WMPI
Message Passing Interface for Win32 Clusters

José Marinho[†] and João Gabriel Silva[‡]

[†]Instituto Superior de Engenharia de Coimbra, Portugal
fafe@isec.pt
[‡]Departamento de Engenharia Informática, Universidade de Coimbra, Portugal
jgabriel@dei.uc.pt

Abstract. This paper describes WMPI[1], the first full implementation of the Message Passing Interface standard (MPI) for clusters of Microsoft's Windows platforms (Win32). Its internal architecture and user interface are presented, along with some performance test results (for release v1.1), that evaluate how much of the total underlying system capacity for communication is delivered to the MPI based parallel applications. WMPI is based on MPICH, a portable implementation of the MPI standard for UNIX® machines from the Argonne National Laboratory and, even when performance requisites cannot be satisfied, it is a useful tool for application developing, teaching and training. WMPI processes are also compatible with MPICH processes running on Unix workstations.

1. Introduction

Parallel platforms based on heterogeneous networked environments are widely accepted. This kind of architecture is particularly appropriate to the message-passing paradigm that has been made official by the Message Passing Interface standard (MPI) [1]. Some relevant advantages over massively parallel machines are availability and excellent competitive performance/cost ratios, and the main disadvantage relies on the underlying networks and communication subsystems that are not optimised for message exchange performance but reliability and low cost.

Most of the researchers presently interested in parallel programming (and probably in the future) may not have access to massively parallel computers but generally do have networks of both PC and UNIX machines that are able to communicate with each other. MPI libraries were originally available for clusters of UNIX workstations with the same or even lower capabilities than the present personal computers (PCs) running the Microsoft Win32 operating systems (Win32 platforms). Being these PCs

[1] This work was partially supported by the Portuguese Ministério da Ciência e Tecnologia, the European Union through the R&D Unit 326/94 (CISUC), the project ESPRIT IV 23516 (WINPAR) and the project PRAXIS XXI 2/2.1/TIT/1625/95 (PARQUANTUM)

almost everywhere and having reached competitive levels of computational power [2,3], there was no reason to keep them out of the world of parallel programming. Additionally, since many local area networks (LAN's) consist of a mix of PC and UNIX workstations, protocol compatibility between UNIX and Win32 MPI systems is an important feature. These considerations lead to the development of the WMPI package, the first MPI implementation for clusters of Win32 machines, first released on April 1996. WMPI provides the ability to run MPI programs on heterogeneous environments of Win32 (Windows 95 and NT) and UNIX architectures. The wide availability of Win32 platforms makes WMPI a good learning and application development tool for the MPI standard.

Section 2 introduces the bases of the WMPI package development. Then, section 3 shortly describes how to use WMPI for developing and running parallel applications. In section 4 some internal details are explained and, finally, the performance of WMPI is evaluated in section 5 with clusters based on Win32 platforms.

2. Design Philosophy

MPICH [4,5], a message passing implementation from Argonne National Laboratory/Mississippi State University, is fully available for general-purpose UNIX workstations and it enables heterogeneous UNIX platforms to cooperate using the message-passing computational model. Therefore, an MPICH compatible WMPI implementation was considered to be the most appropriate and time-effective solution for the integration of Win32 and UNIX platforms into the same virtual parallel machine.

MPICH has a layered implementation. The upper layer implements all the MPI functions, is independent of the underlying architecture and relies on an Abstract Device Interface (ADI) that is implemented to match a specific hardware dependent communication subsystem [6,7]. Depending on the environment, the latter can be a native subsystem or another message passing system like p4 or pvm.

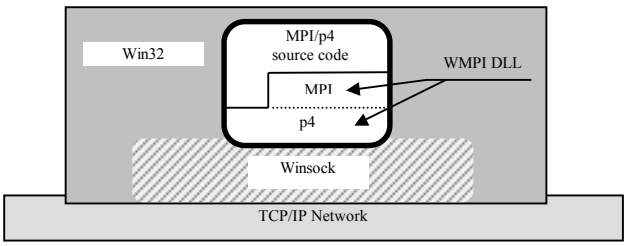

Fig. 1. WMPI and Wp4 process structure

For the sake of compatibility with UNIX workstations and to shorten development time, p4 [8], an earlier portable message passing system from the Argonne National Laboratory/Mississippi State University, was chosen because it is the communication

subsystem that is used by MPICH for TCP/IP networked UNIX workstations (MPICH/ch_p4). Most of the porting work is just concerned with p4, being this layer the only one that directly works on top of the operating system.

