Performance Modeling and Evaluation of MPI

Khalid Al-Tawil * Csaba Andras Moritz †

Abstract

Users of parallel machines need to have a good grasp for how different communication patterns and styles affect the performance of message-passing applications. LogGP is a simple performance model that reflects the most important parameters required to estimate the communication performance of parallel computers. The message passing interface (MPI) standard provides new opportunities for developing high performance parallel and distributed applications. In this paper, we use LogGP as a conceptual framework for evaluating the performance of MPI communications on three platforms: Cray-Research T3D, Convex Exemplar 1600SP, and a network of workstations (NOW).

Our objective is to identify a performance model suitable for MPI performance characterization and to compare the performance of MPI communications on several platforms.

Keywords: LogP, MPI, LogGP, Parallel Processing, Workstations.

1 Introduction

Recent advances in computer and semiconductor technologies make parallel systems widely applicable for solving real problems. The complexity of designing efficient parallel applications and algorithms requires that models be used at various levels of abstraction.

Several approaches to model the communication performance of a multicomputer have been proposed in the literature [1, 3, 10]. LogP is a simple parallel machine model that reflects the most important parameters required to estimate the real performance of parallel computers [10].LogGP [3] is an extension of LogP capturing the increased network bandwidth for long messages.

The message passing interface (MPI) standard [16, 17] provides a flexible environment for developing high performance parallel applications. MPI is a very flexible communication layer providing several mechanisms for point-to-point and collective communications. It provides an efficient standard to implement message-passing applications on different platforms.

In MPI, it is often possible to express the same application communication requirement in many different ways, using different combination of MPI primitives, with different type of resource demands. The performance implications are complex and not easy to understand. Information about resource usage, communication performance, and latency hiding opportunities are required to help MPI programmers select appropriate communication mechanisms.

This paper presents several approaches to model the performance of MPI point-to-point communications. First, we use the LogGP model directly on the MPI primitives, without explicitly modeling the underlying protocol messages. The cost of protocol messages is often small compared to the cost of user messages, thus such simplification gives acceptable prediction in most of the cases. Furthermore, a simple model is also advantageous because it is platform independent, and can be used in performance comparisons between different platforms. In addition, exact information about the underlying messaging protocols is often not available for MPI programmers. We describe a set of communication benchmarks and use them to extract the LogGP parameters on three platforms: Cray Research T3D, Convex Exemplar 1600SP, and workstation clusters.

Next, we present a methodology for how to model the underlying protocols used, by decomposing the MPI primitives into low-level non-blocking communication primitives. We show how to derive the communication performance for several typical MPI communication protocols. Incorporating details about the implementation may be necessary for MPI communication primitive designers.

Finally, we describe how to model the performance of MPI programs running on machines with two tiered organization such as the Convex Exemplar 1600SP. This model exposes the performance gap between inter-cluster and intra-cluster communications in cluster of shared memory type of machines.

The remainder of this paper is organized as follows. In Section 2, the basic concepts and communication styles of MPI, the LogGP model parameters, and the hardware platforms are introduced. In Section 3, we describe the MPI specific model extensions to LogGP and the extension for the two tiered architectures. In Section 4, we present MPI communication performance comparisons for all the platforms. Section 5 concludes the paper.

2 Background

Message Passing Interface

The MPI standard was defined in a forum involving the active participation of more than 40 different vendors and organizations. A very important goal of MPI is to provide a widely portable and efficient programming library without sacrificing performance. Since the release of the standards [16], several MPI implementations have become publicly available: CHIMP from Edinburgh Parallel Computing Center (EPCC) [2], University of Edinburgh; LAM from Ohio Supercomputer Center [7]; MPICH from Argonne National Laboratory [13]; and Unify from Mississippi State University [12]. All of these implementations have a similar performance if compared on the same computing platform [18].

In the message-passing programming model, a program consists of a set of processes, where each process performs an independent computation. These processes communicate via a number of communication channels. Point-to-point communication is the fundamental communication mode of any message passing multicomputer. This section gives a short introduction to the different point-to-point communication primitives in MPI. A summary of the different communication styles is shown in 1. More information about MPI communication styles can be found in [18].

MPI provides different send and receive types, with different synchronization and resource usage semantics. The type of a send operation can be blocking or non-blocking with the following modes: buffered, standard, ready, and synchronous. Blocking in MPI typ-

*Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran – 31261, Saudi Arabia, e-mail: khalid@ccse.kfupm.edu.sa

†Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, e-mail:andras@lcs.mit.edu

To appear in Journal of Parallel and Distributed Computing.

ically means blocking the current process until resources used for the operation can be reutilized. A send operation is considered local if the operation completes before a communication with a remote process is initiated.

The behavior of the standard mode is not precisely defined in the MPI standard. A popular interpretation is that the send completion be independent of the receiving process as long as there is sufficient system buffering space; when this space is finished, send completion can be delayed for a matching receive.

The buffered mode buffers the outgoing message when no matching receive is posted. The operation is local even for the blocking call. The buffer cannot be reutilized directly after the non-blocking sends. A difference between the explicit buffered mode and the standard mode (in a buffered implementation) is that in the case of the buffered mode additional buffer space can be declared via a special MPI function call. In a blocking buffered mode the sender can essentially continue with the next operation after the message is in the buffer.

The synchronous mode requires some handshaking between the sender and the receiver, thus this operation is non-local. Completion of a blocking synchronous send means only that the receiver has reached a certain point in its execution, and it is not a guarantee for a completed communication.

