Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2 Multicomputer

Zhiwei Xu and Kai Hwang

CENG Technical Report 95-14

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213)740-4470

Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2 Multicomputer*

Zhiwei Xu and Kai Hwang
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089
{kaihwang,zxu}@aloha.usc.edu

Summary: To develop a parallel program on a message-passing multicomputer, the user needs to know the delays and overheads incurred with various communication operations, such as *point-to-point communication*, *collective communication*, and *collective computation*. This information provides a basis for selecting the most efficient grain sizes and parallelization strategy. Hockney [3] modeled the point-to-point communication overhead t using a linear equation $t=t_0+m/r_\infty$, where the message length m is a variable and the latency t_0 and the bandwidth r_∞ are both constants.

In this paper, we generalize Hockney's formula to cover all three types of communication operations, using a non-linear equation $t=t_0(n)+m/r_\infty(n)$, where both t_0 and r_∞ are a simple function of the number of nodes n involved in a communication operation. This formula is validated by a performance study of the *Message Passing Interface* (MPI) and the IBM *Message-passing Library* (MPL) on a 400-node IBM SP2 multicomputer. Overheads of all three types of communication operations were measured with various combinations of machine size and message length. The collected performance data are used to derive the constants in the timing formulae. These formulae have been successfully used in the development of a suite of message-passing parallel programs for real-time radar signal processing.

Our results show that the overhead for a single communication operation on SP2 can be long enough to perform thousands or more floating-point operations. Therefore, only coarse-grain parallelism should be exploited on SP2. Compared with the native IBM MPL library, MPI performs equally well for point-to-point communications. For collective communication or collective computation operations, the derived overhead formulae not only indicate that the current portable implementation of MPI (MPICH) is slower than MPL, but also identify where the bottleneck lies. Modeling communication delays, overhead, and bandwidth becomes critically important for designers and programmers of massively parallel processors. Our communication models can be used by both MPP designers and message-passing programmers in the optimization of system/program performance.

---

* This work was supported in part by a research subcontract from MIT Lincoln Laboratory to the University of Southern California. For all future correspondence, contact Kai Hwang via Email: kaihwang@aloha.usc.edu.

Contents :

1. Introduction

There are three types of fundamental communication operations on current message-passing multicomputers. In a *point-to-point communication*, only two tasks are involved: the source task sends a message to the destination task. In a *collective communication*, a group of tasks (usually more than two) send messages to one another. In a *collective computation*, a group of tasks synchronize with one another or aggregate local values into global values. To develop a parallel program on a message-passing multicomputer, the user needs to know the overheads incurred by these operations. These overheads depend on many factors, such as the number, n, of computing nodes involved, the message length m, and the type of communication operation. It would be ideal if the user can estimate overheads with a simple, closed-form formula.

Hockney [3] has suggested a simple linear equation to model the overhead of point-to-point communications. In this paper, we generalize Hockney's model to cover all three types of message-passing communication operations. This new communication overhead model is validated by a performance study of the *Message Passing Interface* (MPI) and the IBM *Message-passing Library* (MPL) on a 400-node IBM SP2 multicomputer at Maui High-Performance Computing Center (MHPCC) [11]. Up to 256 dedicated nodes of the SP2 were used. Overheads of all three types of communication operations were measured with various combinations of machine size and message length. The collected timing data are used to derive the constants in the timing formulae. Overall, overheads projected by these formulae match closely with the measured overheads on SP2.

The new overhead model are used to evaluate the MPL and the MPI message-passing libraries. MPI [9,10] is a public-domain standard, developed by the MPI Forum, a broadly-based consortium of parallel computer vendors, software writers, and application specialists. MPL [6,7] is an proprietary library developed by IBM to support message passing functions on the IBM RS/6000 architecture, including from low-end, PowerPC-based workstation clusters to high-end distributed memory SP2 multicomputers. The overhead model is used to quantitatively compare

these two libraries and to identify implementation problems.

The overhead model is also very useful for parallel programs development on any multicomputer like SP2. It provides a basis for selecting the most efficient grain sizes and parallelization strategy. We have successfully used this model in parallelizing a suite of radar signal processing applications [5] consisting of more than 4,000 lines of C code. The model predicts various communication overheads rather close to measured ones, within 25% of accuracy.

The new communication overhead model is general enough to characterize the communication performance of other multicomputer architectures using different communication software packages. Although the timing formulae derived from this model are obtained from measured data on SP2, they can be modified to predict other message-passing systems such as Intel Paragon, Meiko CS2, etc; in most cases one needs to change only a few constants.

2. The SP2 Platform and Testing Conditions

The IBM SP2 is an message-passing, multicomputer consisting of a large number of computing nodes with distributed memories. These nodes are interconnected by an internal ethernet as well as a *High-Performance Switch* (HPS). The HPS is an any-to-any, packet-switched, multi-stage Omega network. The SP2 system at MHPCC consists of 400 nodes. Each node employs a 66.7 Mhz POWER2 processor, which is a high-performance implementation of the IBM RS/6000 architecture. At each clock cycle, the POWER2 superscalar processor can issue up to six instructions, and perform up to 4 floating-point operations on its two Floating-Point Units. Thus, a single node has a floating-point speed of 266 MFLOPS in its peak performance.

To conduct a meaningful performance evaluation, we have assumed the following testing conditions throughout the benchmark experiments:

(1) The data structures used are made small enough to fit in each node memory of 64 MB,

so there will not be extensive page faults for disk I/O operations.

(2) All test programs are written in C. The only library functions used are standard I/O functions, timing functions, and MPI (or MPL) functions. No machine-specific library functions nor any assembly codes are used.

(3) The best compiler options are used: *cc -O3 -qarch=pwr2 program.c* for sequential programs and *mpcc -O3 -qarch=pwr2 program.c* for parallel programs. The test programs for MPI are run using the "mpirun" script file.

