HPCx
CAPABILITY COMPUTING

Single Sided Communication on HPCx

LAPI and MPI-2

Adrian Jackson

August 14, 2003

Abstract

Single sided communications allow for data transfer without the need for matching send and received operations between processes. This can improve the performance of various types of scientific codes. This report compares the performance of LAPI and MPI-2 single sided communication functions on HPCx. Only the basic `Put` and `Get` functions are benchmarked. These functions are tested using a simple ping-pong benchmark, timing the communication, and recording the amount of data sent. In general LAPI has a better performance for small messages, and MPI-2 outperforms LAPI for large messages.

1 Introduction

The single sided communication model allows one process (the source process) to access data on a remote process (the target process). This enables communications and data transfer, between remote processes, without the need for matching `send` and `receive` operations. Such Remote Memory Access (RMA) functionality is especially useful for codes that involve processing distributed data [2].

There are two main RMA functions that are generally provided by single sided communication libraries, `Put` and `Get`. `Put` is used to copy data from the address space of the the source process to the address space of the target process; `Get` is used to copy data from the target process to the source process.

HPCx is a cluster system, currently composed of a large number of 8-processor shared-memory logical PARtitions (LPARs), connected via an interconnect and associated switch hierarchy. It has two communication libraries, LAPI and MPI, both of which contain single sided communication routines. This report compares the performance, and functionality, of the different methods for single sided communication on HPCx. Sections 1 and 2 describe the main features and implementation details of the communication libraries. Section 3 details the benchmarking tests performed, and the results obtained from them.

2 LAPI

The Communications Low-Level Application Programmers Interface (LAPI) [3] is an IBM library designed to give optimal communication performance on IBM hardware. It provides two communication mechanisms:

- Data Transfer

- Active Message

The *data transfer* functions implement the RMA Put (`LAPI_Put`) and Get (`LAPI_Get`) operations, as well as providing global synchronisation and completion checking functionality. As the `Put` and `Get` operations are inherently unilateral (i.e. the operation initiated by one process does not require the other process to take some complementary action), completion of a RMA function is signalled using predefined counters. These counters are incremented when the operation has completed, and can be checked using `Getcntr` (non-blocking) or `Waitcntr` (blocking) functions.

The active message function (`LAPI_Amsend`) relies on user defined *handlers*, which are invoked and executed in the address space of the target process. As with the data transfer functions, active messages are non-blocking calls which require no complementary action by the target process. When an active message arrives at a target process the user defined `header_handler` is executed. This defines where the data associated with the active message should be copied to, and the address of the `completion_handler`. This second handler is called after the whole active message has been received, and can be used to perform additional processing on the received data.

All LAPI communication relies upon processes *publishing* the addresses of memory that can be accessed remotely. This is done using the `LAPI_Address_init` operation (a collective operation) which allows each process to maintain different memory address maps, but still access portions of each other memory (the alternative would be for each process to have the same memory address map, which is restrictive).

Whilst the active message functionality is beyond the scope of this report and its performance will not be directly evaluated, it is the underlying infrastructure for LAPI, upon which the data transfer functions are constructed.

3 MPI-2

MPI-2 is the extension to the MPI standard that allows, amongst other things, single sided communication [1]. Three communication calls are available:

- *MPI_Put*
- *MPI_Get*
- *MPI_Accumulate*

Put and Get implement the generic RMA functions needed for single sided communication. Accumulate is a Put operation that does not overwrite the data in the target area, but is combined with that data using a specified operation (i.e. +). As with LAPI, MPI uses a collective call to publish the areas of memory that are available for remote access. This call, `MPI_Win_create`, provides a handle that can be used in the communication functions for each process, specifying the start address and size of the memory available for RMA use. These windows may be of different sizes, and in differing locations, for each process, providing that the memory access performed by the communication functions fits within the target window used (i.e. the data transfered must not be large than the window).

To complement these communication operations, there is a range of synchronisation functions provided, supporting different synchronisation styles. `MPI_Win_fence` is a collective call that supports simple, global, synchronisation. The functions; `MPI_Win_start`, `MPI_Win_complete`, `MPI_Win_post`, and `MPI_Win_wait`, can be used to synchronised two processes (the minimal form of synchronisation). Finally, `MPI_Win_lock` and `MPI_Win_unlock` are available to provide passive synchronisation (i.e. emulating a shared memory model).

