



Intel Virtual Interface (VI) Architecture Performance Suite User's Guide

Preliminary Version V0.3
December 16, 1998



DISCLAIMERS

THIS INTEL VIRTUAL INTERFACE ARCHITECTURE PERFORMANCE SUITE USER'S GUIDE ("USER'S GUIDE") IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR ANY OTHER RIGHTS OF THIRD PARTIES OR OF INTEL, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY SPECIFICATION, DOCUMENTATION, SOFTWARE OR OTHER MATERIALS REFERENCED HEREIN.

Nothing in this document constitutes a guarantee, warranty or license to any intellectual property right, express or implied, by estoppel or otherwise. Intel makes no representations or warranties and specifically disclaims all liability as to the User's Guide, the test suite software ("Software"), or specifications and the information and test cases contained therein with respect to: (i) liability for infringement of any proprietary rights, including without limitation, intellectual property rights; (ii) sufficiency, reliability, accuracy, completeness or usefulness of same; and (iii) ability or sufficiency of same to function accurately as a representation of any standard. Furthermore, Intel makes no commitment to update the information contained in the foregoing items, and Intel reserves the right to make changes at any time, without notice, to the User's Guide, the Software, the test cases, or the test specification.

Performance tests are designed to yield approximate performance results for products tests, by measuring specific computer systems and/or specific components. Any differences in system hardware or software design or configuration may affect the performance results, as well as the actual performance of the computer systems and/or specific components. Potential purchasers of computer systems and/or components should be (i) informed as to the specific hardware and software design or configuration used in obtaining specific performance results, and (ii) advised to consult other sources of information to evaluate the performance of systems or components prior to purchase.

Third parties may have intellectual property rights which may be relevant to this document and the technologies discussed herein, accordingly the reader is advised to seek the advise of competent counsel as required, without obligation to Intel.

Copyright © Intel Corporation 1998.

AlertVIEW, i960, iCOMP, iPSC, Indeo, Insight960, Intel, Intel Inside, Intercast, LANDesk, MCS, NetPort, OverDrive, Pentium, ProShare, SmartDie, Solutions960, the Intel logo, the Intel Inside logo, and the Pentium Processor logo are registered trademarks of Intel.

BunnyPeople, CablePort, Celeron, Connection Advisor, Intel Create & Share, EtherExpress, ETOX, FlashFile, i386, i486, InstantIP, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel® InBusiness, Intel® StrataFlash, Intel® TeamStation, MMX, NetportExpress, Paragon, Pentium® II Xeon, Performance at Your Command, RemoteExpress, StorageExpress, SureStack, The Computer Inside, TokenExpress, the Indeo logo, the MMX logo, the OverDrive logo, the Pentium OverDrive Processor logo, and the ProShare logo are trademarks of Intel.

Intel® AnswerExpress, Mediadome, and PC DADS are service marks of Intel.

*All other brands and names are property of their respective owners.

SOFTWARE LICENSE AGREEMENT

BY INSTALLING OR USING THIS SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. DO NOT INSTALL OR USE UNTIL YOU HAVE CAREFULLY READ AND AGREED TO THE FOLLOWING TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ANY ACCOMPANYING ITEMS.

LICENSE. Intel Corporation ("Intel") grants You the nonexclusive, royalty-free right to install and use the Intel Virtual Interface Architecture Performance Suite software programs ("Software"), pursuant to the terms and conditions set forth below:

YOU MAY:

1. Install, use, and copy the Software.
2. Make a reasonable number of backup and/or archival copies of the Software.
3. Publish the results derived from such use ("Results") and use, reproduce and distribute such Results pursuant to the following limitations: (i) The Results derived from each utilization of the Software (i.e., the "Test Report") must be published in its entirety, (ii) Each Test Report must include the complete disclaimers set forth in Exhibit A hereto, and (iii) Each Test Report must reference: (1) the specific version of the Software from which it was derived, (2) the performance test Results, (3) the specific hardware and software design or system configuration(s) used, and (4) complete disclosure as to the type and consistency of the test conditions associated with each Test Report.

YOU MAY NOT:

1. Copy, modify, rent, sell, distribute or transfer any part of the Software except as provided in this Agreement, and You agree to prevent unauthorized copying of the Software.
2. Reverse engineer, decompile, or disassemble the Software.
3. Sublicense the Software.

OWNERSHIP OF SOFTWARE AND COPYRIGHTS. Title to all copies of the Software remains with Intel or its suppliers. The Software is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions. You may not remove any copyright notices from the Software. Intel may make changes to the Software, or to items referenced therein including without limitation, the test specifications, at any time without notice, but is not obligated to support or update the Software. Except as otherwise expressly provided, Intel grants no express or implied right under Intel patents, copyrights, trademarks, or other intellectual property rights. You may transfer the Software only if the recipient agrees to be fully bound by these terms and if You retain no copies of the Software.

THIRD PARTY SOFTWARE. Third party software (e.g., drivers, utilities, operating system components, etc.) which You may use in connection with use of the licensed Software is subject to the third party licenses supplied with such software. Intel expressly disclaims liability of any kind with respect to Your installation or use of third party software.

LIMITED MEDIA WARRANTY. If the Software has been delivered by Intel on physical media, Intel warrants the media to be free from material physical defects for a period of ninety days after delivery by Intel. If such a defect is found, return the media to Intel for replacement or alternate delivery of the Software as Intel may select.

