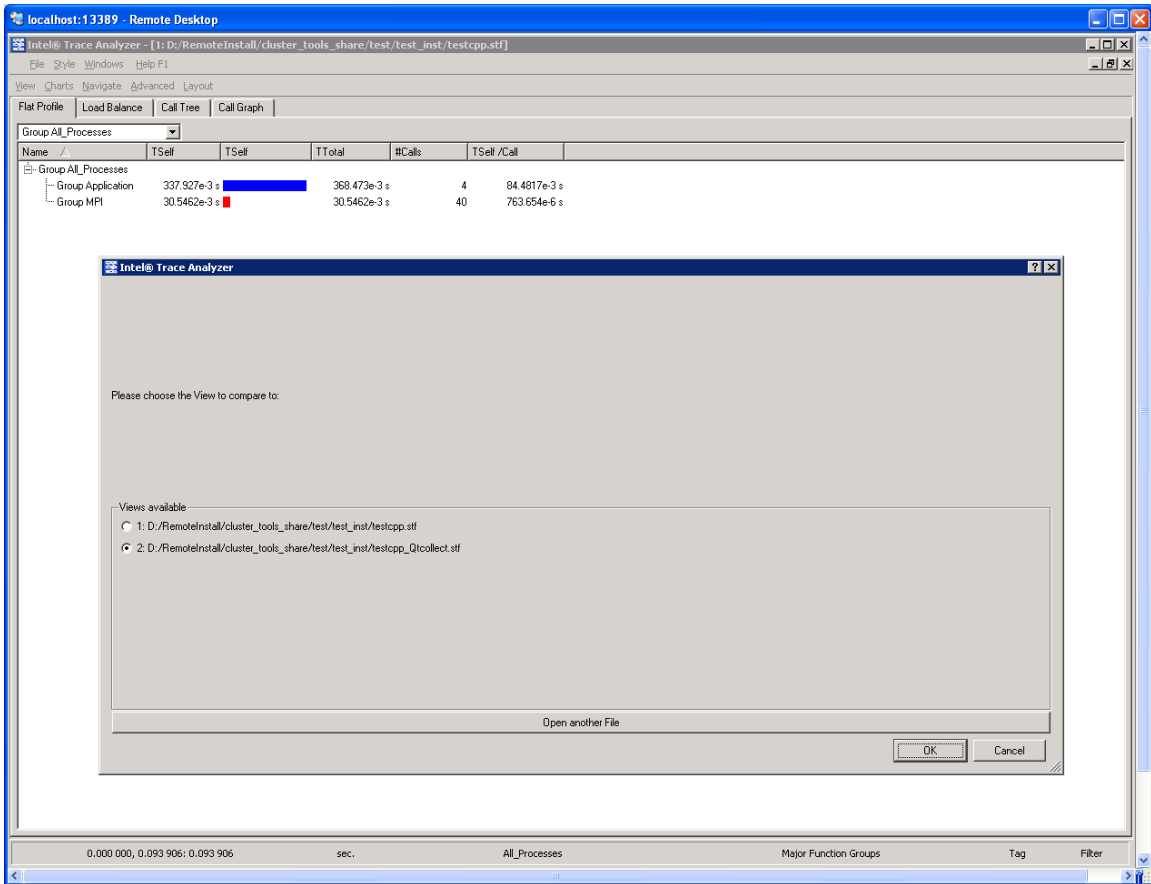


Figure 13.5 – Sub-panel Activating the Second STF File for Comparison

Click on the Ok button in Figure 13.5 and the comparison display in Figure 13.6 will appear. In Figure 13.6, notice that the timeline display for `testcpp_Qtcollect.stf` (i.e. the second timeline) is longer than that of the top timeline display (`testcpp.stf`).

