

MPI on NT: A Preliminary Evaluation of the Available Environments

Mark Baker¹ and Geoffrey Fox²

¹Dept. of Computer Science and Mathematics
University of Portsmouth, UK
mab@sis.port.ac.uk

²Northeast Parallel Application Center
Syracuse University, NY, USA
gcf@npac.syr.edu

Abstract

The aim of this paper is to discuss the functionality and performance of the current generation of MPI environments available for NT. The three environments investigated are WinMPICH^[1] from the Engineering Research Center at Mississippi State University, WMPI^[2] from the Instituto Superior de Engenharia de Coimbra, Portugal and FM-MPI^[3] from the Dept. of Computer Science at the University of Illinois at Urbana-Champaign.

In the first part of the paper we discuss briefly the advantages of using clusters of workstations and then move on to describe NT and the MPI environments being investigated. In the second part of the paper we describe, and then report on, our experiences of assessing the functionality of the MPI environments. In the third part of the paper we make a preliminary evaluation of the performance characteristics of the environments assessed using a number of simple, descriptive benchmarks. Finally, we summarise our findings and suggest a number of improvements that could be made to the environments assessed.

1. Introduction

The use of workstation clusters to prototype, debug and run parallel applications is becoming an increasingly popular alternative to using specialised, typically expensive, parallel computing platforms such as the Cray T3E. An important factor that has made the usage of workstations a practical proposition is the standardisation of many of the tools and utilities used by parallel applications. Examples of these standards are the message passing library MPI^[4] and data-parallel language HPF^[5]. Standardisation in this context enables applications to be developed, tested and even run on workstation clusters and then at a later stage be ported, with little modification, onto dedicated parallel platforms where CPU-time is accounted and charged for.

The reasons why workstation clusters are preferred over specialised parallel computers are well documented^{[6][7][8][9]}. Clearly the workstation environment is

better suited to applications that are not communication intensive as the existing LAN architectures are not capable of providing higher performance.

The rapid convergence in processor performance and kernel-level functionality of UNIX workstations and PC-based machines in the last three years, can be associated with the introduction of high-performance Pentium-based machines and the Window NT operating system. These factor, coupled with the comparatively low cost of PCs and their widespread availability in both academia and industry has helped initiate a number of software projects whose primary aim is to harness these resources in some collaborative way.

The two most popular high-level message-passing systems for scientific and engineering application are the PVM - Parallel Virtual Machine - from Oak Ridge National Laboratory^[10] and MPI - Message Passing Interface^[4]. PVM is both an environment and a message passing library which can be used to run parallel applications on systems ranging from high-end supercomputers through to clusters of workstations. The MPI standard^[11] standard only defines a message passing library and leaves, amongst other things, the initialisation and control of processes to individual developers to define. Like PVM, MPI is available on a wide range of platforms. Generally application developers choose MPI as it is fast becoming the *de facto* standard for message passing. MPI and PVM libraries are available for FORTRAN 90, ANSI C as well as other languages, including Java^[12].

Microsoft Corp.^[13] is the dominant provider of software in the personal computing market place. Microsoft provides two basic operating systems: Windows 95 (soon to become Windows 98) and Windows NT 4 (soon to become Windows NT 5)^[14]. NT and Windows 95 had approximately 66% of desktop operating systems market share in 1996^[15] - IBM OS/2, UNIX, Mac OS and DOS comprise the remainder of the market share.

2. NT Overview

Windows NT is a 32-bit Operating System^{[16][17][18][19]}. It is a pre-emptive, multi-tasking, multi-user operating system from Microsoft Corp. NT supports multiple CPU's and provides multi-tasking, using symmetrical multiprocessing. NT is a fault tolerant - each 32bit application operates in its own virtual memory address space. Unlike earlier version of Windows, NT is a complete operating system, and not an addition to DOS. NT has object-based security model and its own special file system (NTFS) that allows permissions to be set on a file and directory basis.

NT has several built-in networking protocols, such as - IPX/SPX, TCP/IP, and NetBEUI) and APIs, such as NetBIOS, DCE RPC, and Windows Sockets (WinSock). TCP/IP applications use WinSock to communicate over a TCP/IP network.

WinSock^[20] is based largely on the Berkeley UNIX sockets model, which uses sockets as the basic building blocks for application-to-application communication. The latest version of WinSock has a number of enhancements, including: protocol independence, support for multiple name and service resolution mechanisms, performance enhancements, and support for a variety of additional services.

