Windows HPC Server 2008

Using Microsoft Message Passing Interface (MS MPI)

White Paper

Eric Lantz

Senior Program Manager Lead, Microsoft Corporation

Updated: June 2008

This document was developed prior to the product's release to manufacturing, and as such, we cannot guarantee that all details included herein will be exactly as what is found in the shipping product.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, Microsoft System Center, Visual Studio, SQL Server, Microsoft Exchange, Windows, and Windows Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Microsoft

Contents

Introduction

Windows® HPC Server 2008 is a two-DVD package that includes Windows Server® 2008, HPC Server Edition on the first DVD and Microsoft® HPC Pack 2008 on the second DVD. HPC Pack 2008 includes a job scheduler, a new high performance Message Passing Interface (MPI) stack, and a new management application that uses the Microsoft® System Center interface to integrate node imaging and deployment, diagnostics, reporting, and cluster management, giving you the tools you need to bring high performance computing (HPC) to readily available x64-based computers.

What Is MPI?

Message Passing Interface (MPI) software is a portable, flexible, vendor- and platform-independent standard for messaging on HPC nodes. MPI is the specification, and Microsoft MPI (called *MS MPI*), MPICH2, and others are implementations of that standard. MPI2 is an extension of the original MPI specification.

Fundamentally, MPI is the interconnection between nodes on an HPC cluster. MPI ties nodes together, providing a portable and powerful inter-process communication mechanism that simplifies some of the complexities of communication between hundreds or even thousands of processors working in parallel.

MPI consists of two essential parts: an application programming interface (API) that supports more than 160 functions, and a launcher that enables users to execute MPI applications.

Comparing MS MPI with MPICH2

The best known implementation of the MPI specification is the MPICH2 reference implementation maintained (and largely created) by Argonne National Laboratory. MPICH2 is an open-source implementation of the MPI2 specification that is widely used on HPC clusters. For more information about MPI2, see the MPI2 specification on the MPI Forum Web site.

MS MPI is based on and designed for maximum compatibility with the MPICH2 reference implementation. The exceptions to compatibility with MPICH2 are on the job launch and job management side of MPICH2; the APIs that independent software vendors (ISVs) use are identical to MPICH2. MS MPI exceptions to the MPICH2 implementation were necessary to meet the strict security requirements of Windows HPC Server 2008. MS MPI includes a complete set of MPI2 functionality as implemented in MPICH2 with the exception of dynamic process spawn and publishing, which may be included in a later release.

Note: When using Windows HPC Server 2008, you do not have to use MS MPI. You can use any MPI stack you choose. However, the security features that Microsoft has added to MS MPI might not be available in other MPI stacks.

MS MPI Features

As a full-featured implementation of the MPI2 specification, MS MPI includes more than 160 APIs that ISVs can use for inter-process communication and control, in addition to an application launcher that provides a high degree of control over the execution and parameters of each MPI application.

Programming Features of MS MPI

Although MS MPI includes more than 160 APIs, most programs can be written using only about a dozen of them. MS MPI includes bindings that support the C, Fortran 77, and Fortran 90 programming languages. Microsoft Visual Studio® Professional Edition and Visual Studio Team System include a remote debugger that works with MS MPI. Developers can start their MPI applications on multiple compute nodes from within the Visual Studio environment; then, Visual Studio automatically connects to the processes on each node, so the developer can pause and examine program variables on each node. Used with MS MPI, Windows HPC Server 2008 supports 32-bit and 64-bit applications.

MPI uses objects called *communicators* to control which collections of processes can communicate with each other. A communicator is a kind of “party line” of processes that can all receive messages from each other, but that ignore any messages within the communicator except messages directed to them. Each process has an ID or rank in the communicator. The default communicator includes all processes for the job and is known as MPI_COMM_WORLD. Programmers can create their own communicators to limit communications to a subset of MPI_COMM_WORLD.

MS MPI communication routines also include collective operations. Using collective operations allows the programmer to collect and evaluate all the results of a particular operation across all nodes in a communicator in one call.