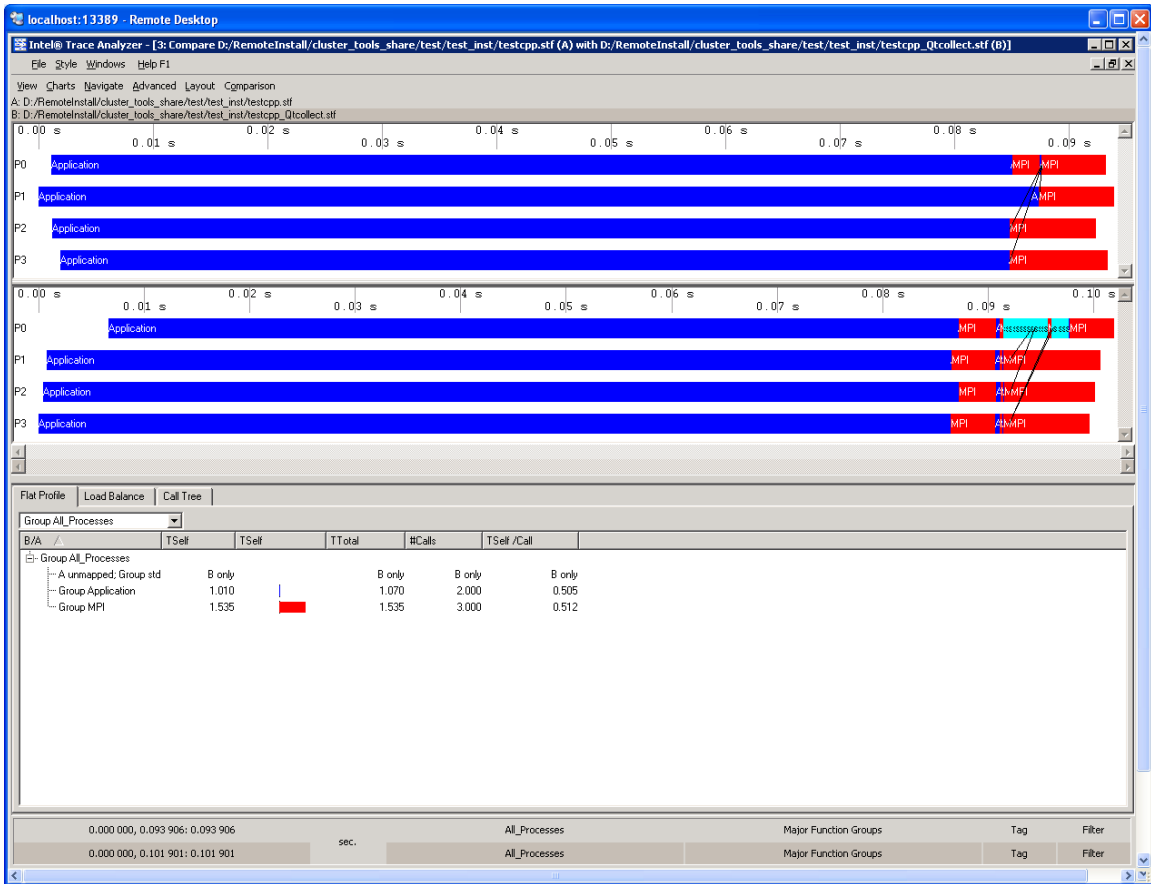


Figure 13.6 – Comparison of testcpp.stf and testcpp_Qtcollect.stf

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

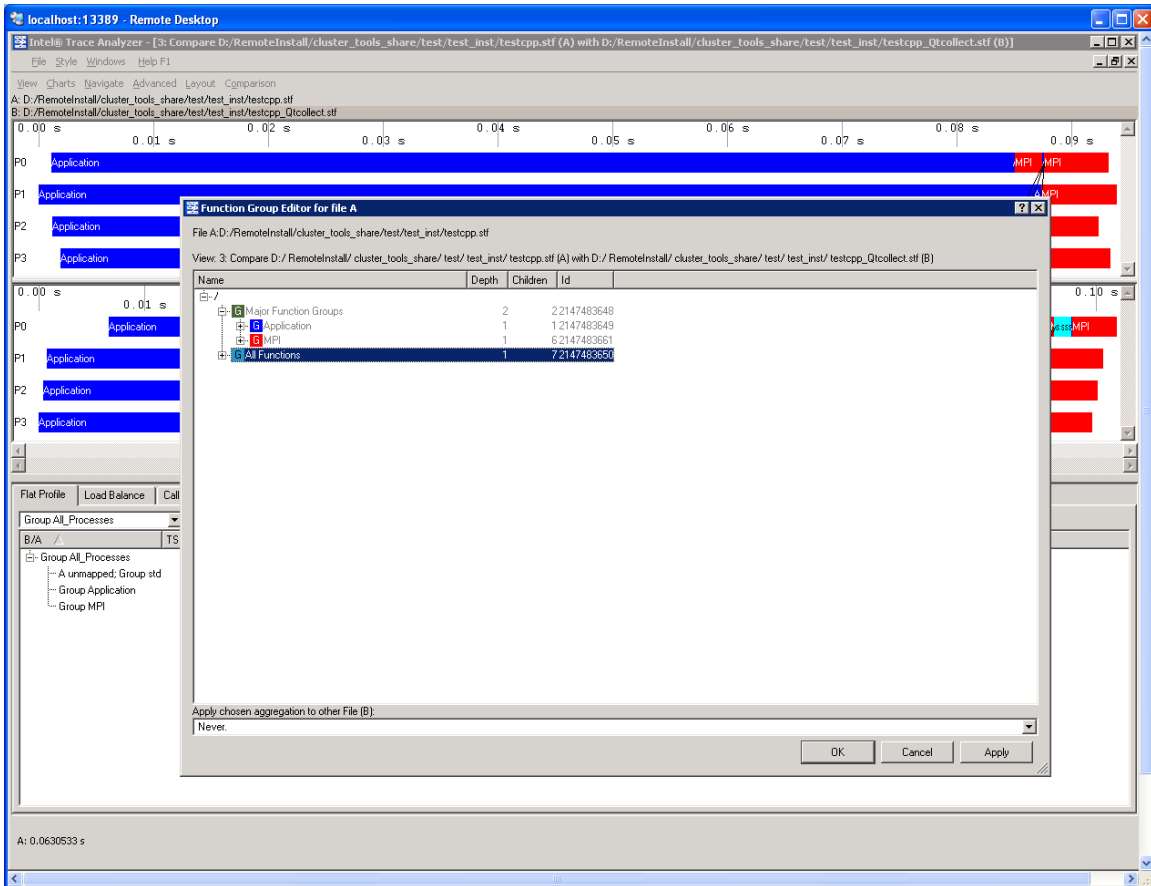


Figure 13.7 – “Function Group Editor for file A” Display (i.e for file testcpp.stf)