(4) Use the system resources as dedicated as possible, so that the interferences from the operating system and other user tasks can be minimized. Each of the 256 nodes is set to be solely used by one task, by specifying in the host.list file:

hostname dedicated unique

We also set the following environment variables for dedicated use of the HPS:

setenv EUIDEVICE css0

setenv EUILIB us.

However, during our testing, other user tasks were also running in the remaining nodes of the 400-node SP2, which demanded portion of the HPS. Therefore, the communication operations of our programs had to compete for the HPS against other user tasks.

(5) We measure both the wall clock time and the CPU time. The Unix function gettimeofday( ) is used to measure the wall clock time, and the Unix times( ) function is used to measure the CPU time. Only wall-clock times are used in interpreting results and deriving timing formulae.

(6) Each test program is executed multiple times. The minimal time, the maximal time, and the mean time of each communication operation are collected. However, To interpret the results, we focus on the minimal time, because we feel it is the most

accurate one, with the least interference from the OS and other users.

(7) In all communication operations, single-precision (4-byte) floating-point numbers are used. In other words, for MPI the data type of the message elements is always MPI_FLOAT.

In the IBM SP2 native parallel programming environment, user tasks are scheduled statically. In all of our testing runs, we assign at most one task to a node. The tasks of a program can form different *groups*. We always use one unique group: the group of all tasks. Thus the group size is always equal to the number of tasks in a program run, which in turn is equal to the number of nodes used in that run. The *number of nodes* (tasks) used, denoted by n, range to a maximum of 256 nodes. We denote the *message length* by m (bytes), where m ranges from 4 bytes to 16 MBytes.

In each test run, each message-passing operation is executed twenty times, and the average time of the last eighteen executions is calculated on each task. The maximum of all these n average timing values, one from each task, is reported. Five test runs are done for every n nodes (n=2, 4, 8, ..., 256). The reason for filtering out the first two executions is that it is common on SP2 that the first one or two executions of a communication function could take considerably longer (more than one order of magnitude) than the rest of executions, probably due to the overhead of loading the function into the memory or the cache.

3. MPI and MPL Message-Passing Functionalities

The functionalities of MPI and MPL are compared in Table 1. Among its objectives, MPI is designed to be portable, powerful, and efficient. The portability of MPI is supported by its public domain and machine independent nature. Compared to the widely used public domain library PVM (Parallel Virtual Machine) [2], MPI is indeed more powerful for message-passing, with its enhanced capabilities in collective communication, communicator (context and grouping),

virtual topology, etc. Not intended to be a complete software package for parallel processing, both MPI and MPL only provide *message-passing* functionalities. They do not provide functionalities such as task creation or machine configuration as PVM does.

Table 1. Comparison of MPI and MPL Message-Passing Functionalities

Attributes	MPI	MPL
Portability	A Standard, Architecture Independent, Public-Domain Implementations Available	IBM Proprietary, Portable among any versions of the RS/6000 Architecture
Total Number of Functions	125	32
Point-to-Point Communication	Blocking Send/Receive, Nonblocking Send/Receive	Blocking Send/Receive, Nonblocking Send/Receive
Collective Communication	Broadcast, Gather, Scatter, Total Exchange	Broadcast, Gather, Scatter, Total Exchange, Shift
Collective Computation	Barrier, Reduction, Parallel Prefix (Scan)	Barrier, Reduction, Parallel Prefix (Scan)
Group Management	Group Construction, Destruction, Inquiry, etc.	Group Construction, Inquiry, etc.
Communicator Management	Inter- and Intra-Communicator Construction, Destruction, etc.	N/A
Topology	Various Topology Management Functions	N/A

MPI and MPL have about the same functionality for point-to-point communication, collective communication, and collective computation operations. MPI has introduced two new concepts: the *communicators* and the *topology*. A communicator is a task group plus the notion of safe communication context. Communicators guarantee that a communication is isolated from other communications in the system. This is valuable for supporting safe inter-group

communication and parallel libraries. Topology is useful in mapping applications to tasks groups. There exist several implementations of MPI, including the IBM experiment MPI-F [1]. The MPI used in this study is MPICH, a portable implementation of the full MPI specification developed at Argonne National Laboratory [9].

There were very few studies on the performance of MPI or MPL in the past [1,8]. The existing studies focused on point-to-point communication. Our study is the first one covering all three types of communication operations over a large range of machine sizes. These communication operations are illustrated in Fig.1 for readers not familiar with the terminology.

In a *point-to-point* communication, one task sends a message to another task. Thus only two tasks, one sender and one receiver, are involved. Note that the original copy of the message in the sender task is always retained. The IBM SP2 uses a HPS to speedup up the message passing operations. Theoretically, all nodes have equal distance from one another. The concept of neighboring nodes or remote nodes does not exist in SP2. Consequently, the time for a point-to-point communication is a function of message length, but not of the number of nodes, i.e., $t=f(m)$. This is different from other multicomputers (e.g., Intel Paragon) that employ a point-to-point network, where communication between two neighboring nodes is less expensive than between two remote nodes.

In a *collective communication operation*, a group of tasks send messages to one another, and the *communication time*, including all communication overhead and delays, is a function of both the message length and the group size, namely,

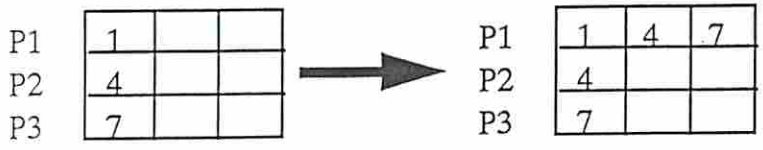
$$t=f(m,n) \tag{1}$$

Five representative collective communication operations were measured:

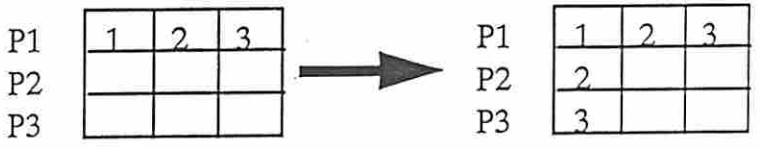
- In a *broadcast* operation (MPI_Bcast in MPI or mpc_bcast in MPL), node 0 sends an m-byte message to all n nodes.
- In a *gather* operation (MPI_Gather or mpc_gather), node 0 receives an m-byte message from each of the n nodes, so in the end mn bytes are received by node 0.