MPI supports a high degree of control over communications and buffers across the nodes of the cluster. The standard routines are appropriate for many actions, but when you want specific buffering, MPI supports it. Communications routines can also be blocking or non-blocking, as appropriate.

MPI supports both predefined data types and derived data types. The built-in data type primitives are contiguous, but derived data types can be formed with contiguous or noncontiguous memory.

MPI2 supports the following four multithreading models:

- **{ MPI_THREAD_SINGLE}**. Only one thread will execute.
- **{ MPI_THREAD_FUNNELED}**. The process can be multithreaded, but only the main thread will make MPI calls (all MPI calls are "funneled'' to the main thread).
- **{ MPI_THREAD_SERIALIZED}**. The process can be multithreaded, and multiple threads can make MPI calls, but only one at a time; MPI calls are not made concurrently from two distinct threads (all MPI calls are "serialized'').
- **{ MPI_THREAD_MULTIPLE}**. Multiple threads can call MPI, with no restrictions.

All of these values are monotonic; that is, MPI_THREAD_SINGLE < MPI_THREAD_FUNNELED < MPI_THREAD_SERIALIZED < MPI_THREAD_MULTIPLE.

MS MPI supports up to and including MPI_THREAD_SERIALIZED; MPICH2 does not yet fully support MPI_THREAD_MULTIPLE.

Mpiexec Features

The `job submit` command is the usual mechanism for starting jobs in a Windows HPC Server 2008 cluster. This command is used to run mpiexec.exe, the application launcher for MS MPI, on a Windows HPC Server 2008 cluster and includes many parameters to control the application’s execution. You can specify the number of processors required, and the specific nodes that should be used for a given job. mpiexec, in turn, gives you full

control over how a job is executed on the cluster. A typical command line that runs an MPI application using the job scheduler might be:

```
job submit /numprocessors:8 /runtime:5:0 mpiexec myapp.exe
```

This command line submits the application MyApp.exe to the job scheduler, which assigns it to eight processors for a running time not to exceed five hours. Table 1 shows other common mpiexec arguments.

Table 1. Common mpiexec Arguments

Argument	Result
`mpiexec MyApp.exe` `mpiexec -n * MyApp.exe`	Runs a MyApp.exe process on each core assigned to the job or, when used with the HPC Server 2008 SDK, on each core of the local system.
`mpiexec -hosts 2 Node1 1 Node2 1 MyApp.exe`	Runs one MyApp.exe process on each of two specific compute nodes.
`mpiexec -cores 1 MyApp.exe`	Runs one MyApp.exe process on each of the nodes assigned to the job.
`mpiexec -n 4 MyApp.exe`	When used with `job submit`, runs MyApp.exe on four of the cores assigned to the job. When run using the HPC Server 2008 SDK, simulates a cluster on one computer by running four MyApp.exe processes on the local computer (where mpiexec was run).
`mpiexec -wdir \\headnode\MyFolder MyApp.exe`	Specifies a single, shared working directory for all the MyApp.exe processes in the job.
`mpiexec -wdir "%%USERPROFILE%%\My Folder" MyApp.exe`	Specifies a separate working directory (with spaces in the path which necessitate the double-quote marks) for each MyApp.exe process in the job. This directory will be the user's home directory on each node which is taken from the USERPROFILE environment variable on each node.
`mpiexec –n 1 master : -n * worker`	Runs one master process and worker processes on the remaining cores assigned to the job or, when used with `the HPC Server 2008 SDK`, on the remaining cores of the local host.
`mpiexec –gmachinefile nodes.txt –n 1 master : worker`	Runs one master process, and 31 worker processes where nodes.txt lists four hosts, each with eight cores per host.
`mpiexec -trace (pt2pt,coll,interconn) -n 2 app1`	Trace the point-to-point and collective MPI calls the two application processes are making, plus the underlying interconnect calls made by MS MPI.
`mpiexec -env MPICH_NETMASK 157.59.0.0/255.255.0.0 MyApp.exe`	Routes MyApp.exe MPI traffic to the 157.59.x.x/255.255.0.0 network in the cluster.

Argument	Result
`mpiexec –lines  MyApp.exe`	Runs MyApp.exe on each core assigned to the job and prefix all output with its source rank.