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

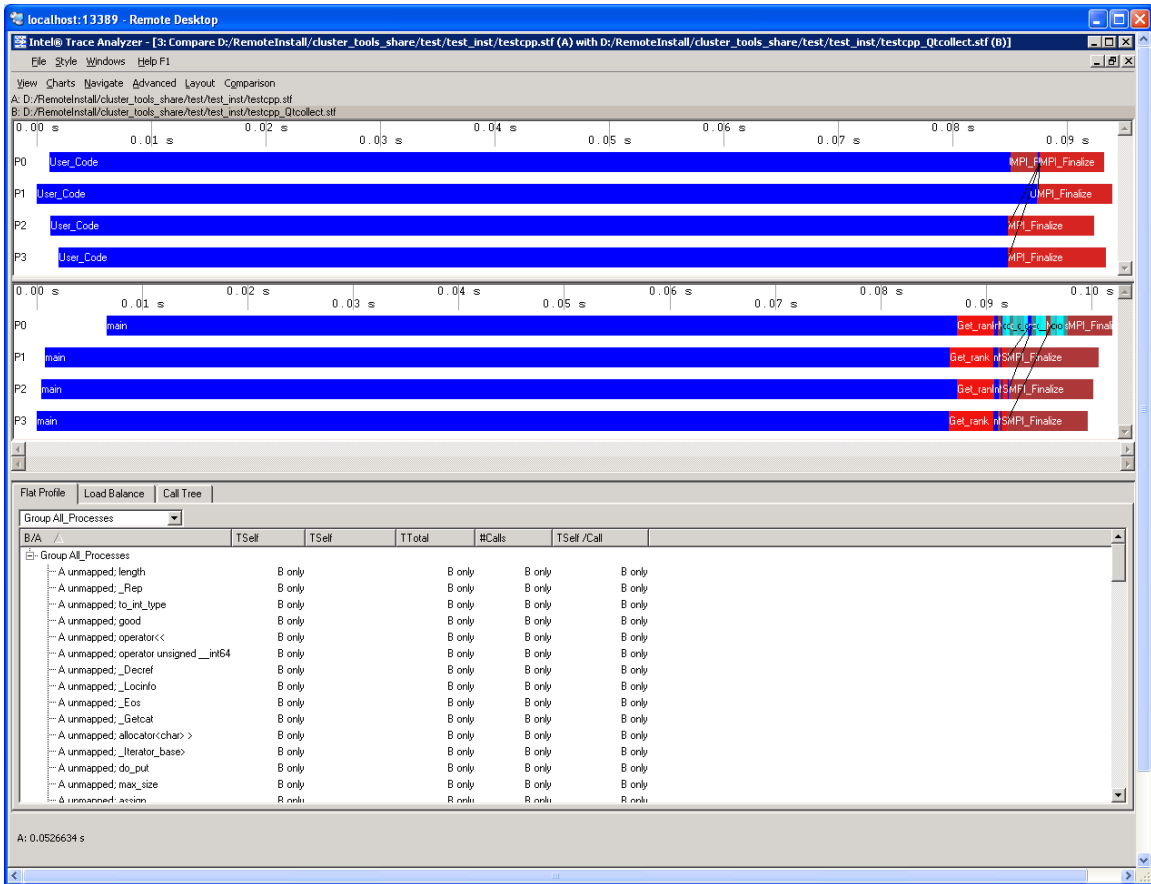


Figure 13.8 – Comparison of STF Files `testcpp.stf` and `testcpp_tcollect.stf` after making the All Functions Selection

