The WMPI Architecture for Dynamic Environments and Simultaneous Multiple Devices[1]

Hernâni Pedroso, Pedro Silva and João Gabriel Silva

Dependable Systems Group
Dept. Engenharia Informática
Universidade de Coimbra – Polo II
3030-397 Coimbra
Portugal
{hernani,ptavares,jgabriel}@dei.uc.pt

Abstract

WMPI was originally based on the MPICH implementation. MPICH's architecture, though well fitted for portability and performance, is not flexible enough to allow the implementation of the new functionality introduced in the MPI-2 standard. Due to this difficulty, we decided to develop a new architecture from scratch. This architecture introduces new concepts that allow for the dynamic creation and termination of processes as well as the usage of simultaneous multiple devices. The new architecture was designed to be completely thread safe and to allow a rapid development of libraries for any communication medium. This paper describes the reasons for developing thr new architecture and presents the implemented solutions to achieve the proposed goals.

Keywords. ***Cluster Computing, MPI, Parallel Computing, MPI-2, WindowsNT***

1. Introduction

The first version of the MPI standard [1] only proposed a specification about how the processes communicate between each other. This decision was on purpose, since the standard tried to unify the functionality present in the several existent libraries and obtain a consensus in the high-performance computing community. MPI was well accepted by the community and became a *de facto* standard for parallel computing. Eager for more functionality, the users and developers presented several requests to the MPI Forum [2] to increase the functionality of the MPI standard. Version 2.0 [3] of the standard included several new types of functionality, from which one-sided communication and dynamic process creation are examples. Although the users and developers embraced these new chapters, they require deep changes in most of the existing MPI libraries.

The Argonne National Laboratory/Missisipi State University developed MPICH [4] alongside with the standard's first version. This library aimed to implement all the functionality specified by the standard in an efficient and portable fashion. Due to its characteristics, MPICH served as a development base for many other implementations, which addressed different operating systems and architectures.

WMPI (Windows Message Passing Interface) [5,6] was the first full implementation of the MPI standard for Windows operating systems. The first versions of WMPI were strongly based on the MPICH implementation. It used the Abstract Device Interface (ADI) [7] to interact with the communication subsystem. A Win32 port of the p4 [8] library was used to setup the environment and manage the communication between processes. Although the base

[1] This work was partially supported by the Portuguese Ministry of Science and Technology through the R&D Unit 326/94 (CISUC) and the project PRAXIS XXI 2/2.1/TIT/1625/95 named ParQuantum

architecture was never changed, we conducted several modifications during the WMPI lifetime to improve performance and usability. In a recent study, which evaluated implementations of MPI for Windows NT environment [9], WMPI was considered the best freely available implementation. In addition, the study concluded that WMPI rivals with other commercial implementations in performance and functionality.

While evolving the WMPI library to MPI-2, we faced several limitations in the used architecture. Although the architecture is simple and well structured, it did not have the necessary flexibility to introduce the necessary changes for implementing the dynamic creation of processes. Necessarily we designed a complete new structure for the WMPI library. During the several years of development and maintenance, other pitfalls of the architecture were identified. For example, it was not thread safe or able to use more than one device simultaneously. This paper presents the new WMPI architecture, which fulfills the new requirements. This architecture is the base for the WMPI 2.0, an MPI-2 compliant version of the WMPI library.

This paper first presents the architecture issues of MPICH, inherited by the first WMPI versions that were making difficult the evolution of the library. Then presents the new architecture. Further on, its objectives and the design decisions. How the library manages more than one device simultaneously and a dynamic environment is extensively presented. Then an analysis about the communication progress in the library. Finally, a comparison with related work and some conclusions are presented.

2. MPICH Architecture Issues

One of the objectives of the MPICH's design was to make a MPI library easily portable to other platforms, though efficient. This objective obviously had a strong impact in the library architecture. The portable MPI management layer excluded all the functionality that required machine dependent code. The WMPI version used a port of the MPICH most common device, the p4 [10] library, to implement the machine dependent code. The setup of the MPI environment is dependent from the underlying platform. The whole MPI environment considers that somehow the processes were created and that they all will be present until the end of the computation. There is no control on how the processes are created, destroyed or how they communicate. This was a good decision considering the static environment of MPI-1. However, this scenario changed in the MPI-2 version. Since there is no mechanism to control the creation of processes, it is not possible to create processes dynamically.