2.1 C and FORTRAN Environments

Two C environments can be used with the MPI for NT environments: Microsoft Development Studio (MSDS) Visual C++^[21] and the Borland C++ Development Suite^[22].

Digital Visual FORTRAN (DVF)^[23] is based on Digital FORTRAN 90 compiler for Digital UNIX and OpenVMS Alpha. DVF is fully compatible with MSDS and is compliant with ANSI F77 and F90.

2.2 MPI Introduction

The Message Passing Interface (MPI)^{[4][11]} is a portable message-passing standard that facilitates the development of parallel applications and libraries. MPI is available in both C and in FORTRAN 77. The goals of the MPI design were portability, efficiency and functionality. Commercial and public-domain implementations of MPI exist, these run on a range of systems from tightly-coupled, massively-parallel machines (MPPs), through to Networks Of Workstations (NOWs).

MPI has a range of features including: point-to-point, with synchronous and asynchronous communication modes; and collective communication (barrier, broadcast, reduce). MPI supports features for grouping communicating processes and isolating unrelated communication contexts. Groups are used to define processes involved in collective communication. MPI rapidly becoming the *de facto* standard and there are a number free implementations available.

2.3 MPICH

MPICH^{[24][25]}, developed by Argonne National Laboratory and Mississippi State University (MSU), is probably the most popular of the current free implementations of MPI. MPICH is a version of MPI built on top of Chameleon^[26]. The portability of MPICH derives from being built on top of a restricted number of hardware-independent low-level functions, collectively forming an Abstract Device Interface (ADI). The ADI contains approximately 25 functions and the rest of MPI approximately 125 functions.

2.4 WinMPICH

WinMPICH^{[1][27]} from the Engineering Research Center at MSU is a port of MPICH for Microsoft Windows NT. WinMPICH allows processes to communicate with each other via either shared memory or over a network. WinMPICH was originally written to explore threads in the device layer for communication. TCP/IP support, which was added later, was added by means of a proxy process that runs when a process requires TCP/IP services. In this model when TCP/IP is used messages get sent via shared memory to the proxy processes on the sending and receiving machines before being copied to the destination process.

The WinMPICH release consists of source and binaries for a set of libraries and servers configured to be compiled and linked using Microsoft Visual C++ 4.51. WinMPICH is under development and is freely available. WinMPICH provides a homogeneous environment which is only capable of inter-operating with Win32 platforms.

2.5 WMPI

WMPI^[2] from the Instituto Superior de Engenharia de Coimbra, Portugal is a full implementation of MPI for Microsoft Win32 platforms. WMPI is based on MPICH and includes a p4^{[28][29]} device standard. P4 provides the communication internals and a startup mechanism. The WMPI package is a set of libraries (for Borland C++, Microsoft Visual C++ and Microsoft Visual FORTRAN). The release of WMPI provides libraries, header files, examples and daemons for remote start-up.

WMPI can co-exist and interact with MPICH/ch_p4 in a cluster of mixed Windows 95 and NT, as well as UNIX. WMPI is still under development and is freely available.

2.6 Illinois Fast Messages (FM)

FM-MP^{[3][30]} is from the Dept. of Computer Science at the University of Illinois at Urbana-Champaign. FM-MPI is a version of MPICH built on top of Fast Messages. The FM interface is based on Berkeley Active Messages^[31]. FM, unlike other messaging layers, is not the surface API, but the underlying semantics. FM contains functions for sending long and short messages and for extracting messages from the network.

FM has a low-level software interface that delivers hardware communication performance; however, higher-level layers interface offer greater functionality, application portability and ease of use. A problem with this is that high level interface abstractions add overhead to communication and generally degrade

overall performance significantly. For this reason a number of high-level APIs have been developed on top of FM: these include MPI, SHMEM and Global Arrays.

To run MPI on FM, the MPICH's ADI was adapted to communicate with FM calls. FM-MPI was first developed in October 1995 and was designed to run via Myrinet-connected systems. Recently, a variant of FM-MPI that runs on top of WinSock 2 was released as part of the High-Performance Virtual Machines (HPVM) project^{[32][33]} being undertaken by the Concurrent System Architecture Group (CSAG).