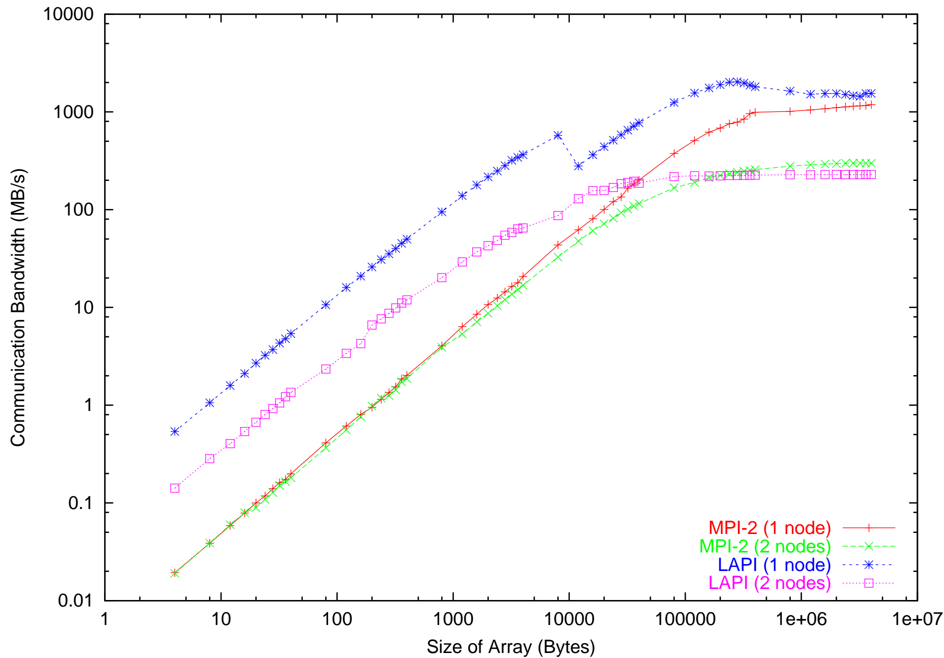

Figure 1: Put Bandwidth

4 Results

The basic RMA functions Put and Get were benchmarked for this report. Whilst all the libraries have specialised RMA functions, the functionality is not consistent across libraries so comparing these features would have been problematic. Put and Get were tested using a range of data sizes, from one to one million integers, using two processors. The performance was benchmarked both within an LPAR, and across LPARs, allowing both the *shared-memory*[1] and *message*[2] performance of the libraries to be explored. As HPCx is a cluster system, both methods of communication are important to performance, although as the LPARs are restricted to 8 processors the majority of communication performed by an average program will be using some form of message-passing.

4.1 Put

A simple ping-pong program was used to benchmark Put, with one process putting data into the address space of the second process, and the second process putting the data back when the first operation has completed. The time to complete one full ping-pong is recorded, and averaged over 100 ping-pongs, for each data size. As well as the time to complete the ping-pong, the bandwidth of the put operation is also calculated.

Figure 1 shows the bandwidth for LAPI and MPI-2 for both within, and between, LPAR communications. Figure 2 shows the times for the communications with the same benchmarks.

These results show that for messages up to 100 Kb, LAPI significantly outperforms MPI-2. In fact, for messages below 8-10 Kb, LAPI can be as much as 100 times quicker than MPI-2. However, this figure only applies to a `LAPI_Put` performed within an LPAR. A `LAPI_Put` operation between LPARs (i.e. using the switch network) is approximately 5 times slower than one within an LPAR (i.e. using shared memory). MPI-2, on the other hand, exhibits no discernible performance difference between