(a) Point-to-point communication: Task P1 sends a message to task 3

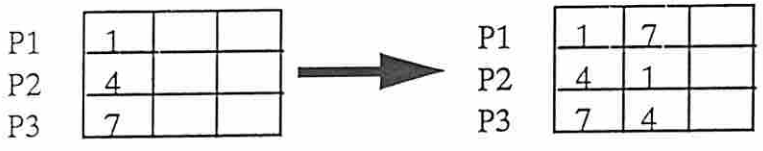
(b) Broadcast: Task P1 sends a message to all tasks



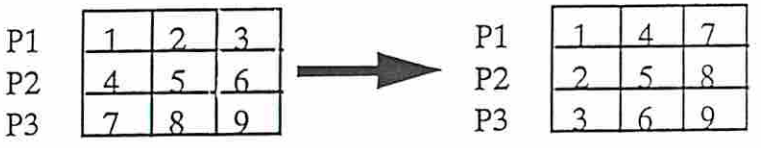
(c) Gather: Task P1 receives a distinct message from each of the three tasks



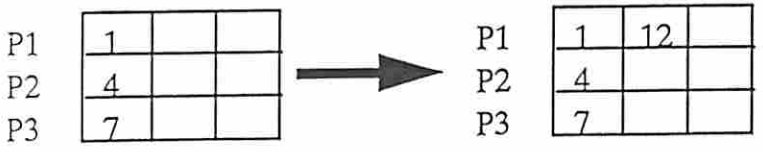
(d) Scatter: Task P1 sends a distinct message to each of the three tasks



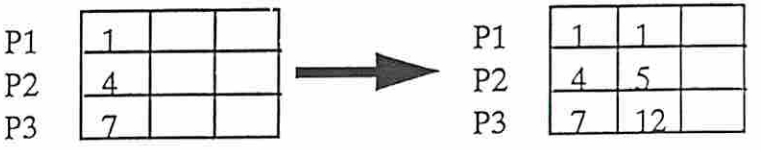
(e) Circular Shift



(f) Total Exchange: Every task sends a distinct message each of the three tasks



(g) Reduction: Task P1 gets the sum of three numbers, one from each of the three tasks, i.e., 1+4+7=12



(h) Parallel Prefix: P1 gets 1, P2 gets 1+4, and P3 gets 1+4+7=12

Figure 1. Examples of Frequently Used Communication Operations

- In a *scatter* operation (MPI_Scatter or mpc_scatter), node 0 sends a distinct m-byte message to each of the n nodes, so in the end mn bytes are sent out by node 0.
- In a *total exchange* (or *all-to-all*) operation (MPI_Alltoall or mpc_index), every of the n nodes sends a distinct m-byte message to each of the n nodes, so in the end mn^2 bytes are communicated.
- In a *circular-shift* operation (mpc_shift), node i sends an m-byte message to node i+1, and node n-1 sends m bytes to node 0.

In a *collective computation operation*, a group of tasks is involved to synchronize among one another or to aggregate partial results. Usually, the time for such an operation is a function of the group size, but not of the message length, as the message length is fixed (i.e., $t=f(n)$). Three representative collective computation operations were measured:

- In a *barrier operation* (MPI_Barrier or mpc_sync), a group of tasks synchronize with one another, i.e., they wait until all tasks execute their respective barrier operation.
- In a *reduction operation* (MPI_Reduce or mpc_reduce), a group of tasks aggregate partial results, one from each task, into a final result. Typical examples of reductions are to sum or to find the maximum of n values, one from each of the n tasks.
- In a *parallel prefix operation* (MPI_Scan or mpc_prefix), also known as a *scan*, a group of tasks aggregate n partial results, one from each of the n tasks, into n final results.

4. Communication Performance Metrics

Hockney [3] suggested an elegant model to characterize the *communication time* (in μs) of a point-to-point communication operation as follows:

$$t = t_0 + \frac{m}{r_\infty} \tag{2}$$

Here m denotes the message length in bytes. The parameter r_∞ is called the *asymptotic bandwidth*

in MB/s, which is the maximal bandwidth achievable when the message length approaches infinity. The parameter t_0 is called the *latency* (or *startup time*) in μs, which is the time needed to send a 0-byte message. Two additional parameters can be derived. The *half-peak length*, denoted by $m_{1/2}$ bytes, is the message length required to achieve half of the asymptotic bandwidth. The *specific performance*, denoted by π_0 MB/s, indicates the bandwidth for short messages. Only two of these four parameters are independent. The other two can be derived by

$$t_0 = \frac{m_{1/2}}{r_\infty} = \frac{1}{\pi_0} \tag{3}$$

Hockney's model (2) has several nice properties. It is *simple*, a linear function of a single variable m. It is *meaningful*: the two parameters t_0 and r_∞ each represent a fundamental quantity of point-to-point communication. It is *architecture-independent*, in that the same model can be applied to different architectures, by changing the values of the two parameters. It is *fairly accurate*, meaning the projected values computed from Equation (2) match closely the measured communication times. In [3], Hockney showed how the linear model was used to characterize communication overheads on Intel Paragon, Intel Delta, Meiko CS-2, and Cray C90.