3. Functionality Tests

3.1 Test Environment

The test environment consisted of cluster of NT machines (Server/Workstation) with 32/64 Mbytes of memory and either a FAT or NTFS filesystem.

3.1.1 Comments On The Set Up Of The MPI Environments

- *Daemons* - if these are set up under a system account the MPI job will inherit administrators' privileges.
- The services had to be individually set up on each workstation - currently there are no native tools for setting up remote services.
- As WinMPICH provides source code the libraries and servers can be debugged and recompiled locally - source code is not provided with WMPI and HPVM.
- In WinMPICH and WMPI each MPI program had its own configuration file. Under HPVM this was not the case. To run a MPI program it was necessary to start it running on every machine by entering its name, the number of processes that were to be used, and a common key string value for all the processes that would be co-operating together.

4. Test Suites

There are several suites available to test the functionality of MPI^[34] ports. As the MPI environment studied in this project is based on MPICH and a suite designed to test this implementation was chosen.

4.1 C Test Suite

The C test suite used was the one developed by IBM^[35] that had modified to comply fully to the MPI standard and be compatible with the MPICH. The suite

consists of eighty-seven C programs that test the following MPI calls and data types; collective operations, communicators, data types, environmental inquiries, groups, point to point and virtual topologies.

4.2 Functionality Test Results

4.2.1 Compilation, Linking and Test Runs

All the codes from the test suite were successfully compiled and linked for each of the three environments. The codes were run in three basic configurations:

- *Shared memory* - up to eight processes were run on one processor.
- *Distributed memory* - each process ran on a separate processor - up to six processors were used.
- *Mixed* - both shared and distributed memory modes were used together.

WinMPICH - All the codes ran to completion in shared memory mode. In distributed memory mode, apart from one code, each program ran to completion successfully. The code, which used non-blocking sends and receives (`ISend/IREvc`), failed to run.

WMPI - All the codes ran to completion in both shared memory and distributed memory modes. The main problem initially encountered with WMPI was memory allocation. By default the amount of global memory used by WMPI is set to 1 MByte. It was necessary to change this from 10 - 15 MBytes to ensure all the test codes ran successfully to completion.

HPVM - All the test codes started to run, but several failed during run-time - it seemed that most of the errors reported were associated with memory problems. In all about ten of the codes from the test suite failed.

4.3 Typical Problems Encountered - Summary

The majority of errors encountered were memory-related problems. Test programs generally failed by hanging indefinitely, reporting no error or by displaying Assert messages on the console - reporting an application error which generally indicated that the program had referenced memory that it could not read.

To run a program under HPVM the user needs to log into each machine being used, start a DOS window, run the Context Manager, then open another DOS window and start the application up. This was tedious, especially when using more than one workstation.

Currently WinMPICH programs are compiled as console applications, so when an application runs a DOS window pops-up for each process in the MPI job. In addition, when running in distributed memory mode another DOS window will pop-up - this is produced by the socket-server process running on each workstation (`mpitcserver`). The appearance DOS windows is annoying when running jobs in shared memory, but can cause problems when using distributed memory. It is probable that a window popping-up on a remote workstation may be killed by its local user - if this happens the application will hang and no warning will be relayed back to the original jobs owner. According to the developers this problem will cease when they implement piping of standard I/O from remote workstations to designated destinations (file or console).

A further problem encountered with WinMPIch was that of application initialisation. Often, when an application was started from the command line, console windows popped up on the appropriate workstations and then hung; on other occasions the application would run successfully. To kill the application it was necessary to interrupt the command-line by typing CTRL-C and then killing all the DOS windows, including all those on remote machines. Experience showed that an application would not initialise itself and run to completion approximately once in three attempts.

5. Performance Tests

5.1 Introduction

There are several suites of distributed benchmark codes including EuroBen^[36], NAS Parallel Benchmarks^[37] and Parkbench^[38]. These popular and well known standard benchmarks are written in FORTRAN and so could not be used. Alternative C variants were used for our preliminary performance purposes. The aim of these initial tests is restricted to gathering data that will help indicate the expected communications performance (peak bandwidth and message latency) of MPI on NT. The configuration of two systems used to demonstration the typical performance of NT systems are shown in Table 1.

	NT Server 4	NT Workstation 4
Processor	150 MHz Pentium Pro	75 MHz Pentium Pro
L2 Cache	256 KB	256 KB
Memory	48 Mbytes	24 Mbytes
Disk	2.1 Gbytes	0.81 Gbytes
Network	Ethernet (10bT)	Ethernet (10bT)

Table 1 - NT System Configuration.