EXCLUSION OF OTHER WARRANTIES. EXCEPT AS PROVIDED ABOVE, THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING, WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF PROPRIETARY OR INTELLECTUAL PROPERTY RIGHTS, FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF THE SPECIFICATION. Intel does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the Software.

LIMITATION OF LIABILITY. IN NO EVENT SHALL INTEL OR ITS SUPPLIERS BE LIABLE TO ANY PARTY FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, LOST PROFITS, BUSINESS INTERRUPTION, COMPUTER FAILURE OR MALFUNCTION, OR LOST INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. INTEL DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS CONCERNING (I) THE USE OF THE SOFTWARE OR ANY ACCOMPANYING WRITTEN MATERIALS, INCLUDING THE USER'S GUIDE, IN TERMS OF SUFFICIENCY, ACCURACY, RELIABILITY, COMPLETENESS OR USEFULNESS OF SAME, AND/OR THE ABILITY OR SUFFICIENCY OF SAME TO FUNCTION ACCURATELY AS A REPRESENTATION OF ANY STANDARD, (II) THE USE OF, DISTRIBUTION OF OR RELIANCE UPON THE TEST REPORTS AND/OR THE RESULTS, OR (III) THE

VIRTUAL INTERFACE ARCHITECTURE SPECIFICATION, THE INTEL VIRTUAL INTERFACE ARCHITECTURE DEVELOPER'S GUIDE, USER'S GUIDE, OR ANY OTHER MATERIALS RELATED TO THE IMPLEMENTATION OF THE SOFTWARE OR INTERPRETATION OF RESULTS. YOU REMAIN SOLELY RESPONSIBLE FOR THE DESIGN, SALE AND FUNCTIONALITY OF YOUR PRODUCT(S), INCLUDING, WITHOUT LIMITATION, ANY LIABILITY ARISING FROM PRODUCT LIABILITY OR PRODUCT INFRINGEMENT. SOME JURISDICTIONS PROHIBIT EXCLUSION OR LIMITATION OF LIABILITY FOR IMPLIED WARRANTIES OR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM JURISDICTION TO JURISDICTION.

INDEMNIFICATION. Intel does not endorse, support or guarantee the Results obtained using the Software. Performance tests are designed to yield approximate performance results for products tests, by measuring specific computer systems and/or specific components. Any differences in system hardware or software design or configuration may affect the performance results, as well as the actual performance of the computer systems and/or specific components. You are advised to consult other sources of information to evaluate the performance of systems or components prior to purchase. You agree to defend, indemnify and hold Intel harmless from and against any and all actions, claims, damages, expenses (including reasonable attorney's fees) and liabilities arising from Your use of the Software and use or distribution of the Test Reports and Results.

TERMINATION OF THIS AGREEMENT. Intel may terminate this Agreement at any time if You violate its terms. Upon termination, You will immediately Discontinue use of the Software and destroy any copies of the Software or return all copies of the Software to Intel. Appropriate provisions shall survive termination of this Agreement.

APPLICABLE LAWS. Claims arising under this Agreement shall be governed by the laws of California, excluding its principles of conflict of laws and the United Nations Convention on Contracts for the Sale of Goods. You may not export the Software in violation of applicable export laws and regulations. Intel is not obligated under any other agreements unless they are in writing and signed by an authorized representative of Intel.

GOVERNMENT RESTRICTED RIGHTS. The Software is provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 *et seq.* or its successor. Use of the Software by the Government constitutes acknowledgement of Intel's proprietary rights therein. Contractor or Manufacturer is Intel Corporation, 2200 Mission College Blvd., Santa Clara, CA 95052.

SEVERABILITY. The terms and conditions stated in this Agreement are declared to be severable. If any paragraph, provision, or clause in this Agreement shall be found or held to be invalid or unenforceable in any jurisdiction in which this Agreement is being performed, the remainder of this Agreement shall remain valid and enforceable.

COMPLETE AGREEMENT. This is the complete any exclusive statement of this Agreement between You and Intel. This Agreement supersedes any and all prior agreement or understandings, written or oral, with respect to the subject matter of this Agreement.

EXHIBIT A

THIS TEST REPORT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR ANY OTHER RIGHTS OF THIRD PARTIES OR OF INTEL, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY SPECIFICATION, DOCUMENTATION, SOFTWARE OR OTHER MATERIALS REFERENCED HEREIN.

Nothing in this document constitutes a guarantee, warranty or license to any intellectual property right, express or implied, by estoppel or otherwise. Intel makes no representations or warranties and specifically disclaims all liability as to the Test Report, the test suite software ('Software') which generated the Test Report, specifications referenced thereby, or information and test cases contained therein, with respect to: (i) liability for infringement of any proprietary rights, including without limitation, intellectual property rights; (ii) sufficiency, reliability, accuracy, completeness or usefulness of same; and (iii) ability or sufficiency of same to function accurately as a representation of any standard. Furthermore, Intel makes no commitment to update the information contained in any of the foregoing items, and Intel reserves the right to make changes at any time, without notice, and without limitation, to the Software, the test cases, or the test specification.

Performance tests are designed to yield approximate performance results for products tests, by measuring specific computer systems and/or specific components. Any differences in system hardware or software design or configuration may affect the performance results, as well as the actual performance of the computer systems and/or specific components. Potential purchasers of computer systems and/or components are advised to consult other sources of information to evaluate the performance of systems or components prior to purchase.

