SAND2006-3183C

Measuring MPI Send and Receive Overhead and Application Availability in High Performance Network Interfaces

Douglas Doerfler and Ron Brightwell

Center for Computation, Computers, Information and Math
Sandia National Laboratories[1]
Albuquerque, NM 87185-0817
{dwdoerf, rbbrigh}@sandia.gov

Abstract. In evaluating new high-speed network interfaces, the usual metrics of latency and bandwidth are commonly measured and reported. There are numerous other message passing characteristics that can have a dramatic effect on application performance, and they too should be analyzed when evaluating an new interconnect. One such metric is overhead, which dictates the networks ability to allow the application to perform non-message passing work while a transfer is taking place. A method for measuring overhead, and hence calculating application availability is presented. Results for several next generation network interfaces is also presented.

Keywords: MPI, High Performance Computing, High Speed Networks, Message Passing Overhead.

Introduction

Scaling efficiency of parallel applications in many instances depends on the ability to overlap communication with computation. In MPI codes, the pre-posted send and receive calls are the primary means of achieving overlap. Unlike other MPI communication metrics, e.g. latency and bandwidth, there is a lack of readily available open source microbenchmarks that measure MPI overhead for non-blocking calls. A method for measuring overhead and application availability is presented and then applied to several current state-of-the-art high performance network interfaces.

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04- 94AL85000.

Method

There are multiple methods an application can use to overlap computation and communication using MPI. The method assumed by this paper is to use the MPI non-blocking MPI_Isend() and MPI_Irecv() calls to post the respective transfer, perform some work, and then wait for the transfer to complete using MPI_Wait(). This method is typical of most applications and hence makes for the most realistic measure of a microbenchmark. Periodic polling methods have also been analyzed [1], but that method only makes sense if the application knows that MPI progress will not be made without periodic MPI calls during the transfer. Overhead is defined to be [2]:

> *… the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations.*

Application availability is defined to be the fraction of total transfer time[2] that the application is free to perform non-MPI related work.

$$\text{Application Availability} = 1 - (\text{overhead} / \text{transfer time}) \quad (\mathbf{1})$$

Figure 1 illustrates the method used for determining overhead and message transfer time. For a given iteration of the post-work-wait loop, the total amount of time for message completion is measured. For each iteration, the amount of work is increased. Initially, the work performed by the application does not effect the total iteration time as it can be completed before the message transfer completes. As the work increases, at some point the interval time starts to increase because the work interval is greater than the transfer time. After this point, the iteration time becomes the amount of time required to perform the work plus the overhead time required by the host processor to complete the transfer. The overhead can then be calculated by measuring the amount of time to perform the same amount of work without a message transfer and subtracting this value from the iteration time.

As mentioned above, the iteration time measurements before the work time becomes a factor is the message transfer time. For this analysis, in order to get a more accurate transfer time value, the measurement values are averaged over the samples taken before the iteration time begins to increase due to work. In the figure, a threshold is shown which when exceeded, the iteration time values are no longer used in the average calculation. This threshold must be set sufficiently high to avoid a pre-mature stop in the accumlation of the values used for the average calculation, but it must be low enough such that values measured after the work becomes a factor are not used. A typical value that worked well for most systems is 1.02 to 1.05 times the transfer time.

The figure also shows an iteration stop threshold. This threshold is not critical and can be of any value as long as it is ensured that the total iteration time is significantly larger than the transfer time. A typical value is 1.5 to 2 times the transfer time. In practice, it is not necessary to calculate the work interval without messaging until the

[2] Per the MPI non-blocking call definitions, the MPI_Wait() call only signifies that for a send the buffer can be reused and for a receive the data can be accessed in the receive buffer [3].

final sample has been taken. Figure 2 illustrates the actual measured values for each iteration at a message size of 64K bytes on the Myrinet 10G test cluster Odin.