5.2 Multi-processor Benchmark - PingPong

In this program increasing sized messages are sent back and forth between processes - this is commonly called PingPong. This benchmark is based on standard blocking MPI_Send/MPI_Recv. PingPong provides information about latency of MPI_Send/MPI_Recv and uni-directional bandwidth. To ensure that anomalies in message timings do not occur the PingPong is repeated 1000 times for messages sizes 64K and smaller.

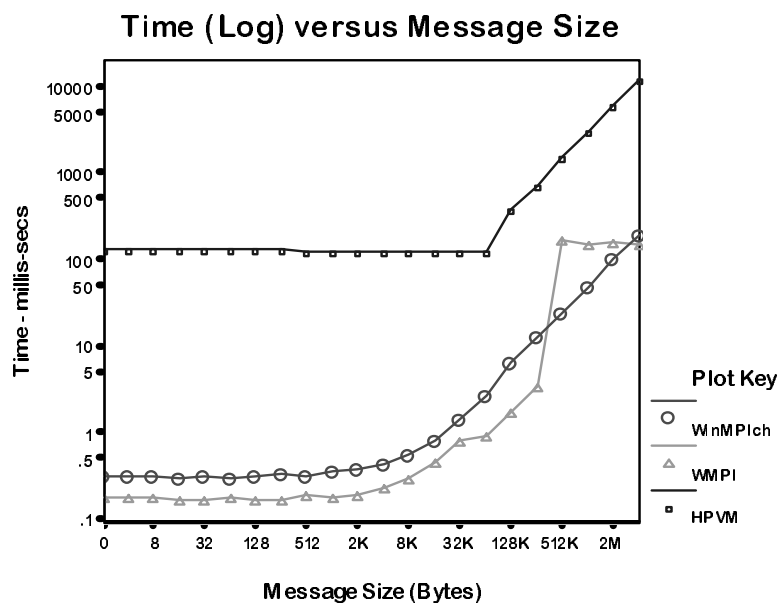


Figure 1 - Message Time versus Size (Shared Memory)

5.2.1 Shared Memory Performance - Latencies

The message start-up latencies for WMPI and WinMPIch are 175 and 300 μ sec respectively. Both show a similar small increase in message communications time up until 4K - at this point the communications times increase almost exponentially. At 128K there is a large step in WMPI communications, thereafter time to send 512K- \rightarrow 4M messages decreases. This step in performance was reported to the WMPI developers. The reason for the step was given as being the boundary where the largest packet size (128K) started to be used in WMPI.

The performance of HPVM in shared memory is poor throughout. Communications times between 0- \rightarrow 64K are constant at approximately 121msec. Thereafter communications time increases almost exponentially. The

reasons for the very poor performance is explained by the developers as being due to the fact that their implementation of MPI-WinSock was designed to be used in a distributed environment and is not optimised for processes running on one process and consequently incurs the large performance hit.

5.2.2 Shared Memory Performance - Bandwidth

The message peak bandwidth for WMPI and WinMPIch are 80 and 25 Mbytes/s respectively. Bandwidth approximately doubles at each discrete new message size up until 4K: thereafter the bandwidth decreases. WinMPIch peaks at 64K and WMPI at 256K. The performance of WinMPIch remains approximately constant. After 256K, WMPI exhibits a sharp decrease and bandwidth falls to about 3 Mbytes/s, thereafter the performance rapidly rises.