Third parties may have intellectual property rights which may be relevant to this document and the technologies referenced herein, accordingly the reader is advised to seek the advise of competent counsel as required, without obligation to Intel.

Table of Contents

1. Introduction	1
1.1. Overview	1
1.2. Related Documents	1
1.3. System Requirements	1
1.3.1. Hardware	1
1.3.2. Software	2
1.4. Limits and Assumptions	2
1.4.1. Scalability	2
1.4.2. Required Conformance Level	2
2. Installation and Quick Start	3
3. Test Harness	4
3.1. Overview	4
3.2. Node Communication	4
3.3. Detailed Description	4
3.4. Test Harness Usage	7
3.5. Test Harness Customization	8
3.6. Analyzing Test Results	8
3.6.1. Build Failures	8
3.6.2. Re-running Test Cases Manually	8
3.7. Vipperf Directory Hierarchy	9
4. Test Case Summary	10
4.1. alltoall	12
4.2. alltoone	13
4.3. component	14
4.4. fanin	16
4.5. pingpong	17
4.6. randalltoall	18
4.7. roundrobin	19
4.8. stream1	20
4.9. stream2	21
Appendix A. Vipperf Environment Tested	22

1. Introduction

1.1. Overview

The *Virtual Interface Architecture Specification* describes the architecture for a high bandwidth/low latency interface between high performance network hardware and computer systems. The *VI Architecture Specification* is primarily intended for network hardware vendors and operating system vendors, rather than application programmers.

In the interest of promoting rapid support for the VI Architecture and a uniform Application Program Interface (API) to VI, Intel has developed the *Intel Virtual Interface (VI) Architecture Developer's Guide*. It describes the Virtual Interface Provider Library (VIPL) and the VI Kernel Agent, which is based on the example VI User Agent in Appendix A of the *VI Architecture Specification*.

Some application communication patterns are very latency sensitive while others demand high bandwidth. VI is designed to encourage both low latency and high bandwidth within the same implementation. This Intel VI Architecture Performance Suite is a tool intended to enhance the accurate measurement of representative communication patterns in a VIPL implementation. It includes typical patterns from commercial and scientific applications that are either latency or bandwidth sensitive. A variety of communication patterns are provided, allowing a meaningful measurement of interconnect performance as communication needs vary by type of application, and even within various stages of an application.

1.2. Related Documents

Information on the Virtual Interface Architecture is available from the following documents:

- *Virtual Interface Architecture Specification*. Compaq/Intel/Microsoft; Revision 1.0, December 16, 1997 (available at http://www.viarch.org/html/Spec/vi_specification_version_10.htm).
- *Intel Virtual Interface (VI) Architecture Developer's Guide*. Intel; Revision 1.0, September 9, 1998 (available at http://www.intel.com/design/servers/vi/developer/ia_imp_guide.htm).
- *Intel Virtual Interface (VI) Architecture Developer's Guide Error Table Supplement*. Intel; Revision 1.0, September 3, 1998 (at http://www.intel.com/design/servers/vi/developer/ia_imp_guide.htm).

1.3. System Requirements

1.3.1. Hardware

2 (or more) IA systems, each containing:

- VI NIC,
- Secondary MS-NET (NT LAN Manager and TCP/IP) interconnect.

For some tests (such as fanin and alltoall), at least 4 nodes are required for meaningful results.

1.3.2. Software

Each system in the cluster to be tested must contain:

- Microsoft Windows NT Version 4.0 + Service Pack 3,
- TCP/IP networking installed and configured,
- RSHD (Rsh service daemon),
- VI NIC run-time software (DLL files).

The system running the test harness must also contain:

- NMAKE and C compiler (to process makefiles and compile Win32 C source files),
- Perl interpreter for NT,
- VI NIC development software (vipl.h and vipl.lib).

Additional information about the C compiler, Perl interpreter and RSHD service that were used to develop and test the Performance Suite is in Appendix A.

1.4. Limits and Assumptions

1.4.1. Scalability

A minimum of two nodes is required. Tests are designed to scale with the number of nodes in the cluster.

Performance Suite tests are designed for clusters of up to 16 nodes, but have only been tested on up to two nodes at release time. In a system with multiple NICs, only one NIC will be tested at a time.

1.4.2. Required Conformance Level

The *Intel VI Architecture Developer's Guide* has defined three phases of conformance to the *Virtual Interface Architecture Specification*, named as follows: Early Adopter, Functional and Full Conformance. The Early Adopter phase has been broadly defined as the set of VIPL API calls and other functionality in the API required for demonstrations and application prototyping. The Functional phase contains the additional functions that ISVs plan to use to develop products. Finally, Full Conformance is defined to be the full set of API and functions defined in the *Intel VI Architecture Developer's Guide*.

In general, the performance tests require Functional Conformance. Specifically, VI name service and peer-to-peer connections are used.

2. Installation and Quick Start

- Install Windows NT, C compiler, Perl interpreter, RSHD service and NIC software. Make sure that the system environment variables PATH, LIB and INCLUDE reflect the installations correctly.

*Additional information about the C compiler, Perl interpreter and RSHD service that were used to develop and test the Performance Suite is in Appendix A. These are **not** supplied with the Performance Suite.*

- Install the Performance Suite on a server drive accessible to each VI test system. The files can be installed on either a shared drive on one of the test systems, or on another server that can be mapped by each system in the cluster to be tested.