The ready mode send is completed only when the matching receive is already posted. The behavior of this mode can be undefined if a send operation precedes a matching receive. In the case of the Cray T3D this mode is for example equivalent with the standard mode. The rational behind the non-blocking modes is that completion of the operation can be checked later, thus overlapping of communication with computation is possible. The implementation of the same communication operation may vary in function of message length and other factors, such as availability of resources [9].

Different protocols could be used to implement the send operation, and Figure 1 illustrates some of the typical protocols used in MPI implementations [9]. The T (Transfer) and TA (Transfer Acknowledge) protocols can be used when message header and data can be accommodated within the protocol message. The RAT (Request Acknowledge Transfer), and RTA (Request Transfer Acknowledge) protocols are used for implementing the standard and the synchronous modes when message data cannot be accommodated within the protocol message.

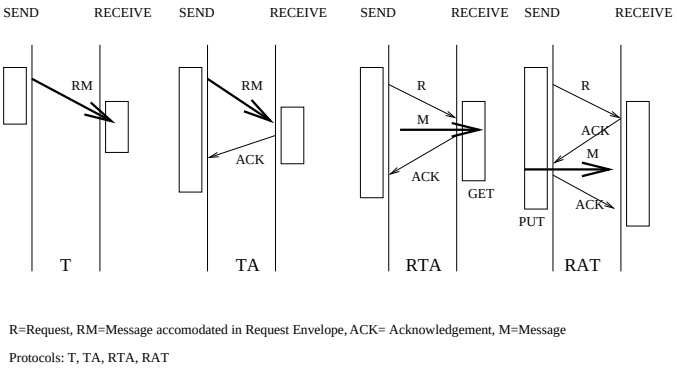

Figure 1: Different protocols used for implementing MPI sends.

LogGP Performance Metrics

The LogP [10] model addresses the performance of a parallel machine in terms of four parameters, as follows: (1) L = latency or the upper bound on the time to transmit a message from its source to destination, (2) o =overhead or the time period during which the processor is busy sending or receiving a message, (3) g = gap or the minimum time interval between consecutive sends and receives, (4) P = the number of processors. The model also assumes a network with a finite capacity, e.g. if a processor attempts to send a message that would exceed the capacity of the network, the processor stalls until the message can be sent. The model is asynchronous, i.e., processors work asynchronously and the latency experienced by any message is unpredictable, but limited by the upper bound L parameter in the absence of stalls. The maximum number of messages in transmit from or to any processor is determined by L/g. Although the model primarily reflects the performance of message-passing systems, the authors claim its applicability for shared memory models based on distributed memory machines.

An extension of LogP for large messages is presented in [3], and a new parameter G is introduced: the *Gap per byte* or the time per byte for long messages. The reciprocal of G characterizes the available per processor communication bandwidth for long messages.

LoGPC [5] is a new model where application specific parameters are introduced to account for network and resource contention effects. LogP is quantified for low-overhead local area networks in [14]. The performance assessment of LogP for fast network interfaces is presented in [11]. The majority of performance measurements for point-to-point MPI communications are based on measuring the round-trip time or the average transfer time between two processes. Benchmarks like the COMMS1 and COMMS2 in the GENESIS suite [9] have been adapted for MPI performance evaluation. The average transfer time is estimated by halving the average round-trip time. In addition, end-to-end bandwidth is estimated by dividing the message size by the total message latency.

The main difference in presenting the communication performance in the context of LogGP is the separation of the software overheads from the other aspects of the communication performance.

Hardware Platforms

We compare the MPI communication performance of three platforms: Cray-research T3D, Convex Exemplar 1600SP, and a cluster of workstations. On the first two of these platforms, we use the same CHIMP MPI implementation, and on the Convex Exemplar we use MPICH.

The Cray T3D is a distributed memory multicomputer based on a high performance three dimensional torus topology. It is based on nodes with two independent processing elements, consisting of a DEC Alpha 21064 processor with a frequency of 150 MHz and 64MB of RAM. The memory interface between the Alpha processor and the local memory involves Cray customized circuitry which extends the local virtual address space to a global address space. Each processor can directly read and write to any other processor memory through the shared memory access library [8]. Cache coherence can be handled at the user's discretion, the interconnect interface hardware allows remote data entering a processor's local memory to invalidate the corresponding cache line. The cost of routing data between nodes is essentially negligible, two cycles per node traversed and one extra clock cycle to turn the corner. The single hop latency is 1-2 microseconds and the bandwidth is 120MB/s for the remote memory put operation and 60MB/s for the remote memory get operation.

The Convex Exemplar 1600SP used in this paper is a shared memory MP system based on HP PA-RISC chip configured in up to 16 hypernodes, each having up to 8 processors, an I/O port, and up to 2 gigabytes of physical memory. Hypernodes are interconnected with 4 rings. The MPI layer used is Convex MPICH V1.0.12.1 and it is compatible with Argonne National Laboratory's MPICH V1.0.12. It can be programmed as a conventional shared memory machine, or as a distributed memory message-passing machine, or as a hybrid of both. The system we used is from the Swiss Center for Scientific Computing and has two hypernodes, with a total of 16 HP-PA 7200 CPUs.