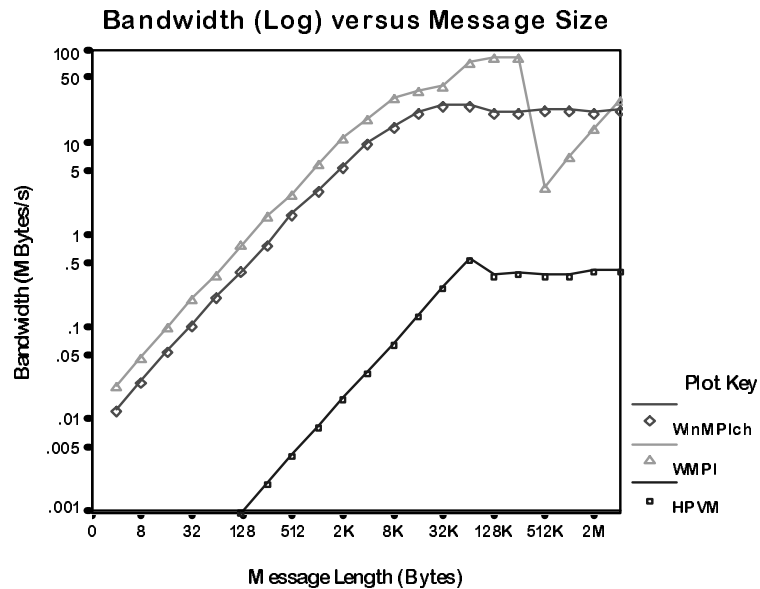


Figure 2 - Communications Bandwidth versus Message Size (Shared Memory)

The performance of HPVM in shared memory is poor throughout. The bandwidth peaks at 64K and remains approximately constant thereafter. Communications times between 0 bytes and 64K bytes are constant at approximately 121 msec.

5.2.3 Distributed Memory Performance - Latencies

The message start-up latencies for WMPI, HPVM and WinMPIch are 3.4, 5.7 and 9.9 msec respectively. For HPVM and WMP messages the time decreases slightly thereafter. HPVM is approximately 2 - 3 msec slower than WMPI up until 4K, thereafter the times for HPVM and WMPI are approximately equivalent.

The performance of WinMPIch in distributed memory is poor throughout. The start-up latency of almost 10 msec is at least twice that which would be expected and can be explained by the extra memory copies that occur between MPI processes and the `mpitcproc` on each machine.

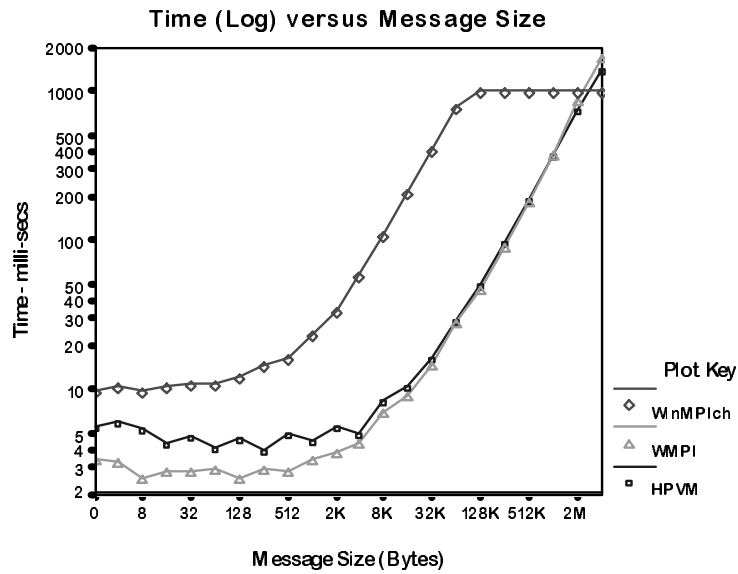


Figure 3 - Communications Time versus Message Size (Distributed Memory)

5.2.4 Distributed Memory Performance - Bandwidth

The message peak bandwidth for WMPI, HPVM and WinMPIch are 2.8, 3.0 and 0.9 Mbytes/s respectively. The performance curves of WMPI and HPVM are approximately equivalent.

The performance of WinMPIch in distributed memory is poor throughout - a maximum 80K bytes/s bandwidth is low compared with the almost 3 Mbytes/s peak achieved by WinMPIch/WMPI. This behaviour is believed to be caused by additional memory copies that occur between MPI processes and the `mpitcproc` on each machine.

It should be noted that system used for the distributed memory PingPong is two NT systems connected by PCMCIA cards and a dedicated Ethernet cable. It is believed that the achieved peak bandwidth of nearly 3 Mbytes/s compared with the normal 1.25 Mbyte/s is due fact that the PCMCIA cards are capable of driving the Ethernet at this higher speed.