3. User Interface

WMPI consists of dynamic link libraries, for console and GUI Win32 applications, that offer all the MPI and p4 application programming interfaces (API) with C/C++ and Fortran 77 bindings, and a daemon that runs in each Win32 machine for automated remote starting. MPI and p4 programs written for UNIX require almost no changes except for UNIX specific system calls (e.g., fork()'s), which are not very frequent in this type of applications anyway.

3.1. Startup

WMPI and Wp4 application's startup and configuration are similar to the original p4 communication system, being every process of a parallel application statically defined in a process group configuration file.

3.2. Remote Starting

For remote starting, the original MPICH and p4 UNIX systems try to contact two specific daemons on target machines - the p4 secure server and the p4 old server. A compatible implementation of the p4 old server daemon is available for WMPI as an application and as an NT service. If this server is not available, WMPI tries to contact a remote shell daemon in the remote machine.

4. Internal Architecture

Some internal details are described in this section. As mentioned earlier, the p4 layer handles most of the platform dependent features. Hence, the major concern of the following discussion is related to this communication subsystem.

4.1. Compatibility and Heterogeneity

For the sake of compatibility with MPICH/ch_p4 for UNIX, the same message structures and protocols are kept for communication between distinct clusters.

Communication can be established between a pair of nodes with different internal data representations. To deal with this situation appropriate data conversions must be performed on message contents to match the destination format. As with the original systems, WMPI and Wp4 processes are aware of data representation for the other processes and handle it in a transparent way to the users. Data conversion only occurs when strictly necessary and a subset of the standard XDR protocol has been

implemented for that purpose, although the MPI layer just uses simple byte swapping whenever it is possible.

4.2. Local Communication

Messages that are sent between WMPI processes in the same cluster (set of processes running in the same machine) are internally exchanged via shared memory, with each process having a message queue. For that purpose, each cluster has a private large contiguous shared memory block that is dynamically managed (global shared memory) using the monitor paradigm. The Win32 API provides some efficient and simple mechanisms to allow the sharing of resources between processes despite of distinct virtual address spaces and contexts [9].

4.3. Remote Communication

Communication between distinct clusters is achieved through the standard TCP protocol that provides a simple and fast reliable delivery service. To access the TCP protocol a variation of BSD sockets called Windows sockets or simply Winsock, which was approved as a standard for TCP/IP communication under MS Windows, is used.

For every process, a dedicated thread (network receiving thread) receives all the TCP incoming messages and puts them into the corresponding message queue. As a result, MPI receive calls just test the message queues for the presence of messages.

4.4. Performance Tuning

WMPI is designed to avoid any kind of active waiting. Any thread that starts waiting for some event to occur stops competing for the CPU immediately and does not use its entire quantum. As an example, when its message queue is empty, a process that makes a blocking receive call stops waiting for a semaphore that is in a non-signaled state (counter equal to zero). Then, just after adding a message to the empty queue, the network receiving thread or a local process turn the semaphore into a signaled state, then enabling the waiting process.

5. Performance Evaluation

The main goal of this section is to quantify the efficiency of WMPI (release v1.1) for delivering the underlying communication capacity of a system to the applications. Also, results obtained with single and dual Pentium boxes are compared.

5.1. Testbed

All the machines involved in the experiments (Table 1) are hooked together by dedicated 10 Base T or 100 Base T Ethernet hubs. Some sources of significant

overhead (software and hardware) already exist between the transmission physical medium and the Winsock interface that is used by WMPI. To quantify the real overhead that the WMPI layer is responsible for, some of the experiments are repeated directly on top of the Winsock (TCP/IP) interface. Every socket is configured with the TCP_NODELAY option, as in the Wp4 layer, in order to avoid small messages to be delayed by the TCP protocol (default behaviour).

Table 1. Machines used for the experiment

CPU	OS	RAM	#
Dual Pentium Pro 200Mhz	NT Server	128 MB	1
Dual Pentium Pro 200Mhz	NT Workstation	128 MB	1
Single Pentium Pro 200Mhz	NT Workstation	64 MB	4

5.2. MPI Benchmark Tests

Message passing overhead is the main bottleneck for speedup. Some results that have been obtained with the Pallas MPI benchmarks (PMB) [10] are reported here.

Table 2. Pallas MPI benchmark tests

PingPong	Two messages are passed back and forth between two processes (MPI_Send/ MPI_Recv).
PingPing	Two messages are exchanged simultaneously between two processes (MPI_Sendrecv).
Xover	Two messages are sent and received in reverse order (MPI_Isend/ MPI_Recv).
Cshift	A cyclic shift of data along a one-dimensional torus of four processes (MPI_Sendrecv).
Exchange	In one-dimensional torus of four processes, each process sends a message to its right neighbour and then to the left one (MPI_Isend). Then, it receives a message from its left neighbour and then from the right one.

