Mpich Communication Protocols: Study of Their Behavior in a Hybrid System

Daniel Alberto Burbano
Advisor: Jaime Seguel

Electrical and Computer Engineering Department
University of Puerto Rico, Mayaguez Campus
Mayagüez, Puerto Rico
dburbano@ece.uprm.edu

Abstract

Clusters of SMP are a complex architecture that requires flexibility in its communications. MPICH implementation of MPI, with its architecture compose by independent layers, is optimal to support it. This study investigates different communication protocol in a cluster of SPM.

1. Introduction

Message passing in a parallel system occurs between processes running on different processors. Communication protocols in parallel systems are group of functions and routines used to send and receive information between processors. Message Passing Interface (MPI) uses protocols from a standard library to create a communication interface. MPICH is a portable suite of MPI implemented for different parallel environments.

The knowledge of the communication protocols’ behavior is crucial to take full advantage of a parallel system. Information about protocol’s routines, their limits, and functions, can be used to improve performance when solving a particular problem. For instance, for some systems, can we use two protocols at the same time, one communicating inside the same machine and other communicating processors on different machine without interference? Is it possible to change the protocol at run time? For example, an algorithm may change from rendezvous protocol to short protocol when its information is too small. The answers to these questions have a definite impact on the performance of a parallel system.

The aims of my research are the following: 1) to verify the influence of protocols for MPICH on Linux installed on a cluster of nodes by changing the standard protocol during MPICH configuration 2) to characterize communication protocols and associate them to algorithms 3) to explore the possibility of creating a multi-protocol environment able to switch protocols at run time.

2. Hardware Architecture

Hardware architecture can be classified as shared memory or distributed memory (see Figure 1). In shared memory, each processor sees a common address space and uses the same media to access resources. Shared memory is divided in Symmetric Multiprocessor (SMP) and Cache Coherence Non Uniform Memory (ccNUMA). In distributed memory, the memory is local to each processor and communication is required for accessing external data. Distributed memory is divided in Massively Parallel Processor (MPP), Beowulf Cluster and Hybrids. A hybrid system is a combination of shared memory and distributed memory. In our case, our system named “Malambo” is composed of 8 nodes with 2 CPUs each, and both CPUs share memory. For this reason our parallel system can be classified as a hybrid or a cluster of SPM.

3. Message Passing Interface

MPI is a library of functions to be used to create message passing programs written in Fortran, C, C++ and Java. MPICH (MPI + Chameleon) is a portable implementation of MPI for different parallel computation systems. It contains visualization and diagnostic tools, as well.

MPICH protocols are used to communicate data between processors either in the same machine or in different machines. Using the most adequate protocol for a parallel system depends on different
characteristics such as data size, algorithm taxonomy, and hardware architecture (shared memory, distributed memory).

Figure 1. Parallel Architecture Hardware

4. Architecture of MPI

The architecture of MPI consists of 3 layers (see Figure 2): Application Programmer Interface (API), Abstract Device Interface (ADI) and Channel Interface (CI). API is the interface between the programmer and ADI. The API uses an ADI to sends and receives information.

The ADI controls the data flow between API and hardware. It specifies if the message is sent or received, handles the pending message queues, and contains the message passing protocols. The message structure used for ADI is composed of a message body, a field that describes the length of the message body, a field of message tag, context-id, and the destination. ADI and message compositions are described in detail in [4-Gropp96, 5-Gropp96]. The ADI can send messages in blocking or non-blocking form (see Figure 3). In **blocking form** the ADI does not return the control to API until the message body is available for use. In a send, this means that the message either is delivered or transferred to memory. In a receive, this means that the message has been received. The **non-blocking form** allows the API overlap communications and computations [Gropp94a].

**Figure 2. Architecture of MPI**

**Figure 3. Blocking and Non-Blocking modes**

4.1 Mpich protocols

CI is implemented by ADI. CI uses protocols to transmit data from a processor to another. There are four protocols in MPICH, namely eager, rendezvous, short, and get. MPICH is configured with eager protocol by default [Gropp96a, Gropp96b].

In **eager protocol**, the sender sends data to the receiver without request. Its performance depends of size data, receiving buffer and handshake. Eager protocol has trouble when large amounts of data are sent without expectation of the receiver, because it allocates the non expected data in main memory [Gropp96a, Gropp96b]. This protocol is used to send short messages [Gropp96a]. In the **rendezvous protocol**, the sender sends the data when the receiver requests them. It
could increase overhead in communications because it uses control message before send the data. This protocol is used to send long messages [Gropp96a].

In **short protocol**, the sender sends the data into the control message. The short protocol is very useful when the data size is small because the data can be included in a field (payload) of the control signal. The payload is the amount of data that can be sent in a control message. Payload can be changed at configuration time with the option `-pkt_size=n`, where `n` is the number of bytes. The configure option `-var_pkt` allows control the payload at run time. For most devices, the default payload size is 1024 bytes [Gropp96b, Gropp94b].

Finally in **get protocol** the receiver just gets the data from the sender. This protocol requires a direct method to transfer data from a process’ memory to another. A typical implementation might use `memcpy` [Gropp96b]. This protocol is useful when the processors are in the same machine.

### 4.2 Channel interface

We can implement or use several CI made for MPICH, for instance, `ch_p4`, `ch_mad`. The `ch_p4` is a message-passing library that implements the ADI. Its CI is implemented with Portable Programs for Parallel Processor (P4), that is, for shared-memory and message-passing models.

Madeleine II is a multi-protocol library that has been integrated into MPICH as an ADI whose its device is `ch_mad` [Mercier00]. Madeleine II uses channel and connection objects to support several network protocols within the same application. The channel has the information of network protocols, NIC and set of connection. The connection handles point-to-point link between two processes belonging to a session [3-Aumage01].

The Madeleine II’s structure is planned on 2 layers (see Figure 4): 1) Buffer management layer, and 2) Transmission modules layer. The Transmission module (TM) selects the optimal Buffer Management Module (BMM) for the communication. TMs are grouped in Protocol Management Modules (PMM) where there is a PMM for each supported protocol [Mercier00] and controls information about drivers, channels and connections of each one.

BMMs control buffers, which are used to storage the information that will be sent or received. These buffers could be static or dynamic. When the buffer depends on TM is called static and when it depends on user is called dynamic.

![Madeleine II's Architecture](image)

Figure 4. Madeleine II's Architecture

### 5. Preliminary Results and Future Work

This document presents the design of MPICH, an implementation of MPI, explain important characteristics of communication protocols. Our proposal is to characterize communication protocols in cluster of SMP on several devices (ADI). MPICH offer flexibility to change from an ADI to another by configuration and allows the configuration of different ADI at same time.

An optimal configuration is `ch_p4` device that is configured with shared memory pattern, which allows working with TCP/IP and shared memory.

A very important characteristic of MPICH is that it allows changing the payload message at configuration time to characterize message length like short format or long format with a threshold size.

Madeleine II is an alternative to characterize dynamic protocols in a cluster of SMP, and then we will evaluate `ch_mad` with other ADIs that are in MPICH.

### 6. References


Distributed Processing Symposium (IPDPS 2001), page 51, San Francisco, April 2001. IEEE.