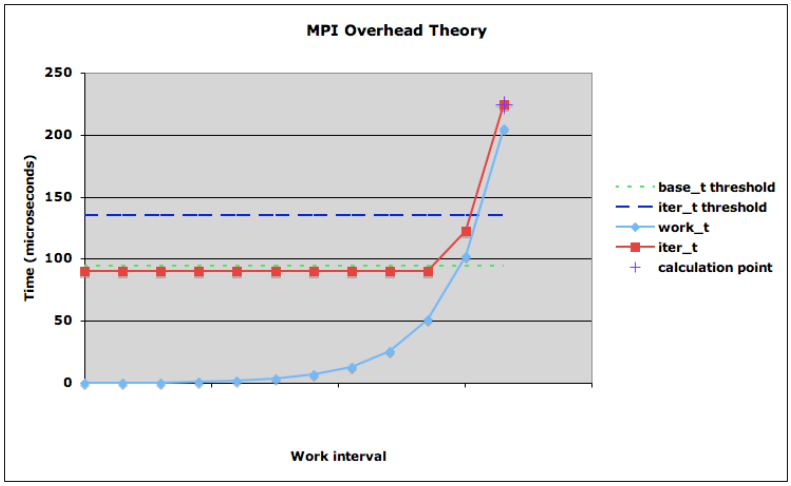

Fig. 1. A conceptual illustration of the message transfer time (iter_t) of a given message size for each iteration of the algorithm, with the work performed (work_t) increasing for each iteration. The message transfer time average calculation threshold (base_t) and iteration stop threshold (iter_t) are also shown along with the point at which the overhead calculation is taken.

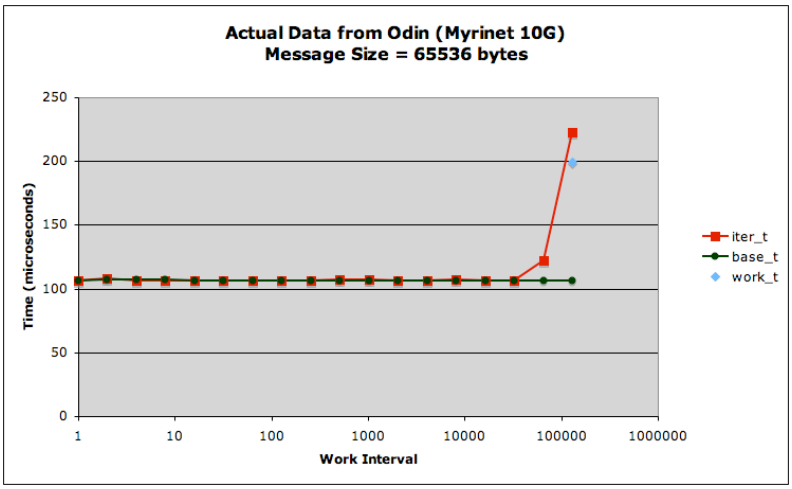

Fig. 2. Actual measured values for a 65K bytes MPI_Isend() operation on the Myrinet 10G cluster. Note that the base_t calculation is not a running average, but an equally weighted average of all values obtained up to that iteration. The work without messaging value is only calculated after the iteration threshold has been exceeded. Overhead is the difference between iter_t and work_t at that point.

Platforms

Overhead and availability was measured on a variety of platforms, summarized in Table 1.

Table 1. Overview of Test Platforms

	Red Storm	*Thunderbird*	*CBC-B*	*Odin*	*Red Squall*
Interconnect	Seastar 1.2	InfiniBand	InfiniBand	Myrinet 10G	QsNetII
Manufacturer	Cray	Cisco/Topspin	PathScale	Myricom	Quadrics
Adaptor	Custom	PCI-Express HCA	InfiniPath	Myri-10G	Elan4
Host Interface	HT 1.0	PCI-Express	HT 1.0	PCI-Express	PCI-X
Programmable coprocessor	Yes	No	No	Yes	Yes
MPI	MPICH-1	MVAPICH	InfiniPath	MPICH-MX	MPICH QsNet

All of the platforms except Red Storm are Linux Clusters using the respective vendors commercial software stacks. The Thunderbird Cluster's MPI software stack has been modified and parameters have been set to reduce the memory required by the MPI stack at a scale of several hundred to a thousand processes. These modifications do affect the real world application performance, but it is unknown how those modifications affect the MPI overhead microbenchmark used in this analysis. The Red Storm platform uses the Catamount light-weight kernel [4]. The Seastar interconnect implements the Portals API [5]. All of the platforms use MPICH 1.x for their MPI software stack, although it is fair to say that this stack may have been optimized for their respective network interface, in particular the collective routines.

Results

From a practical perspective, application availability is usually not a concern for small message sizes, as there is little to be gained trying to overlap computation with communication when transfer times are relatively small. Most applications will only try to overlap computation when they know the message size is sufficiently large. However, as an academic exercise, it still may be interesting to view availability for a large message as it provides information on how an interface's characteristics change at a protocol boundary, such as the switch from a short message protocol to a large message protocol. If an application writer is trying to optimize to a given platform, he/she may want to know where the protocol boundaries are and modify the code to better suit the platform.