Every test is repeated with messages of variable length. For each message length, ranging from 0 bytes to 4 Mbytes, tests are repeated several times[2] to smooth network fluctuations. Latency is half (for PingPong, PingPing and Cshift) or a quarter (for Xover and Exchange) the measured average time to complete.

5.3. Remote Communication

Tables 3 and 4 depict some results of the Ping-Pong test with processes running on distinct machines. A pair of Dual Pentium machines and another of Single Pentium machines are separately used. For this latter, an equivalent Ping-Pong test that is directly implemented on top of the Winsock interface is also executed. The overhead columns of these tables represent the percentage of the available bandwidth at the Winsock interface (the Winsock columns) that, because of its internal operation, the WMPI layer cannot deliver to end-users.

---

[2] For message lengths up to 256 Kbytes: 100 times. For message lengths equal to 512 Kbytes, 1 Mbyte, 2 Mbytes and 4 Mbytes: 80, 40, 20 and 10 times respectively.

Table 3. Bandwidth with a 10 Base T hub

Size	Pair of Dual Pentiums	Pair of Single Pentiums		
(bytes)	WMPI (Mbps)	WMPI (Mbps)	Winsock (Mbps)	Overhead (%)
1	0.01	0.01	0.04	63.6
4	0.06	0.06	0.16	63.6
16	0.23	0.23	0.64	63.6
64	0.73	0.93	2.05	54.5
256	2.40	2.93	5.12	42.9
1024	5.81	6.07	7.80	22.2
4096	7.49	7.98	8.84	9.7
8192	7.99	8.49	8.97	5.2
32768	8.30	8.75	8.95	2.2
65536	8.34	8.81	8.93	1.3
131072	8.28	8.81	8.94	1.5
262144	8.27	8.79	8.94	1.7
1048576	8.27	8.77	8.94	1.9
4194304	8.18	8.74	8.93	2.2

Table 4. Bandwidth with a 100 Base T hub

Size	Pair of Dual Pentiums	Pair of Single Pentiums		
(bytes)	WMPI (Mbps)	WMPI (Mbps)	Winsock (Mbps)	Overhead (%)
1	0.02	0.02	0.05	66.7
4	0.05	0.07	0.21	66.7
16	0.26	0.28	0.85	66.7
64	0.94	1.14	3.41	66.7
256	2.71	4.50	13.65	67.0
1024	14.89	16.38	32.77	50.0
4096	34.86	36.41	59.58	38.9
8192	39.60	48.37	68.62	29.5
32768	51.35	68.89	71.72	3.9
65536	50.83	68.85	74.79	7.9
131072	49.53	69.10	85.11	18.8
262144	51.92	70.05	86.71	19.2
1048576	49.48	71.14	87.94	19.1
4194304	45.50	63.88	87.48	27.0

As expected, WMPI is less performing than the Winsock interface. For small messages the message size independent overhead of WMPI (e.g., constant-size header fields) gives rise to overhead values over 50% in tables 3 and 4. For large messages, the copying of data bytes between WMPI internal buffers and the application allocated buffers is one of the main contributions for the overhead. It doesn't depend on the available bandwidth because only internal processing is included. Thus, the overhead of WMPI for large messages is much higher with the 100 Base T connection. For larger messages, performance starts decreasing because memory management (e.g., copying) gets less efficient for larger blocks.

The Dual Pentium boxes always perform worst than the single ones. A possible reason may be that some data has to be exchanged between processes (e.g., between a WMPI process and the TCP service provider) that are, possibly, running on different processors. Thus, data that has to be exchanged between distinct processes is not found in the data cache of the destination processor.

As a conclusion and despite of some significant overhead, it can be concluded that WMPI is able to give a significant portion of the underlying available bandwidth to the applications. Encouraging maximum values of 8.8 Mbps and 71.14 Mbps are obtained with 10 Base T and 100 Base T connections, respectively.

5.4. More Communication Patterns.

Other PMB benchmarks (PingPing, Xover, Cshift and Exchange) have been also executed with four single Pentium Pro machines and a 100 Base T hub (table 5).

Table 5. Bandwidth (Mbps) with a 100 Base T hub and 4 single Pentium Pro machines

Size (bytes)	PingPong (2 proc.)	PingPing (2 proc.)	Xover (2 proc.)	Cshift (4 proc.)	Exchange (4 proc.)
1	0.02	0.02	0.02	0.02	0.02
4	0.07	0.09	0.08	0.06	0.10
16	0.28	0.37	0.32	0.32	0.39
64	1.14	1.26	1.36	1.28	1.46
256	4.50	5.12	5.12	4.95	5.74
1024	16.38	18.20	20.35	18.20	20.74
4096	36.41	46.81	42.28	32.73	32.71
16384	60.82	65.37	60.89	36.36	35.13
65536	68.85	74.26	66.07	33.24	38.55
262144	70.05	76.71	64.19	34.73	35.68
524288	70.54	77.38	65.34	38.26	36.79
1048576	71.14	77.03	65.80	39.82	40.51
2097152	69.44	75.21	63.97	40.10	39.81
4194304	63.88	69.66	60.02	39.78	39.17

When compared to the Ping-Pong test results, only Cshift and Exchange experience a significant difference for messages up from 4 Kbytes. Being Cshift and Exchange the only tests that make the four processes access the network bus simultaneously to send messages, the increased number of collisions is the main reason for that performance loss.