However, Equation (2) is only suitable for point-to-point communication operations. It needs to be generalized for collective communication and collective computation operations. There are two reasons: First, the times for these collective operations may be a function of both the message length m and the number of nodes n. Second, the functions are usually not linear with respect to n. There are many possibilities to generalize Equation (2). However, a good one should maintain the nice properties of the original Hockney model. We have tried several generalizations and found the best one to be the following: The communication time t is still a linear function of the message length m. However, the latency and the asymptotic bandwidth are now simple functions of the number of nodes n. We define below a generalized timing model:

$$t = t_0(n) + \frac{m}{r_\infty(n)} \tag{4}$$

When n is fixed, the latency and the asymptotic bandwidth will become constants. For collective

computation operations, we only need to focus on the latency. The asymptotic bandwidth can be viewed as a constant. To keep the new model (Equation (4)) simple, the latency and the asymptotic bandwidth must be simple functions of n.

To summarize, we want to derive the communication performance metrics listed in Table 2 for each of the communication operations. The bandwidth r_∞ is the one-way bandwidth for sending an m-byte message between two nodes. To get a sense of how well SP2 supports various collective communication operations, we included in Table 2 another metric, the *aggregated asymptotic bandwidth*, denoted by R_∞ MB/s, which is defined as the ratio of the total number of bytes transmitted in a communication operation to the time of that operation, when the message size m approaches infinity. This metric reflects the aggregated communication capability of the HPS. For a point-to-point communication, $R_\infty=r_\infty$. For a *total exchange*, $R_\infty=n^2r_\infty$. For other collective communications (*broadcast*, *gather*, *scatter*, and *shift*), $R_\infty=nr_\infty$.

Table 2. Performance Metrics for Communication Functions

Communication Time	$t = t_0(n) + \frac{m}{r_\infty(n)}$ μs
Latency	t_0 μs
Asymptotic Bandwidth	r_∞ MB/s
Half-Peak Length	$m_{½}$ Bytes
Aggregated Asymptotic Bandwidth	R_∞ MB/s

Note: n = number of nodes involved, m = message length

5. Point-to-Point Communication

MPI and MPL provide several point-to-point communication functions, we measured the *blocking send* and *blocking receive* operations, since they are representative. The well-known *pingpong* (also known as *hot-potato*) [3] scheme is used in an n-task run: Task 0 executes a

blocking send (MPI_Send or mpc_bsend) to send a message of m bytes to task n-1, which executes a corresponding blocking receive (MPI_Recv or mpc_brecv). Then task n-1 immediately sends the same message back to task 0. All other tasks do nothing. The total time for this pingpong operation is divided by 2 to compute the point-to-point communication time. The results are shown in Fig.2. Our results confirm the fact that there are only small differences in sending messages to different nodes from the same source on SP2.

Using least-square fitting of the measured data, we derived the performance metrics as shown in Table 3. The time formulae of MPL and of MPI are accurate within 20% for most message length m and number of nodes n combinations. The details are discussed in [14,15]. Figure 2 shows that the measured communication times and projected times match closely. In both Table 3 and Fig. 2, we compare our results to the best results reported from IBM [12].

Table 3. Hockney Parameters for Point-to-Point Communication

Metric / System	Communication Time t	Latency t_0	Asymptotic Bandwidth r_∞	Half-Peak Length $m_{1/2}$
MPL	46+0.035m	46 μs	28.67 MB/s	1314 Bytes
MPI	67+0.035m	67 μs	28.67 MB/s	1921 Bytes
IBM	39+0.028m	39 μs	35.54 MB/s	1386 Bytes

The time for a point-to-point communication is a linear function of the message length m, and can be considered independent of the number of nodes n. This communication time can be estimated by Equation (2). For both MPI and MPL, the best bandwidth achieved in our experiments is 35.54 MB/s, which is exactly what IBM reported. An optimistic user may want to use the IBM metrics. The IBM time formula in Table 3 is accurate within 25% for most cases, and it is actually more accurate than the MPI and the MPL formulae for larger messages. The latter two formulae are more accurate for smaller messages. They tend to overestimate the communication time for large messages. The MPI time is slightly longer. However, the difference