Operation	Main type	Mode	Ends	Buffer.	Other
MPI_Send	Blocking	Standard	?	?	Dependent on implementation
MPI_Ssend	Blocking	Synchronous	Non-local	No	
MPI_Rsend	Blocking	Ready	Local	No	Matching receive must precede
MPI_Bsend	Blocking	Buffered	Local	Yes	May need MPI_Buffer_attach
MPI_Isend	Non-block	Standard	Local	?	Need MPI_Test or MPI_Probe
MPI_Issend	Non-block	Synchronous	Local	No	Need MPI_Test or MPI_Probe
MPI_Irsend	Non-block	Ready	Local	No	Need MPI_Test or MPI_Probe
MPI_Ibsend	Non-block	Buffered	Local	Yes	Need MPI_Test or MPI_Probe

Table 1: Summary of MPI point-to-point send operations

The network of workstations used is mainly a cluster of SPARC stations. These are 8 workstations of Sun 4/50 SPARC station IPX with 16 MB RAM running SunOS 4.1.3. The workstations are connected by an ordinary 10Mbit/s Ethernet. Table 2 shows the processor, memory, and the communication interface of the three platforms.

3 Modeling MPI Performance

In this section we introduce different methods to model MPI communication performance. First, we use the LogGP model directly on MPI primitives, ignoring the underlying protocols (and thus the extra protocol messages) in MPI. This approximation is advantageous because it allow us to give first order approximation and comparison between platforms without considering machine and implementation specific details. It is also easy to use by MPI developers because the LogGP metrics can be derived in similar ways on all platforms.

An extension of this model captures the underlying protocols used in MPI operations, applying LogGP to model the low-level asynchronous protocol messages instead. Because communicating short messages typically do not require extra protocol messages this model often can be reduced to the simple model mentioned above. Similarly, when using large messages the cost of extra protocol messages can be ignored. The disadvantage of this model is that the information about the exact protocols in MPI implementations is hard to obtain. Incorporating these details makes sense in models used by MPI designers.

Finally, we describe a new performance model for architectures based on cluster of shared memory multiprocessors.

Note, that the performance results given in this paper are mainly using the simple approach for the reasons described above. For the Convex Exemplar we present the results based on the extension designed for cluster of shared memory multiprocessors.

3.1 A Simple Approach

The goal of a performance model is to present an abstract view of the system removing unnecessary details. A model that is primarily designed for MPI application programmers can be different from a model targeted to MPI communication designers.

We believe that the LogGP model can directly model MPI primitives, and that modeling the underlying protocols in MPI is not necessary if the model is targeted for application writers. Because communicating short messages typically do not require extra protocol messages, there is no difference between the two models for short messages. Similarly, when using large messages the cost of extra protocol messages can be ignored compared to the user message cost.

The only requirement for this approach is to distinguish between the send and receive overheads for different communication styles as these overheads are very different.

Experimental Methodology

This section shows the experimental methodology used to extract the LogGP performance parameters for MPI. In our experiments, messages are exchanged between two processors. The receiving processor returns the message back immediately to the sender. Both the sends and the receives are executed after a barrier that first synchronizes the two processes involved. Non-blocking communications were tested with the test for completion (MPI-test). For buffered communication, MPI buffers were declared. Experiments were performed for different message lengths and pairs of processors/workstations. They were repeated at least 600 times on all platforms. The experiments have been repeated with the root processor of the barrier operation changed from the sender to the receiver, to eliminate the performance impact of the barrier operation itself. The effects of instruction caching are eliminated by running a couple of iterations of the the benchmark before measurements are taken. The general structure of the pseudo-code is shown in Figure 2.

The send overhead (O_s):

Measuring the send overhead for local sends (all send types except the synchronous) is done by $timer11$ in the micro-benchmark. The overhead of the synchronous blocking send is dependent on a matching receive operation, thus both the send and receive are issued after a barrier operation.

The receive overhead (O_r):

Measuring the non-blocking receive overhead is done by measuring the time around the receive operations. The results are obtained with $timer21$. In the case of blocking receives we started both the sends and receives after a barrier but we delayed the receive to make sure that the idle time (synchronization) delay is not included in the receive overhead. The results we provide for the receive overhead are determined by the $timer12$ in the micro-benchmark.

The network latency: L

Measuring the exact value of the latency with MPI is rather difficult due to differences in implementations and underlying protocols used. The latency as defined in the LogP model is often hidden in the software send and receive overheads. The latency parameter can be used to estimate the number of MPI operations one can place between the non-blocking sends and the test for completion operations. In the followings we present a methodology to estimate the network latency with MPI.

First, we estimated the time necessary for a receive operation to be aware of a send after a blocking synchronous send was issued on another processor (both started after a barrier). This time can be used as an upper bound for a small message latency. It is calculated by comparing the receive overhead (see $timer12$) with the receive time that also includes the wait time for the first data or protocol message to arrive from the sender (see $timer21$). As the MPI blocking synchronous send is a non-local operation, we assumed that the operation would start with a protocol request or message transfer in all the implementations. In [19] a benchmark implemented with the Alpha assembler is presented for the exact measurement of communication rates on the Cray T3D. Another possibility would be to use the shared memory library [8] for measuring network latency.

Figure 3 shows the software overheads and network latency for a communication pattern using a blocking synchronous send-

Platform	Processor	Memory/node	Network Interface	Other
CrayT3D	Alpha 21064, 150MHz	64MB	Cray custom, 300MB/s	Torus
NOW	Sun4/50SPARC	16MB	Ethernet, 10Mb/s	SUNOS 4.1.3
Convex	HP-PA-RISC7200	64MB	crossbar for intra-hypernode CTI rings for inter-hypernodes	16 CPU in 2 hypern.