5.5. Local Communication

Table 6. Bandwidth for local communication

Size	Pair of Dual Pentiums	Pair of Single Pentiums		
(bytes)	PingPong (Mbps)	PingPong (Mbps)	PingPing (Mbps)	Xover (Mbps)
1	0.05	0.16	0.16	0.10
4	0.40	0.64	0.64	0.43
16	1.60	2.56	2.56	1.71
32	1.65	5.12	2.56	3.41
128	13.65	20.48	10.24	13.65
512	26.43	81.92	40.96	54.61
2048	105.70	163.84	163.84	163.84
8192	168.04	327.68	327.68	262.14
16384	278.88	524.29	524.29	403.30
32768	280.37	655.36	582.54	386.93
131072	150.77	282.64	265.13	173.75
262144	150.82	238.04	233.93	177.50
1048576	147.07	227.87	222.58	174.29
4194304	145.10	220.46	215.44	173.16

Table 6 depicts some results with two processes running on the same machine. As expected, communication between two processes running on the same machine is much more efficient than remote communication because it is achieved through shared memory. It is also visible that the already noticed performance discrepancy between Dual and Single Pentium boxes and performance decreasing is greatly enhanced. With just a single processor the probability of a receiving process to get a

message, or part of it, from its local data cache is very high because local communication between two WMPI processes is exclusively based on shared data.

6. Conclusions

WMPI fulfills the goals outlined at the beginning of this document, i.e., an MPI support for widely available Win32 platforms that widespread this accepted programming model. It also enables cooperation between low cost Win32 machines and UNIX ones, to offer accessible parallel processing. Additionally, the download of WMPI (http://dsg.dei.uc.pt/w32mpi) by more than 1700 different institutions (until March 98) since its first release (April 96) demonstrates how real is the interest for MPI based parallel processing under Win32 clusters and how valuable and useful has been the development of WMPI.

Presently there are a few other available implementations of MPI for Windows, but WMPI is still the most efficient and easy to use MPI package for Win32 based clusters [11]. More complex communication patterns of some real applications can result in higher communication overheads. Nevertheless, the expected performance is promising due to a positive evolution of the interconnection technologies and of the individual computational power for Win32 platforms.

References

1. Message Passing Interface Forum, “MPI: A Message-passing Interface Standard”, Technical report CS-94-230, Computer Science Dept., University of Tennessee, Knoxville, TN, 1994
2. Tom R. Halfhill, “UNIX vs WINDOWS NT”, Byte magazine, pp. 42-52, May 1996
3. Selinda Chiquoine and Dave Rowell, "Pentium Pro Makes NT Fly", Byte magazine, pp. 155-162, February 1996
4. William Gropp, “Porting the MPICH MPI implementation to the sun4 system”, January 12, 1996
5. P. Bridges, et. Al., “User’s Guide to MPICH, a Portable Implementation of MPI”, November 1994
6. William Gropp, Ewing Lusk, “MPICH Working Note: Creating a new MPICH device using the Channel interface - DRAFT”, ANL/MCS-TM-000, Argonne National Laboratory, Mathematics and Computer Science Division
7. William Gropp, Ewing Lusk, “MPICH ADI Implementation Reference Manual - DRAFT”, ANL-000, Argonne National Laboratory, August 23, 1995
8. Ralph Butler, Ewing Lusk, “User’s Guide to the p4 Parallel Programming System”, Argonne National Laboratory, Technical Report TM-ANL-92/17, October 1992, Revised April 1994
9. Jeffrey Richter , “ADVANCED WINDOWS, The Developer’s Guide to the Win32 API for Wondows NT 3.5 and Windows 95”, Microsoft Press, Redmond, Washington, 1995
10. Elke Krausse-Brandt, Hans-Christian Hoppe, Hans-Joachim Plum and Gero Ritzenhöfer, “PALLAS MPI Benchmarks – PMB”, Revision 1.0, 1997
11. Mark Baker and Geoffrey Fox, “MPI on NT: A Preliminary Evaluation of the Available Environments”, “http://www.sis.port.ac.uk/~mab/Papers/ PC-NOW/”, November 1997