*Run **vipperf3.exe** and follow the instructions in the program. It will install the test cases and harness, and set the system environment variable VIPPERF_HOME to the "root" of the Performance Suite tree. While it is not required that you reboot the system after installing the Performance Tests, note that required environment variables will not be updated until your next login.*

- Edit the file %VIPPERF_HOME%\bin\netaddr to contain the VI NIC address followed by the NT LAN Manager network name of each test system in the cluster, e.g.:

```
00.aa.00.12.34.56      system1
00.aa.00.78.9a.bc     system2
```

See Section 3.2 for information on using VIPL Name Service names instead of physical NIC addresses.

- Edit %VIPPERF_HOME%\include\vipperf.h as necessary (see directions in the file).

At a minimum, you will need to set the NIC Name here, but check the others as well

- Run the Performance Suite, in a Command Prompt window, e.g.:

```
cd %VIPPERF_HOME%
perl bin\Vipperf.pl /t testlist<n>
```

See Section 3.3 for details on running the Performance Suite.

- To uninstall the Performance Suite, first remove all generated files (logs, executables and other intermediate files) with:

```
cd %VIPPERF_HOME%
perl bin\Vipperf.pl /t testlist /uninstall
```

Then, uninstall the Performance Suite with the Control Panel's Add/Remove Programs icon.

Select "Intel Virtual Interface Architecture Performance Suite" and press Add/Remove.

3. Test Harness

3.1. Overview

The VI Performance Suite is Perl script driven. The file %VIPPERF_HOME%\bin\Vipperf.pl is the Perl script responsible for establishing the test environment, building the tests, and then running the test cases on all nodes specified in the file %VIPPERF_HOME%\bin\netaddr. This work is done by the “master node”.

3.2. Node Communication

A Network Interface Controller configured with TCP/IP transport is used for test harness inter-node communication. If the VI NIC supports a TCP/IP interface, you may not need a secondary interconnect, but you should be aware that the test harness will consume VI resources that may affect reported results. The mechanism used for running test cases on cluster nodes is the RSH (remote shell) command, where each cluster node has mapped the shared drive containing the performance test files with the NT LAN Manager NET USE commands.

The file %VIPPERF_HOME%\bin\netaddr is used by the harness to determine the network name of each cluster node, and by test cases to determine the NIC addresses of the other cluster nodes. This text file contains one entry for each cluster node, and must be configured by the user before running the test harness. Each line contains a VI NIC address followed by the NT LAN Manager network name of each test system in the cluster, e.g.:

```
00.aa.00.12.34.56      system1
00.aa.00.78.9a.bc     system2
```

You may provide a VIPL Name Service name instead of a VI NIC address. The VIPL name must be enclosed in quotation marks, e.g.:

```
"ViplTest1"          system1
"system2"            system2
```

When specified in this format, tests will use *VipNSInit()* and *VipNSGetHostByName()* to obtain the default NIC address for the specified systems. Note that the VIPL Name Service name may or may not be the same as the NT LAN Manager network name; for that reason both must be supplied, as shown above.

3.3. Detailed Description

This section describes how the test harness works. The general algorithm is as follows:

Harness system “Master node”	Cluster node(s)
Build required libraries and tools	<i>Wait for work (RSH service)</i>
Share test directories (if %VIPPERF_HOME% is on a local file system) <i>net share . . .</i>	
Map test directories on all Cluster nodes	<i>net use . . .</i>
For each test case: (either a specified text file list or the test in the current directory if not specified)	
Chdir to test directory	

Issue nmake to build test <i>log.nmake</i>	
Invoke test remotely, via RSH, on each Cluster node, and locally if harness is executing on a Cluster node.	Test executable invoked
Execute test.	Execute test.
Synchronize with all Cluster nodes	<i>Wait for work (RSH service)</i>
Until all tests run	

The “master node” runs the performance tests within the harness framework. When Vipperf.pl is launched on the master node, it sets up the test environment, including:

- building the test libraries in %VIPPERF_HOME%\lib,
- building other binaries used by the harness in %VIPPERF_HOME%\bin,
- discovering the characteristics of the %VIPPERF_HOME% drive (i.e. is it a UNC mapped drive or a local drive, etc.),
- generating a list of host names in the cluster from %VIPPERF_HOME%\bin\netaddr, and mapping %VIPPERF_HOME% on the cluster nodes where it is not already mapped, and
- parsing the testlist.

Once the environment is set up, the harness builds a test using “nmake”, and then launches the test on each node in the cluster via an “rsh” command. The test harness prints out various messages with time stamps to monitor the progress of the tests. After all the tests have completed, the results are tabulated and printed to the screen. See Section 4 for information on interpreting and investigating test failures.