Table 2: Comparison of the three platforms

```
Processor1:
 repeat for all types {
   repeat 10 times { // eliminate instruction caching effects
      Barrier
      Send[type] to P2:
      if (type is non-blocking) test_completion
      Barrier
      wait delta
      Receive[type] from P2:
      if (type is non-blocking) test_completion
   }
   repeat n times { // measure
      Barrier
      start timer11
      Send[type] to P2:
      stop timer11   => *send overhead*
      if (type is non-blocking) test_completion
      Barrier
      wait delta   // wait enough for message to arrive.
      start timer12
      Receive[type] from P2:
      stop timer12   => *receive overhead*
      if (type is non-blocking) test_completion
   }

 }

Processor 2:
 repeat for all types {
  repeat 10 times { //eliminate instruction caching effects
     Barrier
     Receive[type] from P1:
     if (type is non-blocking) test_completion
     Barrier
     Send[type] to P1:
     if (type is non-blocking) test_completion
  }
  repeat n times { // measure
     Barrier
     start timer21
     Receive[type] from P1:
     stop timer21 // if (type is blocking synchronous) timer 21
                  // measures the *receive overhead* plus *latency*
                  // for small messages.
     if (type is non-blocking) test_completion
     Barrier
     start timer22
     Send[type] to P1:
     stop timer22
     if (type is non-blocking) test_completion
  }
 }
```

Figure 2: Pseudo-code for point-to-point benchmarking

receive pair implemented with the RTA protocol. The software overheads and latency with the non-blocking synchronous send but a blocking receive is shown in Figure 4. In the case of the blocking synchronous send, we can observe that the network latency is completely overlapped with the send overhead O_s. The reason for presenting these figures is to illustrate the difference between the blocking and non-blocking sends where communication costs can be overlapped with useful computation. The number of messages generated between the sender and receiver is dependent on the protocol used for the operation.

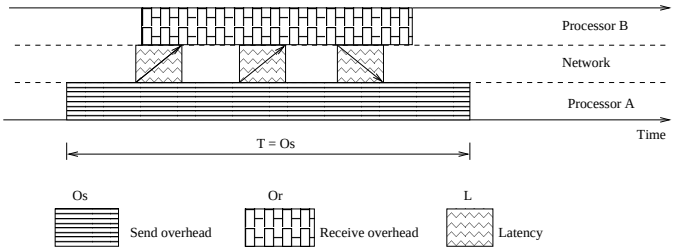

Figure 3: Software overheads and latency with a blocking synchronous send-receive pair implemented with the RTA protocol. Note, that the network latency is part of the blocking send overhead.

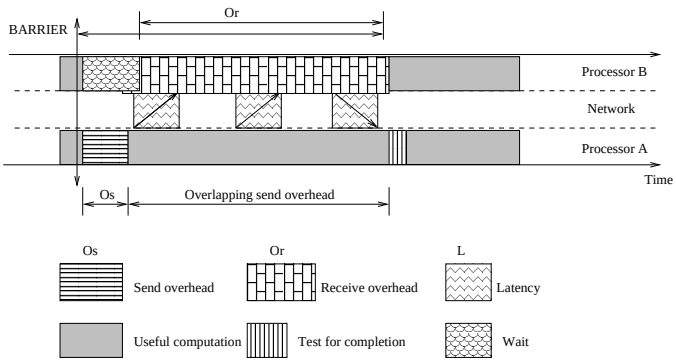

Figure 4: Software overheads and latency with non-blocking synchronous send and blocking receive based on the RTA protocol. This mode could be used for large messages to overlap the message transfer time with useful computation.

The gap: g
The gap is the minimal time interval between consecutive transmissions or receptions at a processor. It basically reflects the processor to network communication bandwidth. For fast, low-overhead communication layers, such as Active Messages [4], the gap is shown to be larger than the sum of the send and receive overheads. MPI overheads for short messages are however so high, that messages could be injected directly after they are constructed. Thus, the minimum distance between consecutive sends can be approximated with a send overhead.
Gap per byte (G):
The G parameter can be estimated by measuring the time around a blocking receive overhead, extracting the non-blocking receive overhead.

An overview of the LogGP performance signature on Cray T3D and NOW for the non-blocking standard and synchronous modes is shown in table 3. The performance signature of the Convex Exemplar 1600SP is presented in section 3.3. A more detailed performance comparison for various message lengths and blocking communications is presented in section 4.

MPI	L	o_s	o_r	g	G
Cray T3D	3.1	30-130(24-100)	15 - 80	-	0.04
NOW	124	1000-1100(1000-1100)	1000	-	0.2

Table 3: LogGP signature for the standard non-blocking modes and synchronous non-blocking modes on Cray T3D and NOW (in μs). The numbers in the parenthesis are for the non-blocking synchronous case. The overheads shown regard 16 and 10000 byte messages. The gap parameter G corresponds to peak communication bandwidths.

3.2 MPI Performance by Decomposition

More details about implementations of MPI can be captured by modeling the underlying low-level operations. We express the performance of MPI by decomposing the MPI operations into basic non-blocking messaging operations similar to Active Messages [4]. These short messages typically reflect the size of network input and output queues. Additionally, extra protocol messages are used to implement flow-control between the sender and the receiver. The objective of this section is only to show the methodology of how to model these low level details.

We denote with $T_s^{protocol}$ the time after which the sender can continue with the execution of its next operation and with $T_{s-r}^{protocol}$ the end to end message delivery time. If the two times are equal then we only show $T_{s-r}^{protocol}$.