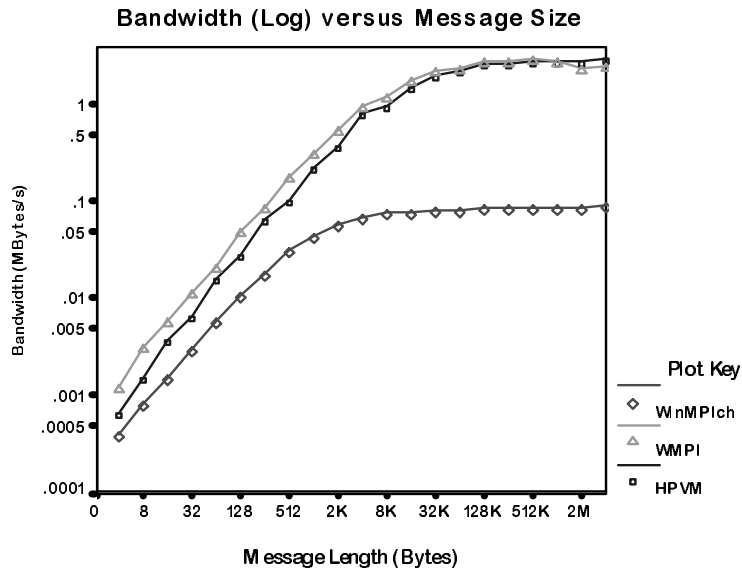


Figure 4 - Communications Bandwidth versus Message Size (DM)

6. Conclusions

6.1 Introduction

Within this paper we have discussed our experiences testing the usability, set up, functionality and performance of the three main MPI environments for NT. It should be noted that all three environments are either still in beta or at a very early stage of release. Originally, it was hoped that the authors would be able to investigate the FORTRAN interfaces to MPI. However, these interfaces are either not available or not configured for the current release of FORTRAN compilers available. The following sections summarise our findings so far.

6.2 Usability and Functionality

- *Administration* - The set up and configuration of WinMPIch and WMPI is similar to that of MPICH under a UNIX environment. The main difference is that remote administration of workstations under NT is more difficult. Either third-party tools need to be purchased or each machine needs to be administrated by logging on locally. Under HPVM the Global Resotrce Manager (GRM) and the CM need to be run each time a parallel job is started - due to the fact that neither the GRM or CM has been released as a NT service. Consequently, unlike WinMPIch and WMPI where the services are set up just once, HPVM requires the GRM and CM to be set up again and again.

- *Running applications* - WinMPIch and WMPI provide a command-line interface with which a user of MPICH on a UNIX workstation would be familiar - `mpirun` or `p4` switches. In addition WMPI provides a “friendly” VC++ dialog-box pop-up. The usability of HPVM is more problematic. There are two means of starting MPI jobs, via a Java Applet (which was not tested) or via a DOS window and its command-line - here the user cannot use a MPI configuration file but rather has to enter command-line options on each of the machines used for the MPI job.
- *Compilation and linking* - All three environments provide fully functional C interfaces to MPI. At this stage the functionality of the FORTRAN interface to MPI has not been tested due to the reasons mentioned earlier in this paper .
- *Heterogeneity* - WMPI provides support for a heterogeneous environment (Win32 and UNIX), WinMPIch provides a homogeneous environment (Win32). HPVM can support a heterogeneous environment - this ability is dependent on the machines being used having the same byte-ordering, but according to the developers it has not been rigorously tested.

6.3 Performance

6.3.1 Shared Memory

Both WinMPIch and WMPI provide fully functional, fast and reliable MPI environments in shared memory. WMPI is faster than WinMPIch by approximately a factor of two up to 32K and then by a factor of four up to 256K. WMPI does, however, exhibit a very large performance degradation when messages sizes go beyond 256K. It suddenly plummets to 3 Mbytes/s at 512K before recovering to near 30 Mbytes/s at 4M. In comparison WinMPIch peaks at approximately 25 Mbytes/s and is thereafter relatively steady - remaining at around the 22 Mbytes/s.

The performance of HPVM is very poor throughout, In fact message communications times decrease steadily from 124 msec (0B) to 121 msec (64K). Thereafter times approximately double at each new discrete message size. Even though the communications bandwidth steadily increases as message size increases, a peak bandwidth of 0.4 Mbytes/s is poor.

6.3.2 Distributed Memory

The performance of WMPI and HPVM is fairly similar throughout the range of message sizes used. The start-up latency of HPVM is approximately 60% greater than WMPI. This difference in times to communicate a message is nearly constant until 4K, thereafter the times and bandwidths are almost the same. Both WMPI and HPVM reach the peak bandwidth achievable on the test system of 3 Mbytes/s. The start-up latency of between 3.5 and 6 msec is similar to that which would be expected between UNIX workstations.