An example of a small test run is shown below. *The lines in italics appear only if the Conformance Suite is installed on a file system local to the Master Node*

```
P:\Vipperf\tests\alltoall>..\..\bin\vipperf.pl /home p:\vipperf /rel 1
----- 07/24/1998 3:51p: Creating VIPPERF_HOME share -----
vipperf was shared successfully.
----- 07/24/1998 3:51p: Building vipperf utilities -----
----- 07/24/1998 3:51p: Checking VIPPERF_HOME on remote nodes -----
----- 07/24/1998 3:51p: Checking remote node vi-sdk2 -----
System error 1326 has occurred.

Logon failure: unknown user name or bad password.
Remote UNC name not found on vi-sdk2: sending 'net use' command
Please enter domain and username (domain\username): mydomain\myid
Please enter password:
----- 07/24/1998 3:51p: Checking remote node vi-sdk1 -----
System error 1326 has occurred.

Logon failure: unknown user name or bad password.
Remote UNC name not found on vi-sdk1: sending 'net use' command
----- 07/24/1998 3:51p: Building vipperf.lib -----
----- 07/24/1998 3:51p: Building alltoall -----
----- 07/24/1998 3:51p: Starting test: alltoall on host: vi-sdk2 -----
----- 07/24/1998 3:51p: Starting test: alltoall on host: vi-sdk1 -----
----- 07/24/1998 3:51p: Running test: alltoall on host: vi-sdk2 -----
Summary Output
-----
0) Test:          \\XXXX\myid$\vipperf\tests\alltoall\alltoall.exe
1) Device:       Mydevice, version Hardware 1 : Provider 1
2) Node Name:    vi-sdk2
3) Node Number: 1
```

```
4) Iterations:      10
5) Buffer Size:     1024 bytes
6) Data Segments:  1 (1024 bytes each)
7) Call type:      polling
8) Alignment:      64 bytes
10) Elapsed Time:  XX sec
11) Commun. Time:  XX sec
12) Total Band:    XX Mbit/sec
13) Windows:       1
14) Nodes:         2
```

Performance Metrics (thread 0)

```
-----
15) Avg Iter Time: XX usec
16) Avg Latency:   XX usec
17) Avg Bandwidth: XX Mbit/sec
```

Performance Metrics (thread 1)

```
-----
18) Avg Iter Time: XX usec
19) Avg Latency:   XX usec
20) Avg Bandwidth: XX Mbit/sec
```

----- 07/24/1998 3:51p: Running test: alltoall on host: vi-sdk1 -----

Summary Output

```
-----
0) Test:          \\XXXX\myid$\vipperf\tests\alltoall\alltoall.exe
1) Device:        Mydevice, version Hardware 1 : Provider 1
2) Node Name:     vi-sdk1
3) Node Number:   1
4) Iterations:    10
5) Buffer Size:   1024 bytes
6) Data Segments: 1 (1024 bytes each)
7) Call type:    polling
8) Alignment:    64 bytes
10) Elapsed Time: XX sec
11) Commun. Time: XX sec
12) Total Band:  XX Mbit/sec
13) Windows:     1
14) Nodes:       2
```

Performance Metrics (thread 0)

```
-----
15) Avg Iter Time: XX usec
16) Avg Latency:   XX usec
17) Avg Bandwidth: XX Mbit/sec
```

Performance Metrics (thread 1)

```
-----
18) Avg Iter Time: XX usec
19) Avg Latency:   XX usec
20) Avg Bandwidth: XX Mbit/sec
```

----- 07/24/1998 3:52p: All tests processed -----

----- 07/24/1998 3:52p: Closing connections -----

\\XXXX\myid\$ was deleted successfully.

\\XXXX\myid\$ was deleted successfully.

vipperf was deleted successfully

3.4. Test Harness Usage

The test harness is invoked as follows:

```
[perl] [drive:][%VIPPERF_HOME%\bin\]Vipperf.pl [/t <testlist>] [/home <dir>]
      [/debug] [/clean] [/build] [/uninstall] [/<testoption> [...] ]
```

The options are used as follows:

/t <testlist> is used to specify a list of tests to run.

If no testlist is supplied on the command line, then the default action is to run the test in the current directory, which may be located anywhere in the directory tree under %VIPPERF_HOME%. The typical usage model is to run without a testlist, instead running a single benchmark multiple times with different test-specific options.

A testlist consists of a list of test directories relative to the installed location of the Performance Suite %VIPPERF_HOME%. Users can create their own testlist to specify which tests are to be run, or use the default testlist supplied.

NOTE: The test directories specified in the <testlist> file are case sensitive and must use a backslash (\) as a delimiter, i.e.:

```
Correct: tests\alltoall
Incorrect: tests\AlltoAll
Incorrect: tests/alltoall
Incorrect: P:\vipperf\tests\alltoall
```

NOTE: A test directory which begins with a # is treated as a comment.

/home <dir> is used to specify the location of the Performance Suite. This switch overrides the %VIPPERF_HOME% environment variable. The <dir> can be on a local drive, in which case the drive will be "shared" and used by all other nodes under test; or it can be on a mapped network drive, in which case all nodes under test will use the UNC name of the mapped drive to run tests. *It is the responsibility of the tester to make sure the drive is accessible to the other test nodes.*

It is recommended that the user set the %VIPPERF_HOME% environment variable on the "master node" for general testing. If the /home switch is not used, the test harness uses %VIPPERF_HOME% as the location of the Performance Suite.

/debug Causes the debug versions of the tests to be built and run, and passes the /debug flag when test cases are invoked.

/clean Tells the harness to rebuild the test harness libraries, utilities, and all individual tests from source. Test cases will still be built and executed.

/build is used to build but not run the tests. /build can be used in conjunction with /clean to clean and rebuild the test harness library and tests.

/uninstall is used to remove all generated files (logs, executables and other intermediate files). Test cases are not built or executed.

/<testoption> any option not recognized by the harness is passed on to each performance test

application when invoked. See Section 4 for a list of test-specific options.

3.5. Test Harness Customization

The harness uses the output from the `net use` command to determine the success of mount operations. If the systems under test are configured for a language other than English, you may need to edit the harness. See the instructions in `%VIPPERF_HOME%\bin\Vipperf.pl`.