[1] on-LPAR
[2] off-LPAR

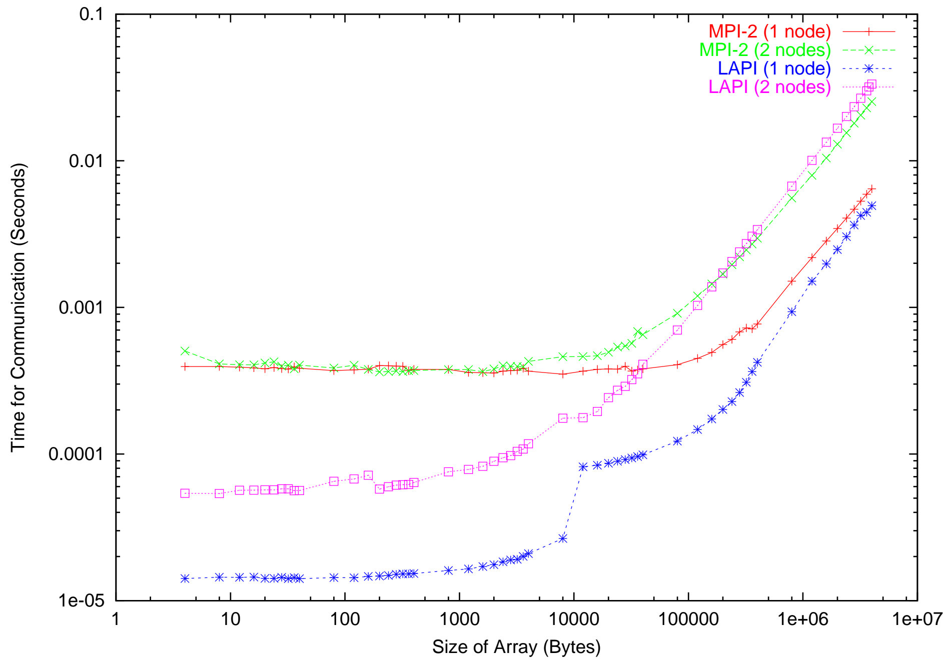

Figure 2: Put Time

the intra- and inter-LPAR Put operations until the message size reaches 2Kb (i.e. for small messages). After this point, the intra-LPAR performance is significantly better than the inter-LPAR performance.

For large messages (>100Kb) the performance of MPI-2 converges with LAPI, although there is still a large difference between on-LPAR and off-LPAR performance. In fact, for very large messages MPI-2 has slightly better performance than LAPI, although the difference is small. LAPI experiences a reduction of on-LPAR performance at $\sim$ 10Kb. This effect is likely to be due to the size of the shared memory buffer that is used by LAPI to perform the on-LPAR data transfer.

The peak bandwidth of LAPI is 2000 MB/s (on-LPAR), and 230 MB/s (off-LPAR). This compares with 1200 MB/s (on-LPAR), and 300 MB/s (off-LPAR) for MPI-2. The peak performance of on-LPAR communication for HPCx is approximately 2000 MB/s, with a corresponding off-LPAR performance of 300 MB/s. Therefore, LAPI exhibits near-peak bandwidth for on-node Puts, but only 75% of maximum off-LPAR bandwidth. On the other hand, MPI-2 only manages 60% of peak bandwidth on-LPAR, but uses near-peak bandwidth off-LPAR. However, whilst MPI-2 reaches a higher peak bandwidth than LAPI (off-LPAR), this is only true for very large data transfers (i.e. $>$ 200Kb). Whilst 200Kb does not seem like a large amount of data, the average data transfer for parallel programs is a lot smaller than this.

The latency off LAPI is $1.4e^{-05}$ seconds (on-LPAR), and $5.4e^{-05}$ seconds (off-LPAR). MPI-2 has latencies of $3.9e^{-04}$ seconds (on-LPAR), and $4.0e^{-04}$ seconds (off-LPAR). This suggests that LAPI is optimised to make the best use of shared-memory communications on IBM systems, such as HPCx.

4.2 Get

MPI-2 Get was tested with the same ping-pong method used for the Put benchmarks. However, due the completion methods used by LAPI the same test could not be used for LAPI Get. Therefore, the `LAPI_Get` function was tested using a one-sided benchmark (i.e. the origin process getting data from the target process). Aside from this difference, the Get benchmarking tests where conducted following the same procedure used for the Put benchmarks.

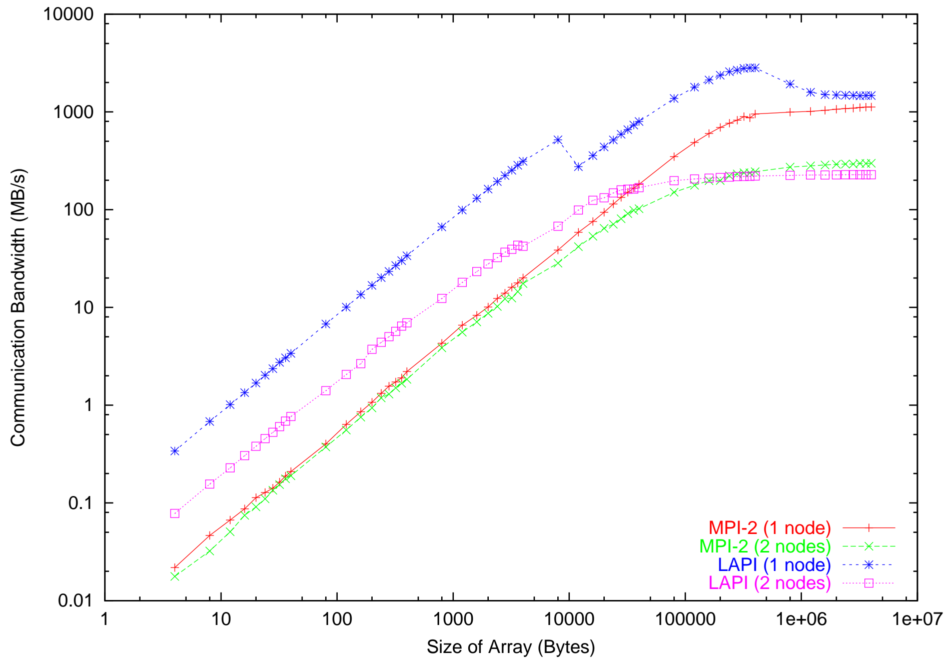

Figure 3: Get Bandwidth

Figures 3, and 4, show the bandwidth and communication times for LAPI and MPI-2 Gets. The bandwidth and time graphs are almost identical to the Put ones.