The T protocol is typically used for asynchronous short messages and thus the message delivery time is given in a similar way as in LogP.

$$T_s^T = o_s \tag{1}$$

$$T_{s-r}^T = o_s + L + o_r \tag{2}$$

The TA protocol is equivalent to a synchronous short message transfer and it can be decomposed into two asynchronous messages, each having a send and receive overhead component, and a latency component.

$$T_{s-r}^{TA} = 2o_s + 2L + 2o_r \tag{3}$$

The RTA protocol can be decomposed into two short protocol messages, and one large message with size k bytes. As the RTA is using a pulling model (we assume hardware support similar to the one in Cray T3D) for the large message it has no send overhead component. The two send and receive overhead components are included because of the R and ACK messages. The third latency (L) component is accounting for the time required for the first byte of the M message to traverse the network. The rest of the $(k-1)$ bytes are pipelined and thus take $(k-1)G$ time.

$$T_{s-r}^{RTA} = 2o_s + 2L + o_r + (k-1)G \tag{4}$$

$$T_s^{RTA} = 2o_s + 3L + 2o_r + (k-1)G \tag{5}$$

The RAT protocol is using a push model for data transfer. It has therefore an extra send overhead component compared to RTA. As the message M is pushed into the destination memory, we assume that no extra software processing is required for receiving it. Note, that we considered that the transfer rates of the pulling and the pushing methods are equal. The model can be extended to capture significantly different transfer rates.

$$T_s^{RAT} = 4o_s + 3L + 2o_r + (k-1)G \tag{6}$$

The end to end delivery time also includes the latency and the receive overhead of ACK.

$$T_{s-r}^{RAT} = 4o_s + 4L + 3o_r + (k-1)G \quad (7)$$

Note, that for large message sizes the cost of communication is dominated by the $(k-1)G$ term, thus the cost of protocol messages could be ignored.

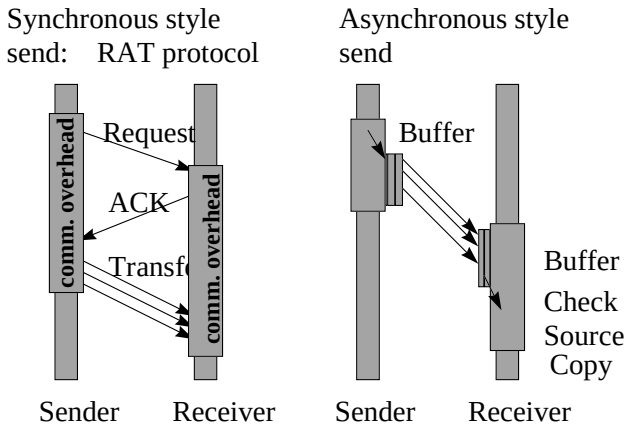

Figure 5: Buffered vs synchronous communications. The RAT protocol shown in the figure is used for example in the implementation of the synchronous MPI send operation on Cray T3D.

3.3 MPI on Clusters of SMPs

LogGP was primarily designed for distributed memory machines with message-passing communication layers.

Clusters of shared memory multiprocessors typically rely on low cost shared memory based communications within hypernodes and message-passing based inter-hypernode communications.

From figures 7 and 6, we observe that there is a factor of ten performance gap between inter- and intra-hypernode message transfers in the Convex Exemplar 1600SP. Similarly, the software overheads for intra-hypernode communication are much less than the inter-hypernode ones because they are based on simple shared memory primitives. This section presents a performance model for two tiered systems called *LoGH*$\{$*sSP*$\}$ to account for these performance gaps.

In the *LoGH*$\{$*sSP*$\}$ model, the choice of parameters reflects the two kinds of inter-processor communication in *TTA*s:

- *intra*-hypernode, based on a messaging mechanism implemented with *shared memory accesses*
- *inter*-hypernodes, based on *messages* and some kind of message passing communication layer

We use the LogP latency model for inter-hypernode communication. We ignore the intra-hypernode latency as it is much smaller than the inter-hypernode latency. We distinguish between the send o_s and receive o_r overheads for inter-hypernode communication, and the the send s_s and receive s_r overheads for intra-hypernode communications. These overheads are differ for various MPI communication styles. Our model for long messages assumes that there is some hardware support available for efficient long message transfer. We use separate parameters to model inter-hypernode transfers

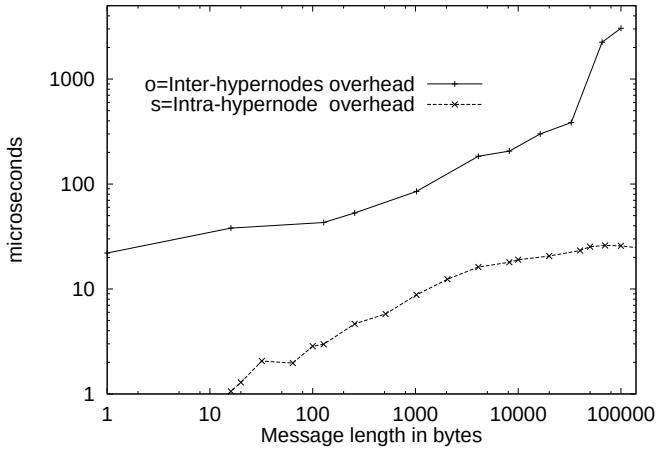

Figure 6: Comparison of inter and intra-hypernode standard non-blocking MPI send overheads on Convex. Variation of overheads in function of message length is shown.