3.6. Analyzing Test Results

3.6.1. Build Failures

If a test fails to build, a message will be printed to the screen at the time of the build failure, and the results summary information will indicate "No results found". The output of the build commands is captured in a file in each test directory called "log.nmake". This file can be examined to determine the cause of the build failure.

3.6.2. Re-running Test Cases Manually

In some cases, it may be desirable to run individual test cases manually without the harness. To do so, `cd` to the appropriate test directory on each system and invoke the test with:

```
testname [/home <dir>] [ /<testoption> [...]]
```

The options are used as follows:

/home <dir> is used to specify the location of the Performance Suite. This switch overrides the `%VIPPERF_HOME%` environment variable.

NOTE: `%VIPPERF_HOME%` is only set on the "master" node when the Performance Suite is installed. It is supplied by the harness when tests are invoked on remote nodes, and must be supplied by the user when running manually.

/<testoption> See Section 4 for a list of test-specific options.

Should the need arise to compile a test case outside the harness environment, the following environment should be established for each test case:

- The `INCLUDE` environment variable should include the directory path containing the Performance Suite include file `vipperf.h`, which is in `%VIPPERF_HOME%\include`, and the path containing the VI Provider Library include file `vipl.h`.
- The `LIB` environment variable should include the path containing the Performance Suite library `vipperf.lib`, which is in `%VIPPERF_HOME%\lib`, and the path containing the VI Provider Library `vipl.lib`.

In a Command Prompt window, change the present working directory to the test directory. To build a release version of the test, invoke `nmake` as follows:

```
nmake /f test.mak
```

To build a debug version of the test, use:

```
nmake /f test.mak CFG="<testname> - Win32 Debug"
```

3.7. Vipperf Directory Hierarchy

The VI Performance Suite is installed in the following directory tree:

```
%VIPPERF_HOME% \bin           netaddr file, test harness, and other tools
                  \lib         performance test library
                  \include      include files
                  \tests
                        \test1
                        \test2
                        ... \testn
```

Test cases are maintained in a tree structure, with one test case per directory. Each test directory contains:

- Test source file (*test.c*), and
- NMAKE file (*test.mak*).

The following files are generated when each test case is invoked by the harness:

- Make log (*log.nmake*),
- Test executable (*test.exe* or *testd.exe*, plus intermediate files),
- Results log from each node (if /o is specified)
- Other intermediate files.

See Section 4 and the test cases themselves for more information.

4. Test Case Summary

All the performance tests (except the component test) share %VIPPERF_HOME%\lib\vipperf.lib. A lightweight main() is a part of the library, and simply invokes the different phases of the benchmarks:

- 1) start elapsed-time clock,
- 2) parse command-line arguments via the args() subroutine,
- 3) open the NIC,
- 4) check that buffersize <= MTU,
- 5) get the node Id of the machine,
- 6) call the communication loop via the loop() subroutine,
- 7) close the NIC,
- 8) stop the elapsed-time clock, and
- 9) output stats via the stats() subroutine.

The above steps are nested within a number of loops. Each loop is intended to increment some variable for the next run of the benchmark. The nesting of the loops is presently hard-coded at 4. This means that at most 4 variables can be updated for each run of the bench. For example, you could run one benchmark varying option "a" from 7 to 11 linearly by 2, option "b" from 3 to 51 exponentially by 3, and option "c" from 4 to 8 linearly by 4. You would specify the following on the command line:

```
/a 7 /aEnd 11 /incL 2 /b 3 /bEnd 51 /incE 3 /c 4 /cEnd 8 /incL 4
```

This would cause the bench to run 3x4x2 or 24 times, once for each combination of inputs:

```
run 1) a=7, b=3, c=4
run 2) a=7, b=3, c=8
run 3) a=7, b=9, c=4
run 4) a=7, b=9, c=8
run 5) a=7, b=27, c=4
run 6) a=7, b=27, c=8
run 7) a=7, b=51, c=4
run 8) a=7, b=51, c=8
run 9) a=9, b=3, c=4
<and so on...>
```

The VIPL APIs for which performance measurements are made are:

<i>VipCreateVI()</i>	<i>VipDestroyVI()</i>	
<i>VipOpenVI()</i>	<i>VipCloseVI()</i>	
<i>VipRegisterMem()</i>	<i>VipDeregisterMem()</i>	
<i>VipPostSend()</i>	<i>VipSendWait()</i>	<i>VipSendDone()</i>
<i>VipPostRecv()</i>	<i>VipRecvWait()</i>	<i>VipRecvDone()</i>

Those exercised but not timed are:

<i>VipOpenNic()</i>	<i>VipCloseNic()</i>
---------------------	----------------------

The module stats.c contains utilities for outputting runtime statistics for the benchmarks. Its main functions are:

- 1) calculate elapsed time
- 2) calculate benchmark loop times and per-iteration times along with max/min/avg. Per-iteration timings may be separately output to a file for later analysis

3) calculate latency/bandwidth using the average iteration times calculated.

If the benchmarks comprise multiple threads, stats.c outputs thread-specific statistics individually for each thread.

The most powerful utility offered by stats() is the ability for the user to specify which metrics to print. Each benchmark may output different metrics, so stats() dynamically builds a table of indices such that each metric is specified with an index. This will allow the user to generate output in a form suitable for later analysis. If a user gives no output format, then an output summary for the benchmark is given, otherwise, the specific format is used.