Figure 3 charts the MPI_Isend() overhead as a function of message size for the platforms tested[3]. Figure 4 charts application availability. The first thing to note is that the overhead for the Red Storm (Seastar) and Red Squall (Elan4) interconnects is relatively constant for all message sizes. As such the application availability increases with message size until it is nearly 100% for large message transfers. The other interconnects are unable to make MPI progress for large messages without the host CPU becoming involved in the transfer. This can be seen in the availability chart for the Thunderbird (InfiniBand) and CBC interconnects (InfiniPath), once the host becomes involved in the transfer host application availability drops significantly. For the Odin cluster (Myri-10G), availability remains high at approximately 80% to 90%, even though overhead rises with message size for large transfers. However, the overhead is a small fraction of the overall transfer time and hence availability remains high.

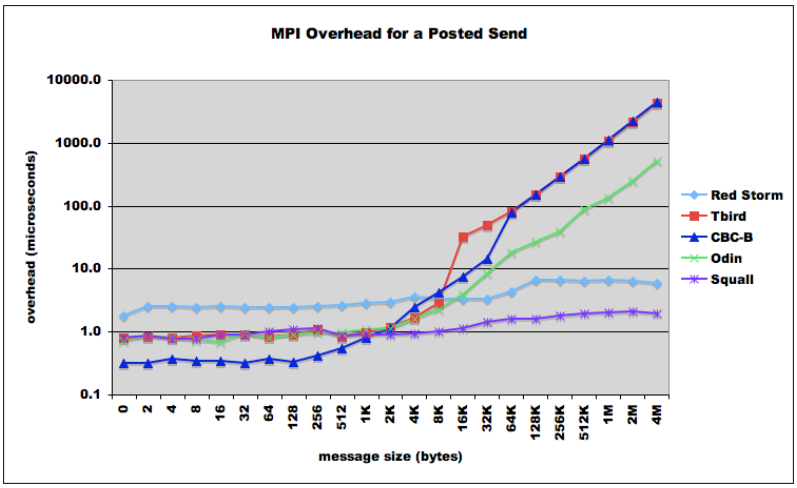

Fig. 3. Overhead as a function of message size for MPI_Isend().

[3] Note that this figure uses a logarithmic axis for overhead. Although the Odin overhead does increase with message size for large messages, it is still an order a magnitude less than the InfiniBand based interconnects.

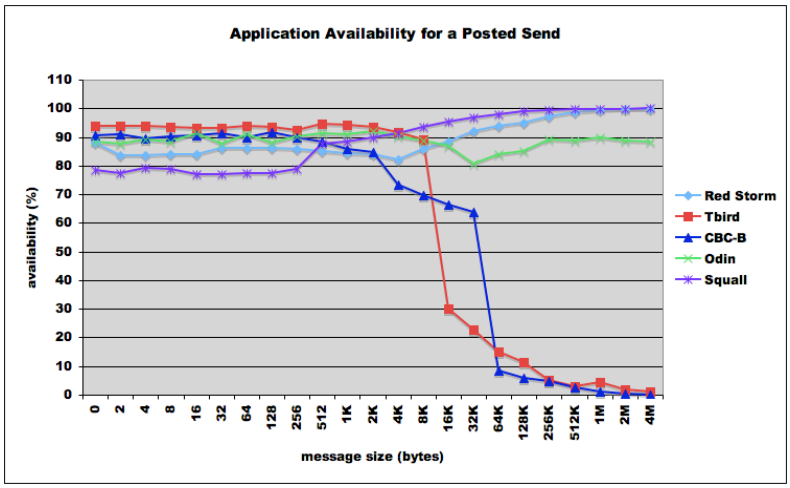

Fig. 4. Application availability as a function of message size for MPI_Isend().

Posted receive performance is charted in Figures 5 and 6. As with posted sends, the Red Storm and Red Squall (Elan4) platforms have essentially a constant overhead independent of message size. This translates to near 100% host availability for large message transfers. The Thunderbird (InfiniBand), CBC-B (InfiniPath), and Odin (Myri-10G) clusters require significant host involvement for large message transfers, and thus limit an applications ability to overlap computation and communication.

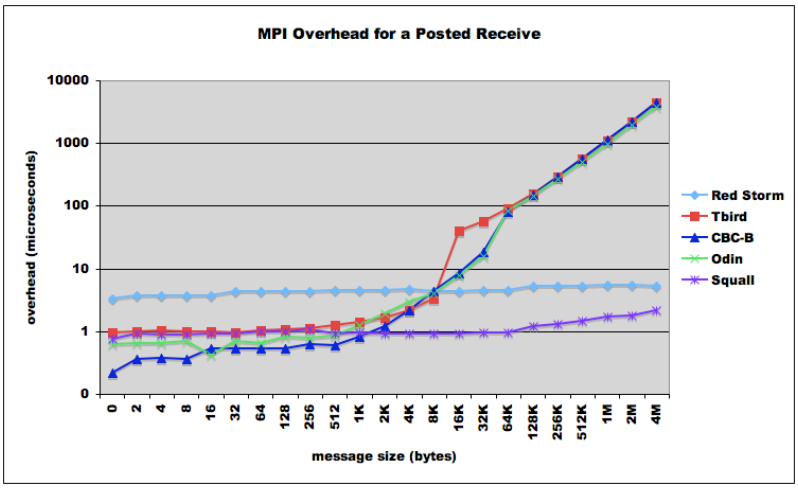

Fig. 5. Overhead as a function of message size for MPI_Irecv().