The MPICH architecture also emphasizes its Unix origins. It uses only one thread per MPI process and works with one type of communication medium at a time. These design characteristics also helped the portability of the library, since not all platforms have threads. Although, some mechanisms were implemented to use more than one device, in fact they were very poor and hard to use. Since only one thread was available, the library had to make polling on the devices to get new messages. In addition, the message passing progress could only be possible when the user thread made a call to a MPI function. This architecture's type is well fitted for supercomputers or dedicated clusters where there is only one process per CPU. However, the typical NT cluster is shared among several users, which may be executing interactive tasks.

3. The New WMPI Architecture

MPI-2 introduced completely new features into the standard. Some of the new types of functionality required deep changes in the library structure. We decided to create a completely new architecture to avoid the implementation of successive patches, which are more prone to errors and make the evolution difficult. Based on our experience with cluster computing and considering the trends on the field, several other reasons were identified that strongly required a new architecture. The design aimed to create an architecture that is able to:

- manage a dynamic environment, where processes could be created and destroyed, join and leave the MPI computation;

- work with several devices simultaneously;
- easily support new communication technologies by reducing the complexity, hence the development time, of the communication dependent code
- be completely thread safe, and;
- diminish the communication latency.

The new architecture does not try to be portable across other platforms. Since the WMPI library is used only in Windows environments, we decided to use the features that the operating system provides. This reduces the complexity and increases the performance of the library.

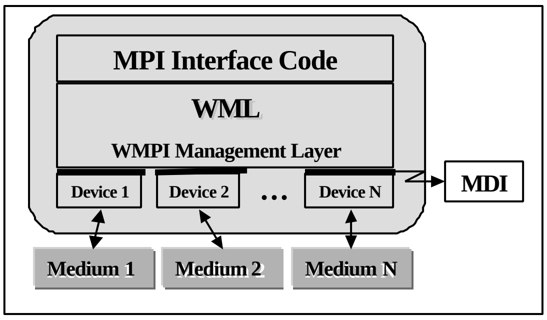


Fig. 1. The WMPI structure.

Figure 1 presents the structure of the WMPI library. The upper layer implements the MPI interface. The WMPI Management Layer (WML) is responsible for managing the MPI environment. The WML interacts with the existing devices through a generic interface, called Multiple Device Interface (MDI). These devices implement the operations, whose implementation is dependent from the communication medium. The WML concentrates most of the features that allow the objectives of the design to be achieved.

Thread safeness and performance were a constant concern during the design and implementation of the library's architecture. Every structure and all the functions that manipulate its data were studied to verify if they have or not to be thread safe. The synchronization points were reduced to the minimum to improve the general performance of the library.

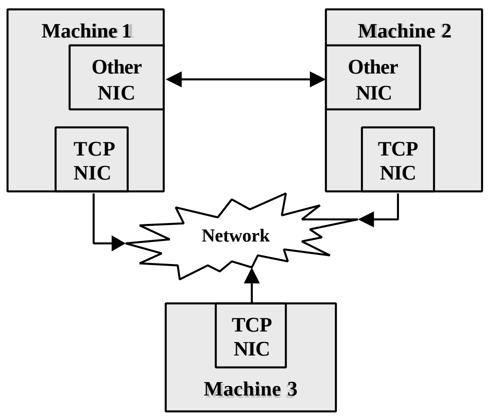
However, one of the architecture goals is that it must easily support new communication technologies. Through an analysis of the library requirements, the operations that have to be performed by medium dependent code have been identified. To diminish the complexity of the communication medium dependent code (device), the set of operations that the WML requires is quite simple. The simplicity of the required functionality increases the number of technologies that can be used with WMPI. The implementation of a device does not require any knowledge of the library internals. It is only necessary to follow the MDI specification that is available at the WMPI web site [2]. This allows virtually anyone to implement a device for WMPI. For example, technology vendors and university researchers can create devices for new technologies or devices that have a better performance in specific environments. We expect that this characteristic will allow the WMPI to be available in a broad set of technologies.