For example, suppose a user wanted to generate a 3-tuple of (<buffer size>, <num nodes>, <num iterations>) for later use in a surface plot. Assume that the indices of these metrics are 7, 3, and 10, respectively. The following output format could then be used:

```
/format "(#7, #3, #10)"
```

Note that perl and rsh will strip quotation marks; when using the harness, escape them with three backslashes on the command line, i.e.

```
/format \\\"(#7, #3, #10)\\\"
```

A user determines the index of a given metric by first viewing the summary output, where the index precedes the metric.

The remainder of this section summarizes each of the test programs and their available options. Additional information on each test case is available in the test source files.

4.1. *alltoall*

Each node communicates using $2*(n-1)$ threads (where 'n' is the number of nodes). The first n-1 threads are for sending, and the second n-1 are for receiving. Each thread creates and opens a VI connection to another node for either sending or receiving. Once the connections are established, every pair of connected threads streams messages, where the thread on one node is the sender and the other the receiver.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)
<code>/w <num></code>	number of windows

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., <code>5 = 0101 = output from threads 0 and 2</code>)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of <code>#<num></code> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of <code>VIPPERF_HOME</code>
<code>/rel <int></code>	<code>VIP_RELIABILITY_LEVEL</code> to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.2. *alltoone*

The root node establishes n-1 VI connections, each in a separate thread (where n is the number of nodes). The non-root nodes use a single VI connection established with the root node. Each VI connection is used to stream messages from a non-root node to the root node, and to receive acknowledgement messages back from the root node.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)
<code>/w <num></code>	number of windows
<code>/server <num></code>	index of server node

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of VIPPERF_HOME
<code>/rel <int></code>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.3. component

The Components Test is implemented through a number of modules, each module testing one or more specific VIPL API functions. The current components tested are :

<i>VipCreateVI()</i>	<i>VipDestroyVI()</i>	
<i>VipOpenVI()</i>	<i>VipCloseVI()</i>	
<i>VipRegisterMem()</i>	<i>VipDeregisterMem()</i>	
<i>VipPostSend()</i>	<i>VipSendWait()</i>	<i>VipSendDone()</i>
<i>VipPostRecv()</i>	<i>VipRecvWait()</i>	<i>VipRecvDone()</i>

The multi-node operation beyond two nodes is characterized by an all-to-all model, where threads are paired up between any two nodes to perform the selected component testing. Each pair of threads operates asynchronously to other threads within the test. By default one thread is created on each node to communicate with one thread on a remote node. More than one thread per remote node may be created by providing an option with an argument of the number of threads to create for each remote node, and the component test will be duplicated within each thread. Thus the number of threads created on each node is (threads selected) * (n-1).

NOTE: The component test is completely self-contained, and does not use the library in %VIPPERF_HOME%\lib\vipperf.lib; it contains custom versions of the routines contained in the library.

NOTE: The component tests are intended to run only on 2 nodes.

Benchmark Options

/test <name, ...> list of component tests to run
 /align <int> align memory on <int> byte boundaries
 /iter <int> number of iterations per run
 /buff <int> starting buffer size (bytes)
 /buffEnd <int> ending buffer size (bytes)
 /w <int> starting operations per window
 /wEnd <int> ending operations per window
 /key <name> char string key
 /genkey <base> char string base for generated key

Multi-run Options

/incE <int> increment var exponentially by <int>
 (e.g., /buff 2 /buffEnd 64 /incE 2)
 /incL <int> increment var linearly by <int>

Debugging Options

/debug debug mode (verbose, checks exit status)
 /debugH <int> print hash mark every <int> iterations
 /debugP <i>:<j> print <i> chars from buffer every <j> iter.

Output options

/format <format> <format> is a double-quote delimited string
 with all occurrences of #<num> replaced with the
 value of the metric on line <num> of the summary
 output (e.g., /format "total time = #9")
 /o <file> redirect output to the file <file>

Timing options

/timeIter time iterations instead of entire loop
 /timeIterF <file> time iterations and save to the file <file>

Misc. options

/home <dir> location of VIPPERF_HOME
 /rel <int> VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
 /repeat <int> repeat bench with same options <int> times
 /help print usage and exit

4.4. fanin

This test uses a fan-in/fan-out communication pattern. The structure of the communication is a binary tree, where nodes fan-in to their parent. Once the root node receives the message stream, an acknowledgement message is fanned-out through the tree to begin the next iteration.

Up to three threads are created on each node, one for the parent node, and one for each child node. The multiple threads are used only to make the VI connections, in order to prevent deadlock conditions between nodes when making the VI connections. Once the VI connections are made all the created threads exit and the main thread continues on to perform the fan-in/fan-out message communication.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)
<code>/w <num></code>	number of windows
<code>/step</code>	step through algorithm

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., <code>5 = 0101 = output from threads 0 and 2</code>)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of <code>#<num></code> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of <code>VIPPERF_HOME</code>
<code>/rel <int></code>	<code>VIP_RELIABILITY_LEVEL</code> to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.5. pingpong

This is the "roundrobin" test on two nodes

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of VIPPERF_HOME
<code>/rel <int></code>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.6. *randalltoall*

This test uses an all-to-all communication pattern where each node streams messages to the other n-1 nodes by sequentially traversing through the nodes in a random order. Each node communicates through n threads (where 'n' is the number of nodes). A single thread is used to send messages, and the remaining n-1 threads are used for receiving messages from the other n-1 nodes. A thread from the set of 'n-1' threads on a node is dedicated to receiving messages from a particular remote node.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)
<code>/w <num></code>	number of windows
<code>/seed <num></code>	random number seed