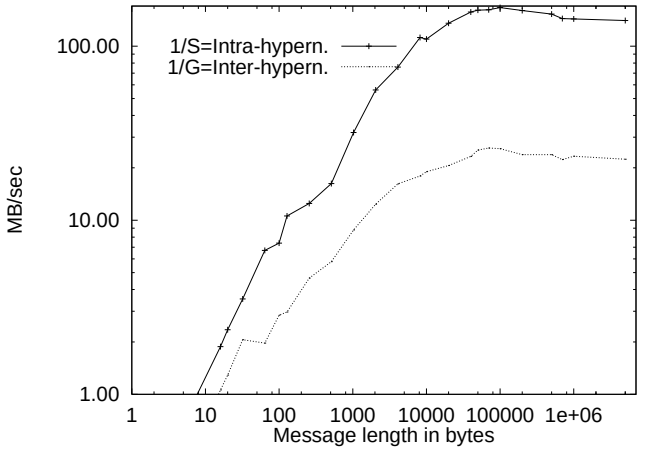

Figure 7: Comparison of $1/G$ inter-hypernode and $1/S$ intra-hypernode bandwidth on Convex Exemplar SPP1600 as function of message length.

L	o	G	H	s	S	P
4-10	30-200	0.05	2	2-20	0.01	4

Table 4: *LoGH*$\{$*sSP*$\}$ signature for the Convex Exemplar with MPI in μs. The overheads shown regard 100 and 10000 byte messages. The gap parameters S and G correspond to peak inter-hypernode and intra-hypernodes communication bandwidths.

Platform	L
Cray T3D	3.1
Convex Exemplar	4-10
NOW	124

Table 5: Network latency L on NOW, Cray T3d and Convex Exemplar (in microseconds).

and intra-hypernode transfers. We model the long message *gap per byte for inter-hypernode* accesses with G. We use the S parameter for the *gap per byte for intra-hypernode* communications.

The end to end delivery time of a large (non-blocking) message of size k bytes (ignoring protocol messages) within a hypernode is given by the following expression:

$$T_{s-r} = s_s + s_r + (k-1)S \tag{8}$$

Note, that we account for software overheads both at the sender and the receiver but assume that the first byte of the message takes no time to reach the destination.

The end to end delivery time of a large message of size k bytes for inter-hypernode communication is:

$$T_{s-r} = o_s + o_r + (k-1)G + L \tag{9}$$

The performance metrics for the Convex Exemplar using MPI are summarized in Table 4.

4 Performance Comparison

The network latency results obtained for the three platforms are presented in Table 5.

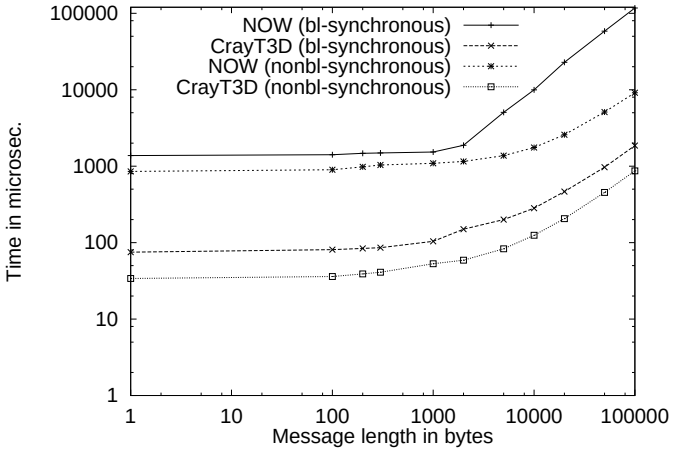

Figure 8: Send overheads in blocking and non-blocking synchronous modes on Cray T3D and NOW. Note, that the blocking synchronous mode is non-local. Variation in function of message size is shown.

Comparison graphs for the software overheads are presented in Figures 8 to 11.

On the Cray T3D, the non-blocking send overheads show small variations in function of message length. This is in line with the assumptions made in the LogP model for considering the asynchronous send and receive overheads as constant. The results obtained for the standard non-blocking send are similar to those obtained with the non-blocking buffered mode.

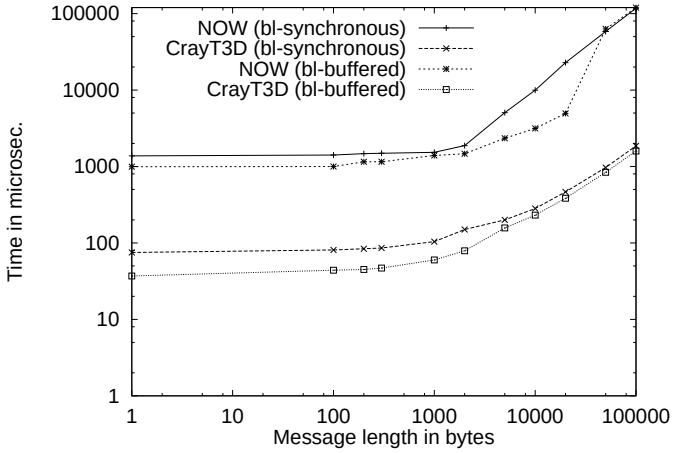

Figure 9: Comparison of blocking buffered and synchronous MPI send overheads. Note, that the blocking buffered mode is local, having a smaller overhead for short messages. However, for larger messages buffering becomes very expensive and the overheads are comparable.