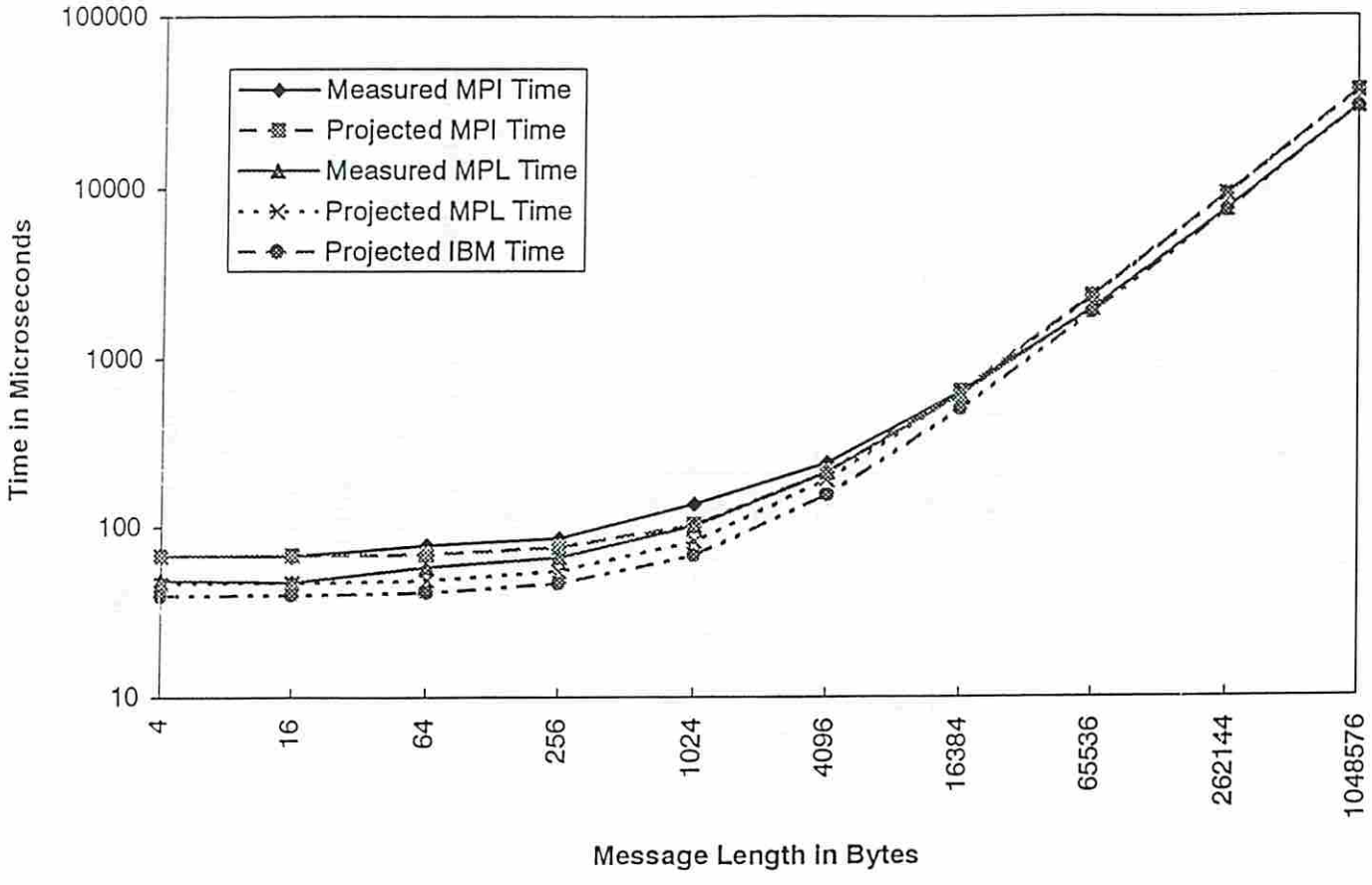


Figure 2. Measured Versus Projected Times for Point-to-Point Communication

is small, especially when medium to large messages are used.

The derived performance metrics revealed a weakness in current implementations of MPI and MPL. The HPS poses a hardware limitation of about 1 μs latency and about 40 MB/s bandwidth for point-to-point communication. At the application level, although the asymptotic bandwidth is close to the hardware limit, the latency is much large than the hardware latency. This hurts short messages. IBM has made great stride in reducing the latency. In the earlier implementation using the IP protocol, the latency is even larger, about 600 μs by our measurement. This is because a communication has to involve kernel calls. The new *User Space* protocol bypasses the kernel, thus reducing the latency to only 39-67 μs. There is still room to further improve the latency, by using techniques such as *active message* [13].

6. Collective Communication Operations

We measured the execution times of four MPI and five MPL collective communication operations over 2 nodes to 256 nodes. The timing results over 32 nodes are shown in Figs. 3-6. We measured all the five MPL operations up to 16MB. However, the MPI program deadlocked when testing *gather*, *scatter*, and *total exchange* using messages of 16 KB or more. Therefore, only the timing data for up to 4 KB are used for those operations. The MPI broadcast operation worked fine for up to 1 MB messages. It encountered "insufficient memory" errors when larger message sizes were used.

After fitting the measured timing data according to Equation (4), we derived the performance metrics for the collective communication operations as shown in Tables 4-6. Throughout the entire paper, all logarithmic functions are base-2. The projected times are plotted in Figs. 3-6. Overall the projected times match the measured times rather closely. The half-peak length $m_{1/2}$ is not shown in Tables 5 and 6, but it can be easily computed using $m_{1/2}=t_0 \times r_\infty$. As a concrete example, the values of the Hockney parameters over 32 nodes are computed in Tables 7 and 8.

Table 4. Comparison of Collective Communication Times in MPI and MPL

Operation	MPL Timing Formula (μs)	MPI Timing Formula (μs)
Broadcast	$(16\log n+10)+(0.025\log n)m$	$(40\log n+20)+(0.037\log n)m$
Gather	$(17\log n+15)+(0.025n-0.02)m$	$(24n+84)+(0.045n)m$
Scatter	$(17\log n+15)+(0.025n-0.02)m$	$(24n+105)+(0.026n+0.03)m$
Shift	$(6\log n+60)+(0.003\log n+0.04)m$	Not Measured
Total Exchange	$80\log n+(0.03n^{1.29})m$	$(125n-22)+(0.06n^{1.29})m$

Table 5. Hockney Parameters for MPI Collective Communication

Operation	Latency t_0 (μs)	Asymptotic Bandwidth r_∞ (MB/s)	Aggregated Asymptotic Bandwidth R_∞ (MB/s)
Broadcast	$40\log n+20$	$27/\log n$	$27n/\log n$
Gather	$24n+84$	$22/n$	22
Scatter	$24n+105$	$1/(0.026n+0.03)$	$n/(0.026n+0.03)$
Total Exchange	$125n-22$	$16.7n^{-1.29}$	$16.7n^{0.71}$

Table 6. Hockney Parameters for MPL Collective Communication

Operation	Latency t_0 (μs)	Asymptotic Bandwidth r_∞ (MB/s)	Aggregated Asymptotic Bandwidth R_∞ (MB/s)
Broadcast	$16\log n+10$	$40/\log n$	$40n/\log n$
Gather/Scatter	$17\log n+15$	$1/(0.025n+0.03)$	$n/(0.025n+0.03)$
Shift	$6\log n+60$	$1/(0.003\log n+0.04)$	$n/(0.003\log n+0.04)$
Total Exchange	$80\log n$	$33.3n^{-1.29}$	$33.3n^{0.71}$