The next chapters present how the WMPI manages to work within a dynamic environment using multiple devices, the two major enhancements of the library.

4. Multiple Devices Management

It is common to find more than one type of technology for communication in a cluster. The most common configuration is, probably, shared memory and TCP/IP. However, other configurations are possible and some of them may use more than two different communication mediums. WML has the capability to interact with any number of different devices simultaneously. Within each process, WML associates a device to each machine of the cluster, according to a cluster configuration. When one MPI process needs to interact with another process, it chooses the correct device and performs the necessary action.

The user is responsible for defining how processes communicate during the WMPI computation through a cluster configuration file. Figure 2 presents an example of such a file. The Windows machine name is the identifier of each machine. For each machine, the user has to indicate which devices the processes running on that machine can communicate with and the identification of the machine using that device. Devices are now independent DLLs that are loaded at the startup of each process according to the cluster configuration.



```
/Machines
machine1
tcp 193.145.134.21
other other_addr1
shmem machine1
machine2
tcp 193.145.134.11
other other_addr2
shmem machine2
machine3
tcp 193.145.134.31
shmem machine3
/Devices
default internal shmem
default external tcp
machine1 machine2 other
```

Fig. 2. Example of a cluster definition**.**

During the design of the new architecture, we identified the necessary operations that the devices should perform. It was important to reduce the expected functionality of the devices to a minimum, since it should be feasible to produce a device for every possible technology. Moreover, simple devices are easier to optimize and guarantee the independence of the library core. The operations can be grouped in four categories:

- **Environment**: The environment functions are used to initialize and finalize the devices. The devices use these functions to allocate and free the necessary resources.
- **Process Creation**: The only communication medium that the WML knows to be available to contact the other machine is the device specified in the cluster's configuration. Hence, WML relies on the devices to create new processes.
- **Connections Management**: WML is connection oriented. A connection is opened between each pair of processes and is maintained while the processes are connected (have a common group/communicator).
- **Communication**: Functions to send and receive data are available. To send data two functions are available, a blocking send and a non-blocking send. When receiving data, the devices can use WML functions to access the receiving queues and get information about the message they are receiving. This way it is possible to receive the message directly from the communication medium into the user buffer.

5. Dynamic Environment Management

MPICH used the `MPI_COMM_WORLD` rank to identify the processes in the computation. WMPI inherited this characteristic. Since the first version of the MPI standard implied a static environment, all the existent processes belonged to this communicator. However, with the introduction of dynamic creation of processes, it is possible for the existence of processes in the computation that do not belong to the `MPI_COMM_WORLD` and with which messages can be exchanged. These processes are the result of a spawn or the joining of two MPI computations, using `MPI_Comm_connect` or `MPI_Comm_join`. These processes can enter and leave the computation in runtime. When a spawn is performed or two MPI computations join not all the processes of both computations have to be involved. This means that each process will have its own set of processes with which it can communicate. Although when the new processes join, the communication has to be performed through an inter-communicator, they can form an intra-communicator using the MPI_Intercomm_merge function. It is also possible to extract the remote group of processes in an inter-communicator and manipulate it as any other group. This implies that an inter-communicator can be closed (which should indicate that the two computations are independent) but some of the processes still communicate, through groups and intra-communicators that were built using the inter-communicator. This situation is an example of the extreme freedom that the users have to manipulate the processes and create the configuration of interconnections and communicators. The usage of ranks to globally identify the processes is impracticable in such an environment. It is necessary to create another form of global identification, which must be valid whether the process starts within the same computation or joins in runtime.

Each process has a set of devices through which it may receive and send messages. The set of devices is defined in the cluster configuration file as the ones that are available in the machine where the process is running. For each device, the process has an address that is used by the other processes to communicate with it. This address must be unique in each device. The set of addresses of the process in the devices available, is a unique key that represents the process in the whole computation, since no other process can have the same addresses in the same devices.