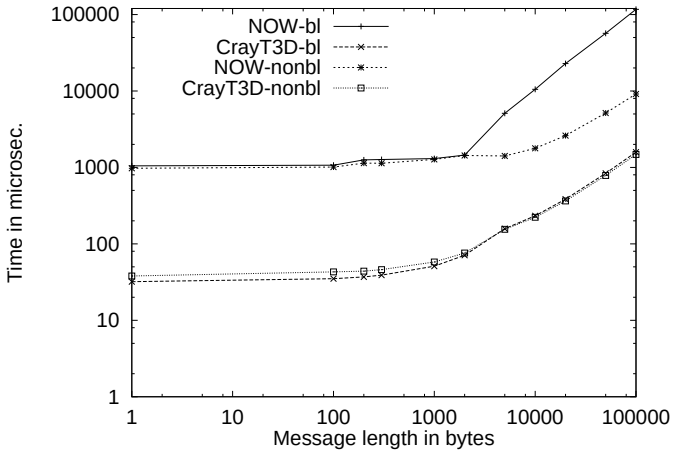

Figure 10: Comparison of blocking and non-blocking standard MPI send overheads. The performance gap between these modes gives an indication about overlapping chances at the sender.

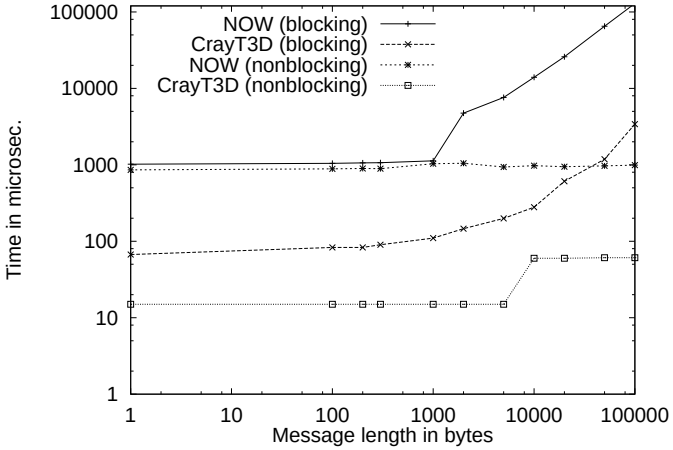

Figure 11: Comparison of blocking and nonblocking receive overheads. The difference between the two modes shows possible communication overlapping for various message lengths if nonblocking mode is used at the receiver.

On the NOW, the blocking synchronous mode for small messages is more expensive than the standard or buffered modes due to larger network latency. As expected, we can see that the synchronous mode is more advantageous in the case of very fast networks such as in the CrayT3D. The non-blocking send overheads on the NOW show small variations, increasing by a factor of ten between 1-100000 bytes. The results obtained show that these overheads are larger by a factor of ten than those obtained for the CrayT3D.

The blocking receive overheads on the NOW suggest good communication/computation overlapping possibilities for messages larger than 1000 bytes. On the CrayT3D, it is advantageous to use nonblocking communication for messages larger than 500-1000 bytes in the case of send operations, and 2000 bytes in the case of receive operations.

Figures 12 and 13 show the comparison of the standard nonblocking send overhead and the network bandwidth for long messages for all the three platforms.

On the Convex Exemplar both the overheads and the long message gaps are significantly larger for the inter-hypernode than the intra-hypernode case. The Convex Exemplar intra-hypernode overheads are the lowest, the inter-hypernode overheads are only slightly larger than the CrayT3D overheads. Messages can be communicated fastest within the nodes of Convex Exemplar hypernodes.

5 Conclusions

We expect MPI to be the high performance communication layer on most massively parallel processors (MPPs) and NOW in the future. In this paper, we used LogGP as a conceptual framework for evaluating the performance of MPI communications on three platforms: Cray-research T3D, Convex Exemplar 1600SP, and a network of workstations. We have discussed how to model the performance of MPI by incorporating more details about the platform and the protocols used. We have developed a simple set of communication benchmarks to extract the performance parameters and presented detailed measurements of the differences in communication performance among the platforms. We found that modeling the performance gap between inter-cluster and intra-cluster message-passing is important. For the Convex Exemplar, the software overheads and

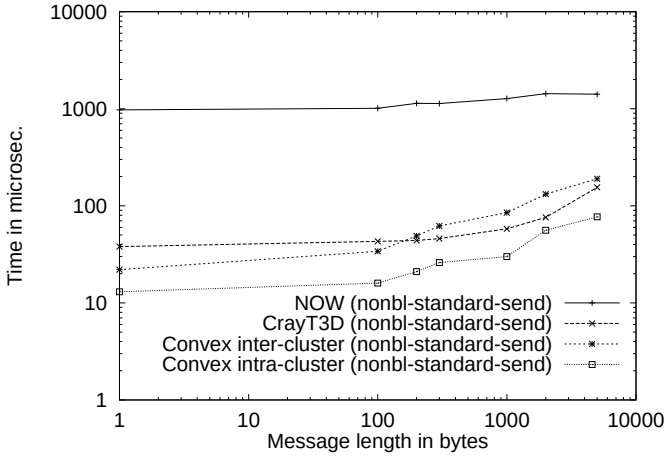

Figure 12: Comparison of standard non-blocking send overheads on Cray T3D, NOW and Convex Exemplar. Variation in function of message length is shown.