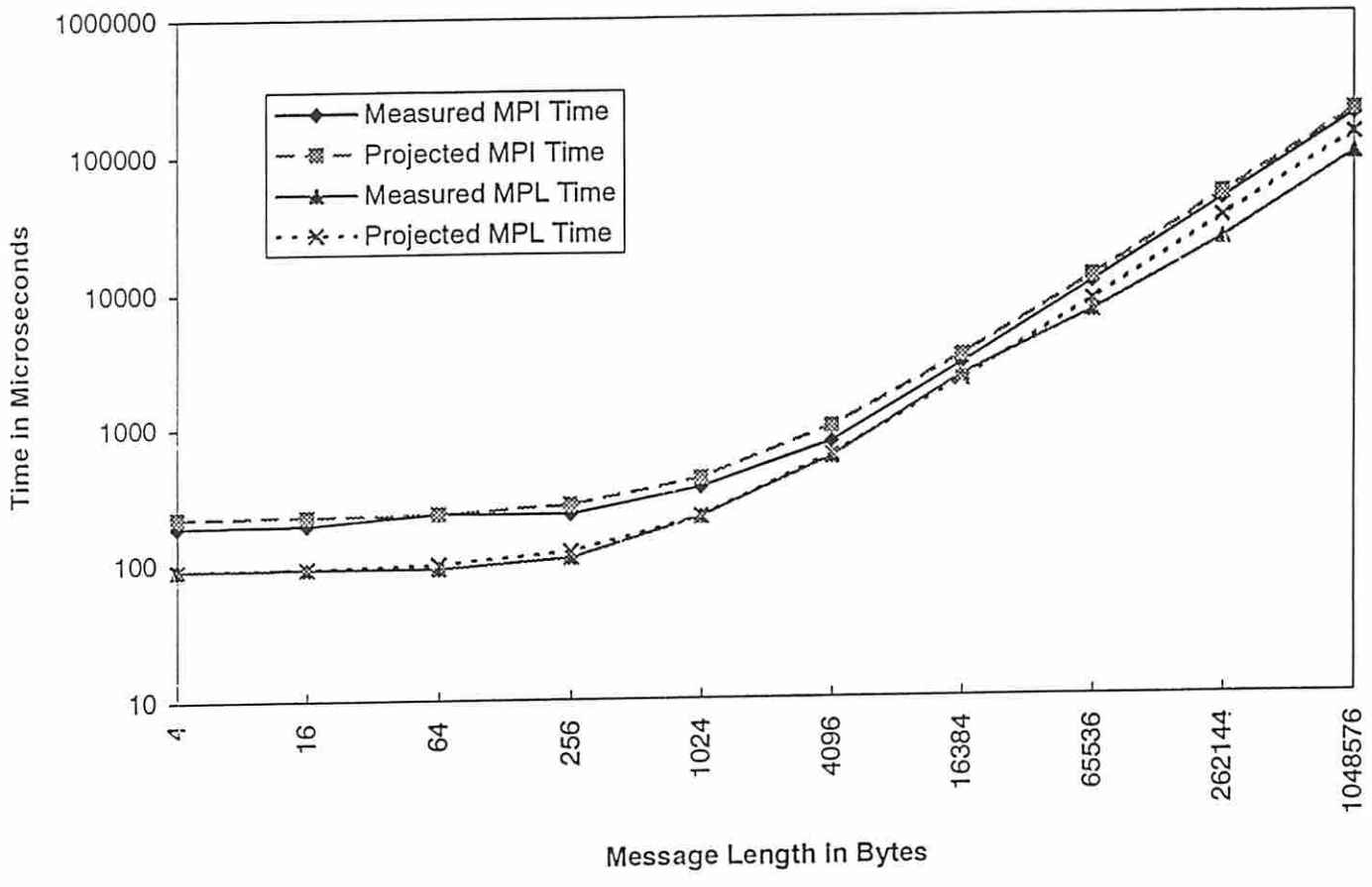


Figure 3. Measured Versus Projected Times for Broadcast over 32 Nodes

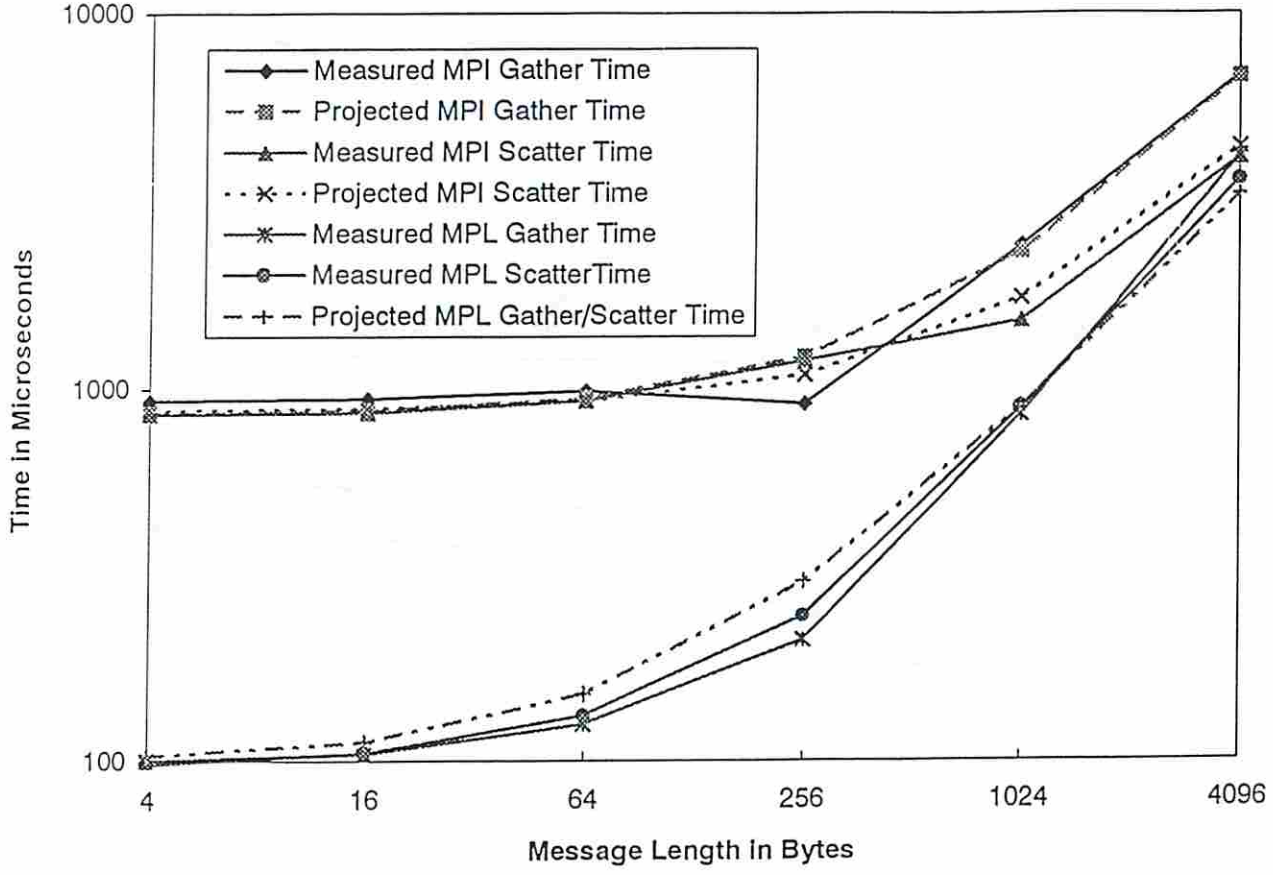


Figure 4. Measured Versus Projected Times for Gather and Scatter over 32 Nodes

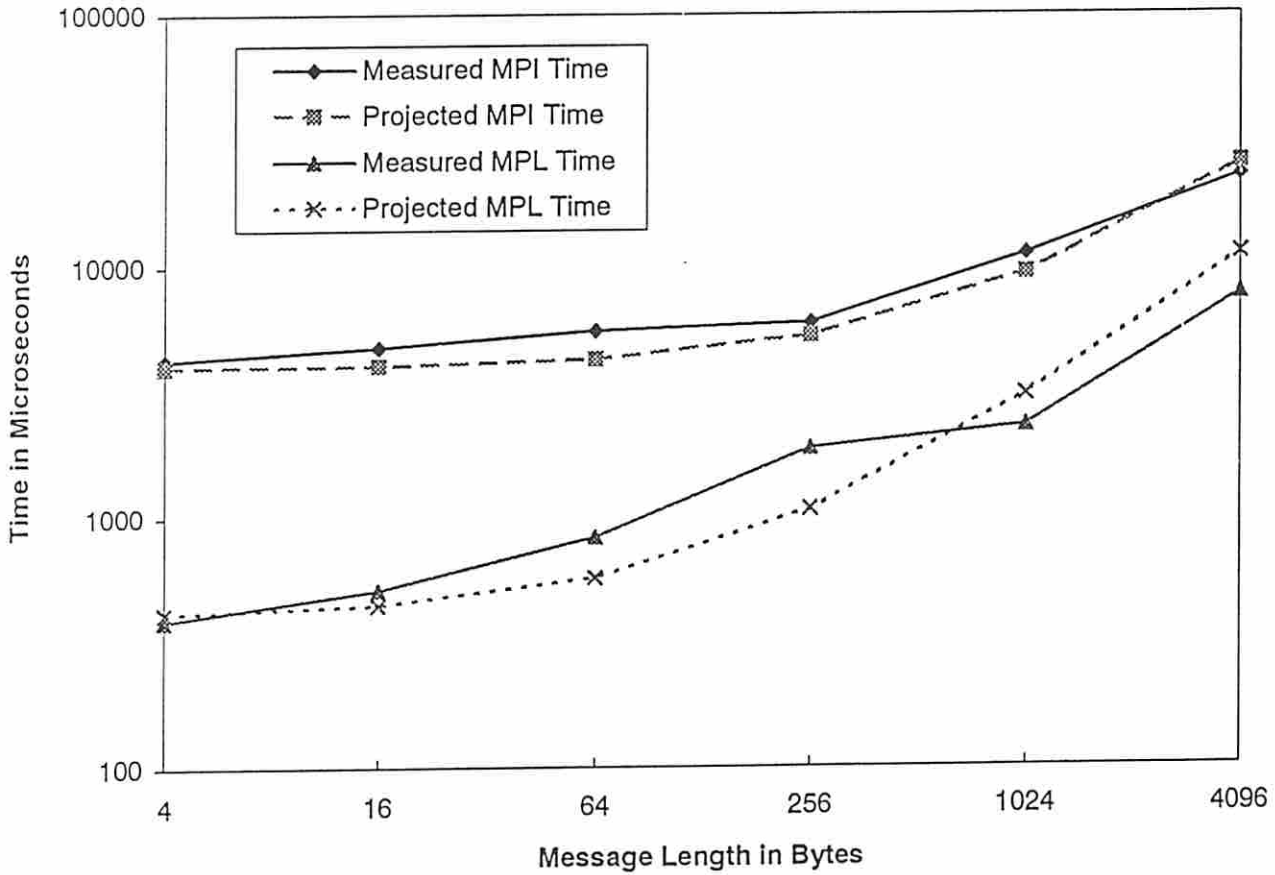


Figure 5. Measured Versus Projected Times for Total Exchange over 32 Nodes

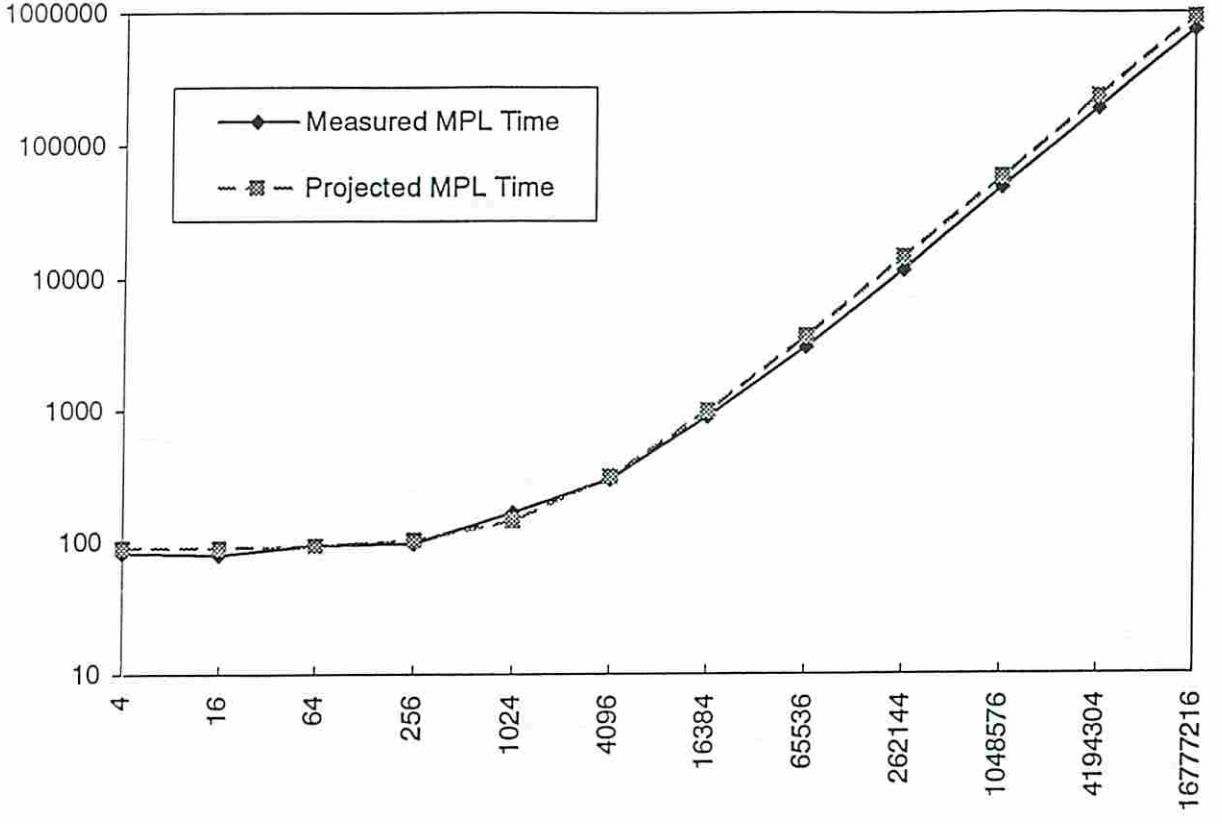


Figure 6. Measured Versus Projected Times for Circular Shift in MPL over 32 Nodes

Table 7. Hockney Parameters for MPI Collective Communication on 32 Nodes

Operation	Latency t_0 (μs)	Asymptotic Bandwidth r_∞ (MB/s)	Half-Peak Length $m_{1/2}$ (Bytes)	Aggregated Asymptotic Bandwidth R_∞ (MB/s)
Broadcast	220	5.40	1188	173
Gather	852	0.69	586	22
Scatter	873	1.16	1013	37
Total Exchange	3978	0.19	759	195

Table 8. Hockney Parameters for MPL Collective Communication over 32 Nodes

Operation	Latency t_0 (μs)	Asymptotic Bandwidth r_∞ (MB/s)	Half-Peak Length $m_{1/2}$ (Bytes)	Aggregated Asymptotic Bandwidth R_∞ (MB/s)
Broadcast	522	8.00	4176	256
Gather/Scatter	100	1.20	120	39
Shift	90	18.18	1637	582
Total Exchange	400	0.38	153	390