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of VIPPERF_HOME
<code>/rel <int></code>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.7. roundrobin

A single-threaded test. Messages are sent around a ring. Two VI connections are made on each node, one to the node's left neighbor, and one to the node's right neighbor in the ring. One VI connection is used to receive messages passed around the ring, the second VI connection to send the message on to the other neighbor.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of VIPPERF_HOME
<code>/rel <int></code>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.8. *stream1*

A single-threaded, two node test. One VI connection is made between the nodes. One node is the sending node, the other the receiving node. Streams of messages are sent from the sending node to the receiving node. For each received stream, the receiving node sends an acknowledgement message to the sending node.

NOTE: The stream tests are intended to run only on 2 nodes.

Benchmark Options

<code>/align <int></code>	align memory on <int> byte boundaries
<code>/iter <int></code>	number of iterations per run
<code>/block</code>	use blocking calls instead of polling
<code>/buff <int></code>	starting buffer size (bytes)
<code>/buffEnd <int></code>	ending buffer size (bytes)
<code>/w <num></code>	number of windows
<code>/host <num></code>	host to connect to
<code>/port <num></code>	port to connect on
<code>/receive</code>	receive instead of send

Multi-run Options

<code>/incE <int></code>	increment var exponentially by <int> (e.g., <code>/buff 2 /buffEnd 64 /incE 2</code>)
<code>/incL <int></code>	increment var linearly by <int>

Debugging Options

<code>/debug</code>	debug mode (verbose, checks exit status)
<code>/debugH <int></code>	print hash mark every <int> iterations
<code>/debugP <i>:<j></code>	print <i> chars from buffer every <j> iter.
<code>/debugM <int></code>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

<code>/format <format></code>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., <code>/format "total time = #9"</code>)
<code>/o <file></code>	redirect output to the file <file>

Timing options

<code>/timeIter</code>	time iterations instead of entire loop
<code>/timeIterF <file></code>	time iterations and save to the file <file>

Misc. options

<code>/home <dir></code>	location of VIPPERF_HOME
<code>/rel <int></code>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
<code>/repeat <int></code>	repeat bench with same options <int> times
<code>/help</code>	print usage and exit

4.9. stream2

A two node, multi-threaded test. Each node creates two threads. Each thread creates a VI connection to its corresponding thread on the opposite node. One VI connection is used to stream messages in one direction, and the second VI connection is used to stream messages in the opposing direction. As with the "stream1" test, acknowledgements are sent to the sending thread once the receiving thread has received a message stream and is ready for the next stream.

NOTE: The stream tests are intended to run only on 2 nodes.

Benchmark Options

/align <int>	align memory on <int> byte boundaries
/iter <int>	number of iterations per run
/block	use blocking calls instead of polling
/buff <int>	starting buffer size (bytes)
/buffEnd <int>	ending buffer size (bytes)
/w <num>	number of windows
/host <num>	host to connect to
/port <num>	port to connect on

Multi-run Options

/incE <int>	increment var exponentially by <int> (e.g., /buff 2 /buffEnd 64 /incE 2)
/incL <int>	increment var linearly by <int>

Debugging Options

/debug	debug mode (verbose, checks exit status)
/debugH <int>	print hash mark every <int> iterations
/debugP <i>:<j>	print <i> chars from buffer every <j> iter.
/debugM <int>	mask for thread output (e.g., 5 = 0101 = output from threads 0 and 2)

Output options

/format <format>	<format> is a double-quote delimited string with all occurrences of #<num> replaced with the value of the metric on line <num> of the summary output (e.g., /format "total time = #9")
/o <file>	redirect output to the file <file>

Timing options

/timeIter	time iterations instead of entire loop
/timeIterF <file>	time iterations and save to the file <file>

Misc. options

/home <dir>	location of VIPPERF_HOME
/rel <int>	VIP_RELIABILITY_LEVEL to test (default is Reliable Delivery).
/repeat <int>	repeat bench with same options <int> times
/help	print usage and exit

Appendix A. Vipperf Environment Tested

The Performance Suite was developed and tested using the following tools:

- **C compiler**

Test cases and binary components of the harness were built and tested with Microsoft* Visual C++ V5.0. Visual C++ is *not* supplied with the performance tests.

- **Perl**

Harness scripts were developed and tested using ActiveState Perl for Win32, Build 315. Perl is *not* supplied with the performance tests; it may be obtained, subject to the GNU General Public License, at <http://www.activestate.com>.

- **RSH service daemon**

The test harness uses the Windows NT RSH.EXE command on the harness master node to execute test programs on remote nodes. The Performance Suite was developed and tested using the Remote Shell Daemon for Windows NT/95 Version 1.6 on the other cluster nodes, which is archived at the Swedish University Network (SUNET) FTP archive. RSHD is *not* supplied with the performance tests; it may be obtained at <http://ftp.sunet.se/ftp/pub/pc/Windows95/mirror-indstate/Daemons/RSHD/>.

The servers used for testing were all members of the same NT workgroup. The RSHD service was installed and started as the "Administrator" user, with an entry for the "master" node in the ".rhosts" file, on each server. Reference the *Remote Shell Daemon for Windows NT/95 Version 1.6* documentation for further details.