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

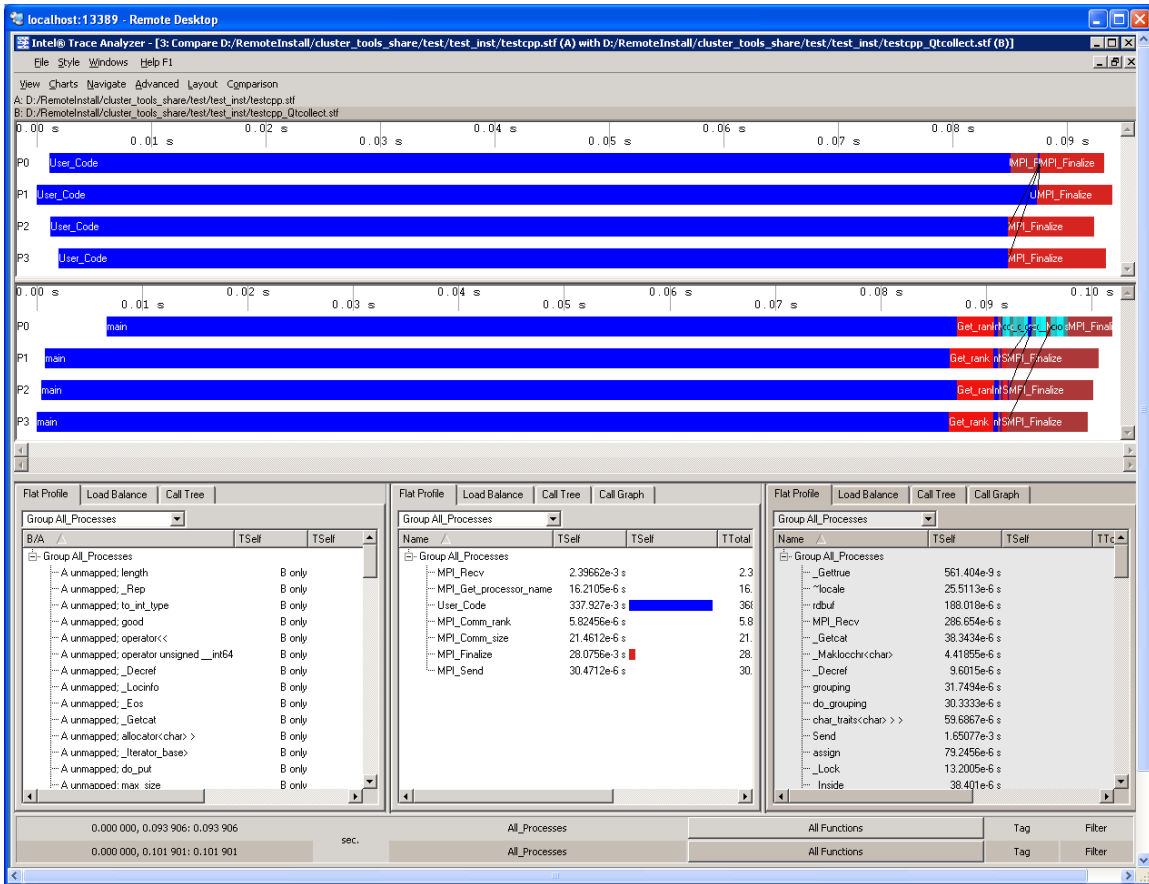


Figure 13.9 – Function Profile Sub-panels in the Lower Middle and Lower Right Sections of the Display for testcpp.stf and testcpp_Qtcollect.stf

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

- 1) testc.stf and testc_Qtcollect.stf
- 2) testf.stf and testf_Qtcollect.stf
- 3) testf90.stf and testf90_Qtcollect.stf

14. Using Cluster OpenMP*

Cluster OpenMP is only available on Linux platforms. The Intel® architectures must be Intel® 64 or IA-64.