Each process contains a structure per process with which it is connected. This structure, or record, contains the set of addresses of the process that it represents. Each record also contains a reference to the device that must be used to communicate with the process. The set of the process's records is called the WorldView (Figure 3). It represents all the processes in the computation with which this process has at least one group in common.

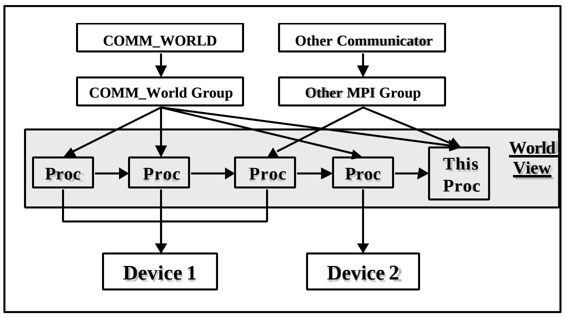


Fig. 3. The WMPI process's World View.

The `MPI_Groups` have an array of references to the process records that belong to the group. The rank of each process is defined by the index of its reference in the array. Each process record has a reference counter that indicates the number of groups that the processes have in common. When this counter reaches zero, it means that there are no more groups in common and hence the two processes are disconnected. In this case the WML removes the process record from the WorldView and erases it from memory.

The comparison between processes at the MPI level was made, in the first version of WMPI, by the rank of the processes in the MPI_COMM_WORLD communicator. Now the memory address of the process record is used. This maintains the speed of comparison. When a new process joins in runtime, WML verifies which device must be used to communicate with it. Then the WorldView is searched to verify if a process record that uses that device and has the same address already exists. If it finds one it means that the process already belongs to the WorldView and no new process record is created.

This design works in a completely dynamic environment. Any process can leave the WorldView if no more common groups exist. Even the processes that belong to the same initial set could be removed if the standard allowed for the destruction of the MPI_COMM_WORLD communicator. The design only requires one communicator to be indestructible, the MPI_COMM_SELF. If this communicator could be destroyed, it would be possible to erase the process record that contains the information of the current process.

6. Communication Progress

The WML does not use any specific thread for communication progress. The communication progress depends upon the implementation of the devices used. The architecture allows the devices to have or not threads to receive and send messages.

Functions of the WML are available to get information about a message that is being received. The device is able to know if the message that it is receiving is expected or not. If expected the data can be copied directly from the communication medium into the user's buffer. This allows for a completely asynchronous reception of the messages. The

MDI also specifies a function to send a message asynchronously. When this function is used, the user's thread does not wait for the completion of the send. The device is responsible for sending the message concurrently with the work of the user's thread and for notifying to the WML, through a WML specific function, when the send is complete. The user's thread verifies the completion of the message, for example when it calls the MPI_Wait function, through the state of the WML data.

If the device has threads for send and receive messages then the communication progress complies with the strictest interpretation of the MPI Progress Rule [11,12].

However, the existence of threads in the devices is not required by the WML. It was considered that if a thread is not necessary, then it represents an added latency, due to context switching, to the communication through that device. If the device does not have a receiving thread, WML uses a receive function (specified in the MDI for this type of devices) to receive messages from the device. To avoid WML from polling over the several devices to get messages, the devices have to use a synchronization system to indicate to the WML that messages are available.

The WML can also work with devices that use polling to receive data. However, in this case only one device can exist in the system. This type of devices is common on SMPs were processes communicate only through shared memory.

7. Related Work

Few MPI libraries implement the interface for dynamic process creation specified in the MPI-2 standard. An implementation from Pallas GmbH for Fujistu machines [13] and the LAM/MPI library from Notre Dame [14] already implement this functionality. None of them works in the Windows NT environment. Through this new architecture, WMPI is able to manage a dynamic environment. Hence to create new processes in run-time.

MPI/Pro [15] is able to run over more than one communication medium and is thread safe. However, it does not use more than one device simultaneously and the users do not have the same ease as with WMPI to configure the computation, to fit the cluster configuration.

The MPICH-NT [16], MP-MPICH [17], FM-MPI [18] and PaTENT MPI [19] cannot use more than one device simultaneously. Moreover, none of these implementations has support for thread safeness and dynamic process creation.