Mpiexec supports command-line arguments, environment variables, and command files for execution, providing maximum flexibility in job control to both the administrator and the user on a Windows HPC Server 2008 cluster. You can set environment variables for a specific job, or you can set them globally, across the entire cluster.

An important improvement that MS MPI brings to MPICH2 is in how security is managed during execution of MPI jobs. Each job is run *with the credentials of the user*. Credentials are present only while the job is being executed and they are erased when the job completes. Individual processes have access only to a logon token for their process, and do not have access to the job credentials or to the credentials used by other processes.

You can also use the job scheduler to reserve nodes for a job (including credentials), and then submit the jobs directly from Visual Studio 2005 using Mpiexec. (For specific guidance on doing this, see Debugging Parallel Applications Using Visual Studio 2005 on the Microsoft Web site.) This functionality greatly simplifies the debugging process when building an application. Alternatively, you can create a smaller debugging cluster to use with Mpiexec directly from Visual Studio.

Event Tracing Integration

Windows HPC Server 2008 adds an important new feature for developers looking to quickly debug and tune their MPI programs: MS MPI links with Event Tracing for Windows (ETW), part of Windows Server 2008. This linking enables developers to:

- **Create traces in production environments** – Without using special builds; just add a `-trace` flag to `mpiexec` to automatically generate trace files for later analysis.
- **Create time-correlated logs of MPI events** – From all processes on all nodes running an MPI application. The `mpisync` command enables high-precision syncing of events, simplifying identification of interactions.

MS-MPI tracing is dual-purpose tracing – it can be used for both performance analysis and application trouble-shooting. The synchronized, high-precision timing data allows the developer to do a precise analysis of where an application spends execution time, while the entry and exit of each MPI API call is logged along with the data that was passed into and out of the function, facilitating trouble-shooting.

The flow of an MPI application using Event Tracing for Windows is shown in Figure 1.