The performance of WinMPIch is relatively poor compared to WMPI and HPVM. The start-up latency of 10 msec is double that which would normally be expected. In addition, even though the communications bandwidth increases steadily, a peak of around 100 Kbytes/s is 300% down on the actual peak achievable.

6.4 Overall Conclusions

In terms of functionality and performance WMPI is clearly the best environment of the three investigated for running MPI jobs. However, WMPI falls down, in the authors' opinion, if user support, documentation and the availability of source code is also taken into consideration.

WinMPIch functionality is good and its performance almost matches that of WMPI in shared memory. However, in distributed memory, WinMPIch has problems with reliability, functionality and performance. The problems encountered with non-blocking send/receive (`Isend/Irecv`) is an area of concern, but may disappear when WinMPIch is upgraded to the latest version of MPICH. There is an obvious structural problem with WinMPIch - the existence of the `mpitcproc` process highlights this fact. The current structure has a large performance impact (many additional memory copies are needed) and is probably the cause of the start-up problem. WinMPIch is fairly well documented and user support is good and responsive.

The HPVM environment has been designed around using Myrinet networks and Fast-Message protocols. The fact that there are several high-level interfaces (MPI, SHMEM and Global-Arrays) and a WinSock interface makes HPVM a potentially very desirable environment. In this project we have been assessing the WinSock interface to HPVM. This is not a target interface originally chosen by the developers and consequently can only be considered an early prototype. In shared memory HPVM performance is very poor and virtually unusable. In distributed memory, the performance of HPVM is very close to WMPI, and with further development could match or out-perform it. Approximately 10% of the test suite codes failed when run in distributed memory under HPVM - most of these failures were thought to be due to a lack of memory. Currently, memory buffer size cannot be increased in HPVM so this could be a problem for some applications. HPVM documentation is fairly comprehensive but is directed toward the Myrinet version of the environment. User support is fairly responsive, but not always helpful.

Finally, it is clear that an MPI implementation should be inter-operable over a heterogeneous computing environment. For example an MPI job should be capable of running and co-exist on NT, Linux, UNIX and any other platform where MPI has been ported. Only WMPI has such a capability. HPVM requires that the CM be ported to each environment and then is only capable of providing

heterogeneous support if the byte-ordering is the same on all machine. WinMPIch currently only on Windows platforms.

6.5 Suggested Improvement

- *Application Initialisation* - A simple Java Applet would provide a good alternative graphical interface for starting MPI jobs running.
- *Application Configuration* - Both WinMPIch and WMPI allow applications to be run via configuration files. The configuration language used is very basic and would be improved greatly if it included macros and recursion.
- *Job configuration files* - This would be a useful feature for HPVM.
- *Console Application* - WinMPIch needs to be redeveloped as Windows applications - popping-up console windows is in convenient and unnecessary.
- *I/O* - WinMPIch I/O needs to be redirected - default to single process or file.
- *Security implications* - The privileges of MPI Services needs to be addressed.
- *NT Services* - HPVM deamons need to be urgently developed so that they can be configured as NT services.
- *Heterogeneous Support* - MPI should be capable of providing heterogeneous computing support.

6.6 Future Work

The authors intend to continue investigating the MPI for NT and will be testing the functionality and performance of the FORTRAN interfaces next. Thereafter, a broader study will look at the performance of MPI on NT against other MPI environments - such as UNIX clusters. In the longer term a study of alternative technologies, such as Java and Corba, will be undertaken. It should be noted that an unabridged version of this paper is available at ^[39].

Acknowledgments

The authors wish to thank the developers of WinMPIch, WMPI and HPVM for their help with this project. The authors would also like to thank Rose Rayner for her time and patience proof reading this paper.

References

^[1] *WinMPIch* - <http://www.erc.msstate.edu/mpl/mpiNT.html>

^[2] *WMPI* - <http://alentejo.dei.uc.pt/w32mpi/>

^[3] *FM-MPI* - <http://www-csag.cs.uiuc.edu/projects/comm/mpl-fm.html>

^[4] Snir, Otto, Huss-Lederman, D. Walker, and J. Dongarra, *MPI The Complete Reference*, MIT Press; 1996

^[5] C. Koelbel, et. al., *The High Performance Fortran Handbook*, The MIT Press, 1994.