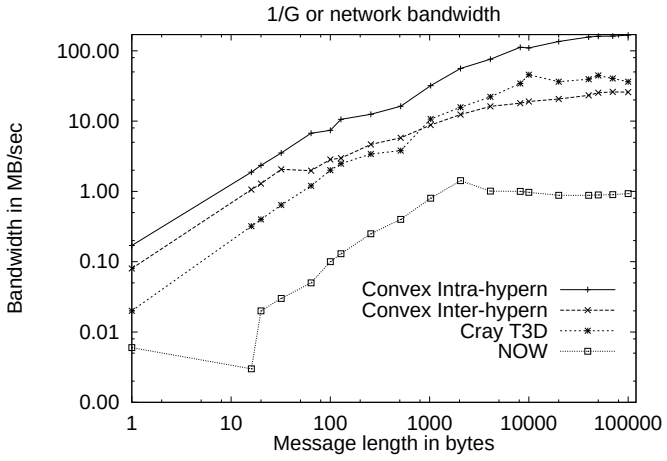

Figure 13: Comparison of $1/G$ parameters, corresponding to network bandwidths for large messages on Cray T3D, NOW and Convex Exemplar.

message transfer rates for inter-cluster communication are factor of ten larger than the inter-cluster ones. Our results show that the MPI software overheads are very high and should be improved in order to utilize the high speed bandwidth provided by the underlying hardware.

Acknowledgments: We would like to acknowledge the support of the TRACS program and KFUPM. We like also to thank EPCC and the Department of Computer Science of the University of Edinburgh, United Kingdom, where part of this work was done. We would like to thank the CSCA Supercomputing Center in Zurich for the help with the different configurations of the Convex Exemplar 1600SP machine.

References

[1] G. Abandah and E. Davidson, "Modeling the Communication Performance of the IBM SP2," *Proc. 10th IEEE Int'l Parallel Processing Symp.*, Honolulu, Hawaii, pp. 249-257, April 15-19, 1996.

[2] R. Alasdair, A. Bruce, J. Mills, and A. Smith, "CHIMP/MPI User Guide," Technical Report EPCC-KTP-CHIMP-V2-USER, Edinburgh Parallel Computing Centre, The University of Edinburgh, June 1994.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP:Incorporating Long Messages into the LogP Model - one step closer towards a realistic model fpr parallel computation," *Proc. Seventh Ann. ACM Symp. on Parallel Algorithms and Architectures SPAA'95*, New York, pp. 95-105, 1995.

[4] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation", *Proc. of the 19th Int. Symp. on Computer Architecture, pages 256-266'*, Gold Coast, Australia, May 1992.

[5] C Andras Moritz, Matthew I.Frank. "LoGPC: Modeling Network Contention in Message-Passing Applications." *In Performance Evaluation Review , Special Issue Volume 26 no. 1 and Proceedings of ACM Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS /PERFORMANCE 98*, Wisconsin Madison, June 1998.

[6] C. Andras, K. Al-Tawil, and B. Basilio, "Performance Comparison of MPI on MPP and Workstation Clusters," *Proc. 10th Int'l Conf. Parallel and Distributed Computing Systems*, New Orleans, Louisiana, pp. 167-172, Oct. 1-3,1997.

[7] G. Burns, R. Daoud, and J. Vaigl, "LAM: An Open Cluster Environment for MPI," Ohio Supercomputer Center, May 1994.

[8] R. Barriuso and A. Knies, "SHMEM User Guide for C," Technical Report Revision 2.2, , Cray Research Inc., August 1994.

[9] K. Cameron, L. J. Clarke, A. G. Smith, "CRI/EPCC MPI for CRAY T3D" Technical Report, Edinburgh Parallel Computing Centre, The University of Edinburgh, 1995.

[10] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. of Fourth ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming*, vol 28, pp. 1-12, May 1993.

[11] D. Culler, L. Liu, R. Martin, and C. Yoshikawa, "LogP Performance Assessment of Fast Network Interfaces," IEEE Micro, 1996.

[12] F. Cheng, P. Vaughan, D. Reese, and A. Skjeellum, "The Unify System," Technical Report, Mississippi State University, July 1994.

[13] B. Gropp, R. Lusk, T. Skjellum, and N. Doss, "Portable MPI Model Implementation," Argonne National Laboratory, July 1994.

[14] K. Keeton, T. Anderson, and D. Patterson, "LogP Quantified: The Case for Low-Overhead Local Area Networks," *Hot Interconnects III: A Symp. on High Performance Interconnects*, Stanford University, Stanford, CA, Aug. 10-12, 1995.

[15] M. I. Frank, A. Agarwal, M. K. Vernon, "LoPC:Modeling Contention in Parallel Algorithms", *Proc. of the SIXTH ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Las Vegas, Nevada, June 18, 1997.

[16] MPI Committee, "MPI: A message-passing interface standard," *Int'l J. Supercomputer Applications*, Vol. 8, pp. 165-416, Fall/Winter 1994.

[17] MPI Forum, "MPI: A message-passing interface standard," Technical Report UT-CS-94-230, Department of Computer Science, University of Tennessee, 1994.

[18] N. Nupairoj and L. Ni, "Performance Evaluation of Some MPI Implementations," Technical Report MSU-CPS-ACS-94, Department of Computer Science, Michigan State University, Sept. 1994.

[19] R. W. Numrich, P. L. Springer and J. C. Peterson, "Measurement of Communication rates on the Cray T3D Interprocessor Network," Proc. HPCN Europe 1994, Munich, April 18-20.

[20] P. Worley, "MPI Performance Evaluation and Characterization using a Compact Application Benchmark Code," *Proc. Second MPI Developers Conference and Users' Meeting*, eds. A Lumsdaine and A. Skjellum, IEEE Computer Society Press, pp. 170-177, 1996.