Abstract. Most of MPI’s implementations cope with the different underlying means of communication. More than just providing the ability to send a message through a certain protocol the implementations make use of specific features of a protocol to speed up message exchanging. These different communication protocols are integrated with each other and the MPI user does not and should not need to be concerned about it. However, when it comes to One Sided Communications this integration becomes more complicated. Some protocols, like TCP, do not lend themselves to One Sided Communications, while others, like shared memory, are so similar that implementation is trivial. This paper describes the issues we came across when implementing One Sided Communications for an MPI implementation with multi pluggable protocols.

1 Introduction

MPI [1] has become the de facto standard for message passing in parallel computing. Since its release in 1998 it has increasingly been adopted by both industry and academia. Recently its features have been extended [2] with the release of the “MPI-2 standard” [3]. Amongst its most relevant new features is a chapter on single sided communications, i.e. communication that can be done without requiring explicit calls from all processes involved. This single sided communications chapter is a message passing approach to shared memory.

Implementing single sided communications for a shared memory protocol is a simple task since it just requires mapping MPI’s calls to the underlying system’s equivalents [4]. Most vendors have single sided communications implemented only for shared memory systems. Implementing shared memory on other communication protocols like TCP is not as simple, but it has been topic for research [5] and there are even implementations of shared memory over traditional MPI [6]. In terms of MPI-2 single sided implementations for non shared memory systems the authors are only aware of two: [7] and [8].

However, trying to create an implementation that can cope with both at the same time becomes rather complicated. This paper describes the issues involved in developing and implementing an integration for the pluggable protocols for single sided communications. This work uses the SUN implementation of MPI.
SUN has recently made the source code for SUN-MPI available under a community source licence \(^9\). We begin by describing the motivations, followed by an overview of the solution we implemented. After that we discuss the issues involved in this solution. Then some benchmark results are presented. Finally section \(^5\) discusses conclusions which include ongoing and future work.

2 Motivations

In order to provide a complete implementation of MPI-2’s One Sided Communications one must guarantee that an MPI call can be used by all the processes in \texttt{MPI.COMM.WORLD}. The current implementation from Sun only allows single sided communications between processes that are connected through the shared memory protocol. This is an obvious disadvantage and does not meet one of MPI’s goals: portability.

Our implementation overcomes this problem and also integrates the different protocols, thus lifting all previous restrictions to use single sided communications.

Our aim was to implement a generic version of one-sided communications built on top of point to point communications. Ultimately this generic version should be capable of co-existing with optimised protocol specific implementations but provide a fall-back implementation for any protocol that does not implement one-sided communication directly.

One of the examples of the need to use single sided communications with protocols other than shared memory arises when one is intending to use clustered SMPs. Since shared memory cannot be used between the nodes of the cluster the MPI user program would have to be aware of which processes are running on each machine and cope with the fact that some groups of processes cannot use single sided communications between them. Situations like this are what the MPI standard proposes to overcome by making the implementor responsible for dealing with it. Thus it does not specify a way to obtain the information needed to allow the user program to cope with it.

In the following section we present an overview of our approach to solving the problem.

3 The Big Picture

The diagram in figure\(^1\) presents the generic single sided code within the layers of the implementation of MPI we used.

Our generic implementation uses any existing protocol. Whenever the protocol has implemented one-sided functionality this will be used, otherwise our implementation will cover for it. This general purpose implementation’s main feature is an asynchronous agent (Request Agent) which handles the RMA requests.

In this first implementation the agent runs in a thread concurrently with the users’ code and the normal MPI calls.