^[6] M. Baker, et. al., *Review of Cluster Management Software*, NHSE Review, May 1996, - <http://nhse.cs.rice.edu/NHSEreview/CMS/>

-
- ^[7] L. Turcotte, *A Survey of Software Environments for Exploiting Networked Computing Resources*, Engineering Research Center for Computational Field Simulation, Mississippi State, 1993
- ^[8] G.F. Pfister, *In Search of Clusters*, Prentice Hall PTR, 1995, ISBN 0-13-437625-0
- ^[9] T. Anderson, D. Culler, and D. Patterson. *A Case for NOW (Network of Workstations)*. IEEE Micro, 15(1): 54-64, February 1995.
- ^[10] A. Geist, et. al, *PVM - Parallel Virtual Machine - a User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1995
- ^[11] MPI Forum, *MPI: A Message-Passing Interface Standard*, May 5, 1994, University of Tennessee, Knoxville, Report No. CS-94-230
- ^[12] *MPI Java Wrapper* - <http://www.npac.syr.edu/users/yjchang/javamPI/>, November 1997.
- ^[13] *Microsoft Corporation* - <http://www.microsoft.com>
- ^[14] A. Watts, *High Command*, PC Direct, December 97
- ^[15] *The Microsoft Market Share* - <http://newspost.sfsu.edu/ms/markets.html>
- ^[16] H. Custer, *Inside Windows NT*, Microsoft Press, 1993, ISBN 1-55615-481-X
- ^[17] K. Spenser, *NT Server: Management and Control*, Prentice Hall PTR, 1996
- ^[18] E. Pearce, *Windows NT In a Nutshell*, O'Reilly and Associates, Inc, 1997
- ^[19] *Windows NT Server* - <http://www.microsoft.com/ntserver/>
- ^[20] *WinSock Resources* - <http://www.stardust.com/wsresource/wsreserve.html>.
- ^[21] *Visual Studio and Visual C++* - <http://www.microsoft.com/visualtools/>
- ^[22] *Borland C++* - <http://www.borland.com/bcppbuilder/>
- ^[23] *Digital Visual FORTRAN* - <http://www.digital.com/fortran/>
- ^[24] *MPICH* - <http://www.mcs.anl.gov/mpi/mpich/>
- ^[25] W. Gropp, et. al., *A high-performance, portable implementation of the MPI message passing interface standard* - <http://www.mcs.anl.gov/mpi/mpicharticle/paper.html>
- ^[26] W. Gropp and B. Smith, *Chameleon parallel programming tools users manual*. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.
- ^[27] *MPI Implementation for NT*, B. Protopopov, Mississippi State University, 22nd May 1996 - <http://www.erc.msstate.edu/mpi/NTfiles/winMPICHpresent.ps>
- ^[28] R. Buttler and E. Lusk, *User's Guide to the p4 Parallel Programming System*. ANL-92/17, Argonne National Laboratory, October 1992.
- ^[29] R. Butler and E. Lusk, *Monitors, messages, and clusters: The p4 parallel programming system*. Parallel Computing, 20:547--564, April 1994.
- ^[30] S. Parkin, V. Karamcheti and A. Chein, *Fast-Message (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors*, IEEE Microprocessor Operating Systems, April - June, 1997, pp 60 -73.
- ^[31] T. von Eicken, et. al, *Active Messages: a mechanism for integrated communication and computation*, Procs. of International Symposium on Computer Architects, 1992.
- ^[32] *HPVM* - <http://www-csag.cs.uiuc.edu/projects/clusters.html>
- ^[33] S. Parkin, et. al., *High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance*. Eighth SIAM Conference on Parallel Processing for Scientific Computing, March, 1997.
- ^[34] *MPI Test Suites* - <http://www.mcs.anl.gov/Projects/mpi/mip-test/tsute.html>
- ^[35] *IBM Test Suite* - <ftp://info.mcs.anl.gov/pub/mpi/mpi-test/ibmtsuite.tar>
- ^[36] *Euroben* - <http://www.fys.ruu.nl/~steen/>
- ^[37] *NAS Parallel Benchmarks* - <http://science.nas.nasa.gov/Software/NPB/>
- ^[38] *ParkBench* - <http://www.netlib.org/parkbench.html>
- ^[39] <http://www.sis.port.ac.uk/Papers/PC-NOW>