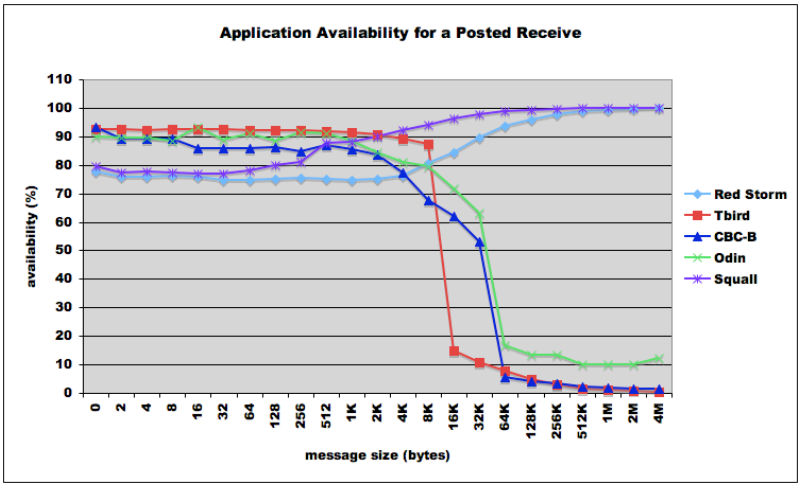

Fig. 6. Application availability as a function of message size for MPI_Irecv().

Related Work

Numerous work has been done to measure overhead and study its effect on application performance [1], [6], [7] and [8]. Lawry [1] looks at application availability, but the analysis and results are for a given message size and charts are a function of the polling interval. Other references do not quantify the overhead as a function of message size, but look at it's effect on application performance. In addition, this paper presents overhead results for some realatively new networking technologies, such as Red Storm's Seastar, Pathscale's InfiniPath and Myricom's Myri-10G.

Conclusion

Even though a network has impressive latency and bandwidth numbers, if the application can take advantage of communication and computational overlap, one network may provide significant performance advantages over another. As such, when analyzing a network interface, one needs to look at more than simple ping-pong and bandwidth metrics. Some of the more recent, and seemingly very popular, high-speed networking solutions, such as InfiniBand, do not provide overlap. Which in some ways is surprising given the fact that the importance of communication offload and its effects on application performance have been common knowledge for several years. However, for many applications it is sufficient to provide low latency and high bandwidth to achieve excellent performance and scaling.

Future Work

It is the intent of the authors to make the source to the code used in this study generally available and downloadable from an open web site. With the hope that this will allow overhead and application availability to become a common microbenchmark in the evaluation of next generation interconnects. This will also encourage contributions to make the code more robust and accurate in its reports.

References

1. W. Lawry, C. Wilson, A. Maccabe, R. Brightwell. COMB: A Portable Benchmark Suite for Assessing MPI Overlap. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 2002)*, p. 472, 2002.
2. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN symposium on Principles and Practice of Parllel Programming,* pp. 262-273, 1993.
3. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, J. Dongara. MPI: The Complete Reference. p. 52, The MIT Press, Cambridge, Massachusetts, 1996.
4. S. Kelly, R. Brightwell. Software Architecture of the Light Weight Kernel, Catamount. In *Proceeding of the 47th Cray User Group (CUG 2005)*, 2005.
5. *Portals API*, http://www.cs.sandia.gov/Portals.
6. R. Martin, A. M. Vahdat, D. E. Culler, T. E. Anderson. The Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the International Symposium on Computer Architecture,* 1997.
7. D. Culler, L. T. Liu, R. P. Martin, C. O. Yoshikawa. Assessing Fast Network Interfaces. *IEEE Micro*, pp. 35-43, Feb., 1996.
8. C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, K. Yelick. An Evaluation of Current High-Performance Networks. In *Proceedings IEEE International Parallel & Distributed Processing Symposium (IPDPS '03),* 2003.
9. R. Brightwell, D. Doerfler, K. D. Underwood. A Preliminary Analysis of the InfiniPath and XD1 Interfaces. In *Proceedings IEEE International Parallel & Distributed Processing Symposium (IPDPS '06),* 2006.