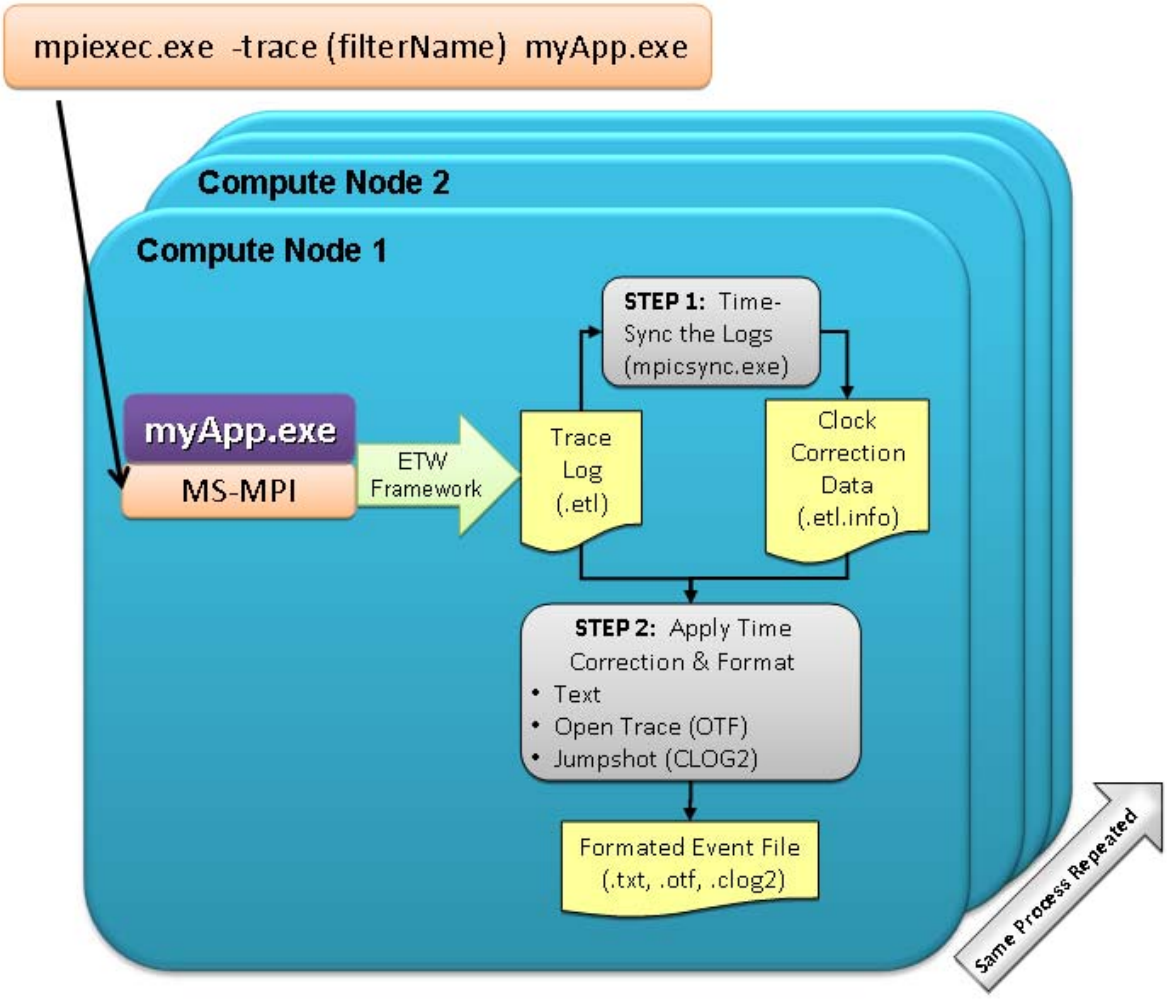

Figure 1. Event tracing flow

Implementing MPI

The original MPI specification was formulated and agreed to by the MPI Forum in the early 1990s. That same group of approximately 40 organizations extended the MPI specification in 1996 to create MPI2. Today, the MPI specifications are the de facto message-passing standards across most HPC platforms.

Programs written to MPI are portable across platforms and across various implementations of MPI without the need to rewrite source code. Although originally aimed at distributed systems, MPI implementations now also support shared-memory systems. In addition, MS MPI supports lower latency, shared-memory communications. The shared-memory communications implementation, which has been completely rewritten for Windows HPC Server 2008, improves the overall efficiency of communications between cores—especially when an application is using both shared memory and networked communication—and reduces the CPU cost associated with message passing between cores. When combined with the new NetworkDirect MS MPI interface and the new networking stack in Windows Server 2008, the result is a significantly more efficient HPC cluster.

What Protocols and Hardware Are Supported?

Windows HPC Server 2008 includes MS MPI as part of HPC Pack 2008. MS MPI uses the Microsoft NetworkDirect protocol for maximum compatibility with high-performance networking hardware and maximum networking performance. MS MPI can use any Ethernet connection that is supported by Windows Server 2008, as well as low-latency and high-bandwidth connections such as InfiniBand, 10-Gigabit Ethernet, or Myrinet. Windows HPC Server 2008 also supports the use of any network connection that has either a NetworkDirect or Winsock Direct provider. Gigabit Ethernet provides a high-speed and cost-effective interconnect fabric, while InfiniBand, 10-Gigabit Ethernet, and Myrinet are ideal for latency-sensitive and high-bandwidth applications.

The NetworkDirect protocol bypasses Windows Sockets (Winsock) and the TCP/IP stack, using Remote Direct Memory Access (RDMA) on supported hardware to improve performance and reduce CPU overhead. Figure 2 shows how MS MPI works with NetworkDirect drivers to bypass the Winsock and TCP/IP stacks.