In contrast to WMPI, none of the libraries has a strong independence between the library core and the devices and all present a different implementation for each communication medium. In addition the development of other devices requires a thorough knowledge of the library.

8. Conclusions

WMPI was originally based on the MPICH implementation. MPICH's architecture, though well fitted for portability and performance, is not flexible enough to allow the implementation of the new functionality introduced in the MPI-2 standard. This difficulty was the most important reason that led to the development of a new architecture.

The architecture was designed to be completely thread safe and to allow a rapid development of the library for any communication medium. One of the biggest achievements of the architecture is to enable anyone to easily develop his or her own device. This can be done because the developers do not have to have any knowledge about the library core. Communication medium vendors or any other research institutions can develop their own device implementation for the WMPI library. This should diminish the time to have the WMPI available over new technologies and to have better devices.

WMPI is now able to work with more than one device simultaneously. The devices are independent DLLs that are loaded according to the cluster configuration, at the startup of the MPI process. The user is able to define which devices are used in the computation. The library has thus an enormous flexibility to adjust to the user's clusters.

The new architecture introduces to the library the ability to manage a dynamic environment, where processes can be added and deleted from the computation in runtime.

The implementation of the MPI-2 standard's process creation interface is now possible and is under way.

The WMPI 1.5 version, which uses this new architecture, is already available at the WMPI's web site [6].

References

[1] Message Passing Interface Forum: MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4):165-414 (1994).

[2] Message Passing Interface (MPI) Forum Home Page, http://www.mpi-forum.org.

[3] Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. (June 1997), available at http://www.mpi-forum.org.

[4] Gropp, W., Lusk, E., Doss, N. and Skejellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing Vol. 22, No. 6, (September 1996).

[5] Marinho, J. and Silva, J.G.: WMPI – Message Passing Interface for Win32 Clusters. Proc. of 5th European PVM/MPI User's Group Meeting, pp.113-120 (September 1998).

[6] WMPI Homepage – http://dsg.dei.uc.pt/wmpi.

[7] Groop, W. and Lusk, E.: MPICH ADI Implementation Reference Manual – DRAFT. ANL-000, Argonne National Laboratory, Mathematics and Computer Science Division (August 1995).

[8] Butler, R. and Lusk, E.: Monitors, messages and clusters: The p4 parallel programming system. Parallel Computing, 20:547-564 (April 1994).

[9] Baker, M: MPI on NT: The Current Status and Performance of the Available Environments. NHSE Review, Volume 4, No 1 (September 1999).

[10] Butler, R. and Lusk, E.: Monitors, messages and clusters: The p4 parallel programming system. Parallel Computing, 20:547-564 (April 1994).

[11] Message Passing Interface Forum: A Message-Passing Interface Standard. Section 3.7.4 (1994).

[12] Herbert, L.S., Seefeld, W., Skjellum, A., Taylor, C.D. and Dimitrov, R.: MPI for Windows NT: Two Generations of Implementations and Experience with the Message Passing Interface for Clusters and SMP Environments. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. 1, pp: 309-317, Las Vegas, Nevada (July 1998). Asai, N., Kentemich, T. and Lagier, P.: MPI-2 Implementation on Fujitsu generic passing kernel. Proceedings of Supercomputer 99, Portland, Oregon (November 1999).

[13] LAM / MPI Parallel Computing - http://www.mpi.nd.edu/lam/

[14] MPI Software Technology, Inc – http://www.mpi-softtech.com

[15] MPICH – A Portable MPI Implementation - http://www-unix.mcs.anl.gov/mpi/mpich/

[16] MP-MPICH: Multiple Platform MPICH - http://www.lfbs.rwth-aachen.de/~joachim/MP-MPICH.html

[17] Lauria, M., Pakin, S., Chien, A.: Efficient Layering for High Speed Communication: The MPI over Fast Messages (FM) Experience. Cluster Computing, HPDC7 special issue (1999)

[18] Genias Software GmbH, PaTENT – Parallel Tools Environment on NT, http://www.genias.de/products/patent/index.html