However, the latency times do differ, with `LAPI_Get` having a latency of $2.24e^{-05}$ seconds on-LPAR and $9.79e^{-05}$ seconds off-LPAR. This is twice the latency of `LAPI_Put`. However, the latency of MPI-2 Get is the same as the latency of MPI-2 Put.

5 Summary

This benchmarks performed show that LAPI has a much lower latency, both on- and off-LPAR, for all but the largest messages, when compared with MPI-2 single sided functions. However, MPI-2 does have a better peak bandwidth for off-LPAR communications, and as each LPAR only contains 8 processors, this constitutes the majority of communication on HPCx. Therefore, if a program needs to perform single sided communications using large amounts of data (i.e. larger than 100Kb), the MPI-2 functions will give slightly better performance. However, if the data being transfered is smaller than 100 Kb, LAPI functions give significantly better performance.

The results also showed that if LAPI is being used to perform single sided communications, `LAPI_Put` should be the preferred operation, as it has approximately half the latency of `LAPI_Get`

6 Future Work

All the communication libraries have extended functionality which could improve single sided performance. It would be interesting to benchmark the performance of these additional functions. It would also be interesting to extend benchmarks performed for this report to cover more than two processes. This would allow for a more detailed profiling of the differences in the communication characteristics of the various libraries.

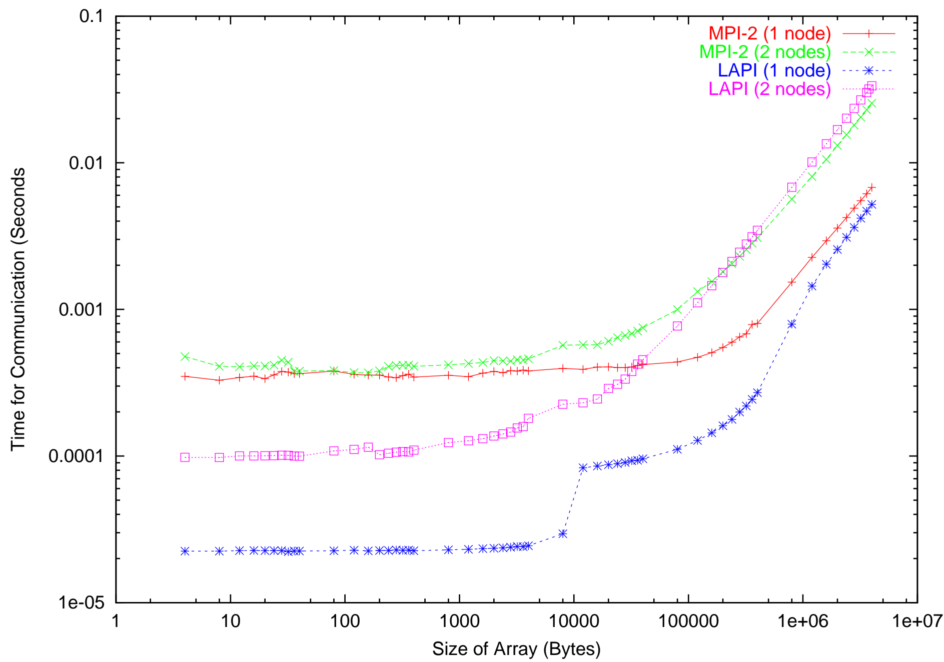

Figure 4: Get Time

References

[1] **MPI-2: Extensions to the Message-Passing Interface**, Message Passing Interface Forum, July 18th, 1997

[2] **Single sided MPI implementations for SUN MPI**, S.Booth, E.Mour ao, EPCC, August 4th, 2000, SuperComputing 2000

[3] Part 7 (Chapters 29, 30, and 31) of **Parallel System Support Programs for AIX: Administration Guide**, Version 3 Release 5, IBM Document Number: SA22-7348-05

[4] **Performance and Experience with LAPI - a New High-Performance Communication Library for the IBM RS/6000 SP**, Gautam Shah; Jarek Nieplocha; Jamshed Mirza; Chulho Kim; Robert Harrison; Rama K. Govindaraju; Kevin Gildea; Paul DiNicola; Carl Bender; Proceedings of IPP'98

Single sided MPI implementations for SUN MPI, S.Booth, E.Mour ao, EPCC, August 4th, 2000, SuperComputing 2000