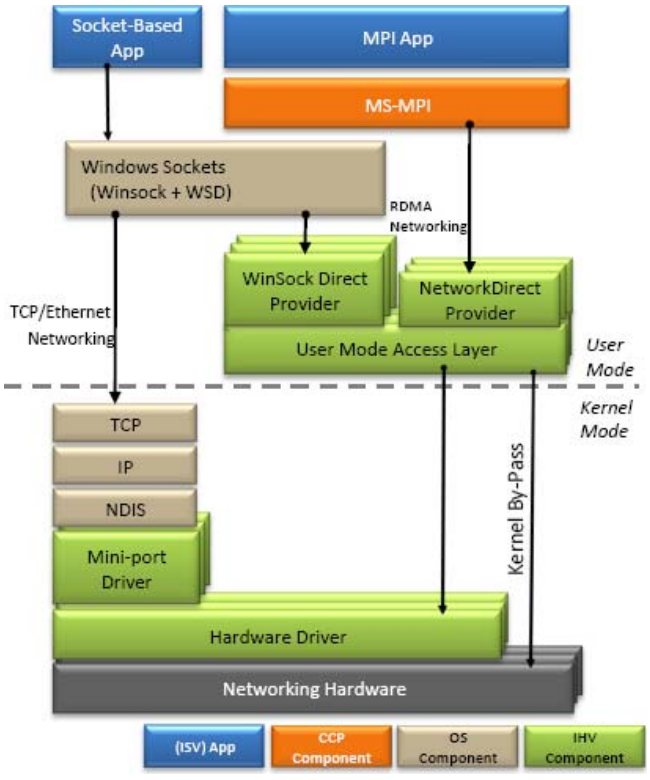

Figure 2. NetworkDirect architecture

Topology

Windows HPC Server 2008 supports a variety of network topologies, including those with a dedicated MPI interface and those that use a private network for cluster communications and MPI. Windows HPC Server 2008 even supports a topology that uses one network interface card (NIC) on each node that shares a public network for communications. Windows HPC Server 2008 supports the following five topologies:

- **Three NICs on each node**. One NIC is connected to the enterprise network; one NIC is connected to a private, cluster management network; and one is connected to a high-speed, dedicated application network.
- **Three NICs on the head node and two on each of the cluster nodes**. The head node has a connection to the enterprise network, and each compute node has a connection to the private network and a connection to a high-speed application network.
- **Two NICs on each node.** One NIC is connected to the enterprise network, and one is connected to the private, dedicated cluster network.
- **Two NICs on the head node and one on each of the compute nodes**. The head node has a connection to the enterprise network, and all nodes connect to a private network.
- **One NIC on each node with all network traffic sharing the enterprise network**. In this limited networking scenario, automated deployment of compute nodes using Windows Deployment Services is not supported, and each compute node must be manually installed and activated.

Windows HPC Server 2008 is built on Windows Server 2008 and is designed to integrate seamlessly with other Microsoft server products. For example, you can use Microsoft System Center Operations Manager to monitor Windows HPC Server 2008, or applications can integrate with Microsoft Exchange Server to mail a job status report to the job owner.

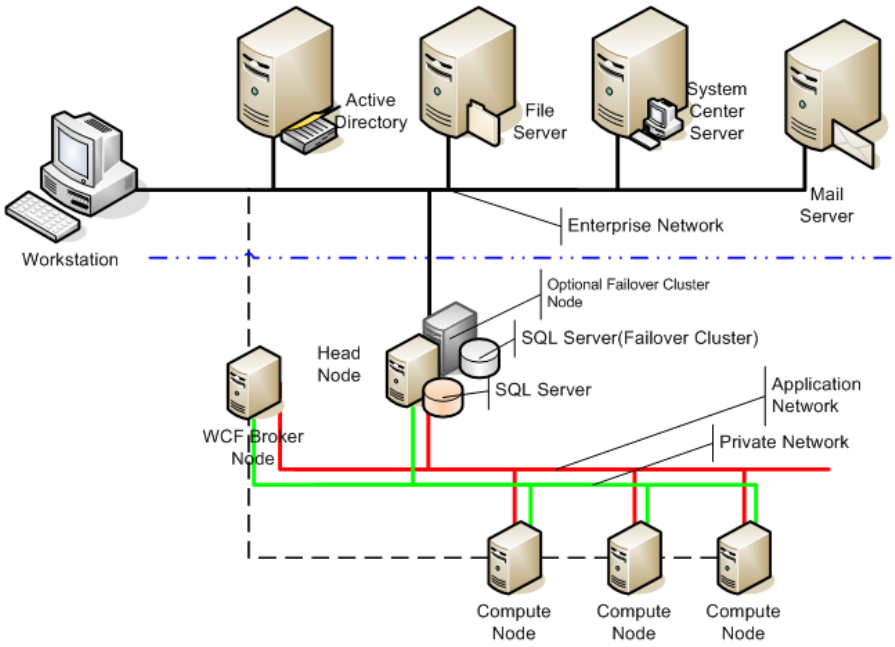

Figure 3. Typical Windows HPC Server 2008 network topology

IMPORTANT: In a debugging environment, the developer's Visual Studio workstation **must** have direct access to the compute nodes to be able to perform remote debugging.

Security and the MS MPI Implementation

MS MPI and the MPICH2 reference implementation differ most significantly in terms of how they manage security. MS MPI integrates with Microsoft Active Directory® directory service to simplify running with user credentials instead of using the root account to run jobs.

Integration with Active Directory

Windows HPC Server 2008 is integrated with and dependent on Active Directory to provide security credentials for users and jobs on a cluster. Before you can install HPC Pack 2008 on the head node and create the cluster, you must either join the head node to an Active Directory domain or promote it to be a domain controller so it will create its own domain.

Active Directory accounts are used for all job creation and execution on the cluster. All jobs are executed using the credentials of the user submitting the job.

Credential and Process Management

When an MPI job is submitted, the credentials of the user submitting the job are used for that job. At no time are passwords or credentials passed in clear text across the network. Credentials are passed using only authenticated and encrypted channels, as shown in Figure 4. Credentials are stored with the job data on the head node and deleted when the job completes. At the user's discretion, the credentials can also be cached on the individual client computer to streamline job submission, but in this option, the credentials are encrypted with a key known only to the head node. While a job is being computed, the credentials are used to create a logon token and then erased. Only the token is available to the processes being run, further isolating credentials from other processes on the compute nodes.

Processes running on compute nodes are run as one Windows job object, enabling the head node to keep track of job objects and clean up any processes when the job completes or is cancelled.

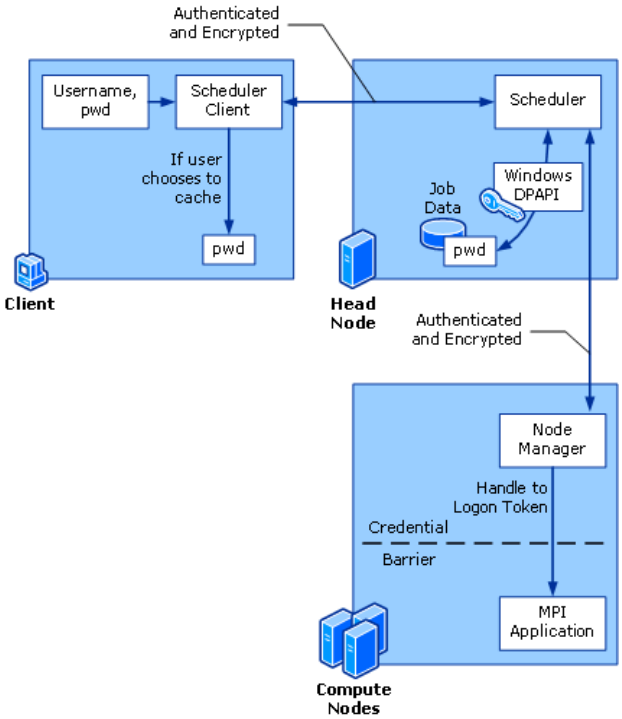

Figure 4. Credential management of MPI jobs

Summary

MPI and MPI2 are widely accepted specifications for managing messaging in high performance clusters. Among the most widely accepted implementations of MPI is the Argonne National Laboratory MPICH2 reference implementation, which is open source. Windows HPC Server 2008 includes MS MPI, which is based on and is highly compatible with MPICH2. At the API level, MS MPI is identical to the more than 160 APIs implemented by MPICH2. At the same time, MS MPI adds enhanced security and process management capabilities for enterprise environments and a new execution tracing feature for Windows HPC Server 2008. MS MPI uses an efficient shared-memory communication between cores on a compute node and NetworkDirect drivers to provide high performance MPI network support for Gigabit Ethernet and InfiniBand adapters, and supports adapters that have a NetworkDirect or Winsock Direct provider.

Appendix

A Sample MPI Program

A frequently used example of an MPI program is the calculation of *pi*, which can be expressed as follows:

```
#include <time.h>
#include <mpi.h>

void main(int argc, char *argv[])
{
        Int     NumIntervals      = 0;    //num intervals in the domain [0,1] of F(x)= 4 / (1 + x*x)
        double  IntervalWidth     = 0.0;  //width of intervals
        double IntervalLength     = 0.0;  //length of intervals
        double  IntrvlMidPoint    = 0.0;  //x mid point of interval
        int     Interval          = 0;    //loop counter
        int     done              = 0;    //flag
        double  MyPI              = 0.0;  //storage for PI approximation results
        double  ReferencePI       = 3.14159265358979323846264643; //ref value of PI for comparison
        double PI;
        char    processor_name[MPI_MAX_PROCESSOR_NAME];
        char    (*all_proc_names)[MPI_MAX_PROCESSOR_NAME];
        int     numprocs;
        int     MyID;
        int     namelen;
        int     proc = 0;
        char    current_time [9];

        if (argc > 1)
        {
                MPI_Init(&argc,&argv);
                MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
                MPI_Comm_rank(MPI_COMM_WORLD,&MyID);
                MPI_Get_processor_name(processor_name,&namelen);

                if (MyID == 0){
                                /* if this is the Rank 0 node... */
                                NumIntervals = (int)_strtoi64(argv[1], 0, 10); /* read number of
                                                intervals from the command line argument */
                                all_proc_names = malloc(numprocs * MPI_MAX_PROCESSOR_NAME);   /* allocate
                                                memory for all proc names */
                }

                /* send number of intervals to all procs */
                MPI_Bcast(&NumIntervals, 1, MPI_INT, 0, MPI_COMM_WORLD);

                /* get all processor names for this job */
                MPI_Gather(processor_name, MPI_MAX_PROCESSOR_NAME, MPI_CHAR, all_proc_names,
                        MPI_MAX_PROCESSOR_NAME, MPI_CHAR, 0, MPI_COMM_WORLD);

                if (NumIntervals == 0)
                {
                                printf("command line arg must be greater than 0");
                }
                else
                {
                                //approximate the value of PI
```

```
IntervalWidth   = 1.0 / (double) NumIntervals;

for (Interval = MyID + 1; Interval <= NumIntervals; Interval +=
        numprocs){
        IntrvlMidPoint = IntervalWidth * ((double)Interval - 0.5);
        IntervalLength += (4.0 / (1.0 + IntrvlMidPoint*IntrvlMidPoint));
}

MyPI = IntervalWidth * IntervalLength;
MPI_Reduce(&MyPI, &PI, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Related Links

MPICH2 Home Page
http://go.microsoft.com/fwlink/?LinkId=55115

MPI tutorial at the Lawrence Livermore National Lab
http://go.microsoft.com/fwlink/?LinkId=56096

Deploying and Managing Microsoft Windows HPC Server 2008
http://go.microsoft.com/fwlink/?LinkId=55927

Using Windows HPC Server 2008 Job Scheduler
http://go.microsoft.com/fwlink/?LinkId=55929

Migrating Parallel Applications
http://go.microsoft.com/fwlink/?LinkId=55931

Debugging Parallel Applications Using Visual Studio 2005
http://go.microsoft.com/fwlink/?LinkId=55932

HPC Community Forums (All)
http://go.microsoft.com/fwlink/?LinkId=118932

HPC Community Forums (Developers)
http://go.microsoft.com/fwlink/?LinkId=118933

HPC Community Forums (MPI)
http://go.microsoft.com/fwlink/?LinkId=118937

HPC Community Forums (IT Professionals)
http://go.microsoft.com/fwlink/?LinkId=118935

HPC Community Forums (Job Submission, Scheduler)
http://go.microsoft.com/fwlink/?LinkId=118936