The performance differences between MPI and MPL are significantly larger for collective communication operations. The communication time can be estimated by Equation (4). Both the MPL and the MPI timing formulae for *broadcast* have the same form. The latency $t_0(n)$ is a logarithmic function of n, so is the inverse of the asymptotic bandwidth. However, the constants are quite different. Consequently, the latency of an MPI broadcast is 150% longer than that of an MPL one. The asymptotic bandwidth of an MPI broadcast is only 68% of that of MPL. For gather, scatter, and total exchange, the latencies for MPI are linear function of n, while those for

MPL are only logarithmic functions. The asymptotic bandwidth for MPI total exchange is only half of that for MPL. These results show that there exists much room for improvement for MPI functions, especially on their latencies.

We also calculated the *aggregated bandwidth*, i.e., how many bytes are communicated by all n nodes in a second. The highest aggregated bandwidth achieved in our experiment is 969 MB/s for MPI in a 256-node, 1-MB broadcast operation. For MPL, the same broadcast achieved 1,593 MB/s aggregated bandwidth. The highest aggregated bandwidth achieved for MPL is 3,779 MB/s in a 256-node, 4-MB circular shift operation.

7. Collective Computation Operations

Three collective computation operations are measured. The timing results are shown in Fig. 7. For MPI, the timing data for parallel prefix are not reliable, thus only the results for barrier and reduction are reported. In all cases, the barrier is the most expensive operation, while the reduction is the least expensive one. The timing differences among these operations become more pronounced as the number of nodes n increases. We curve-fitted the timing data for the collective computation operations. The resultant timing formulae are shown in Table 9. Figure 7 shows that the projected times match nicely with the measured times.

Table 9. Collective Communication Timing Formulae

Operation	MPL Timing Formula	MPI Timing Formula
Barrier	$94\log n+10$	$150\log n-30$
Parallel Prefix	$60\log n-25$	Data Not Reliable
Reduction	$20\log n+23$	$45\log n+15$

Again, there are significant performance differences between MPI and MPL collective

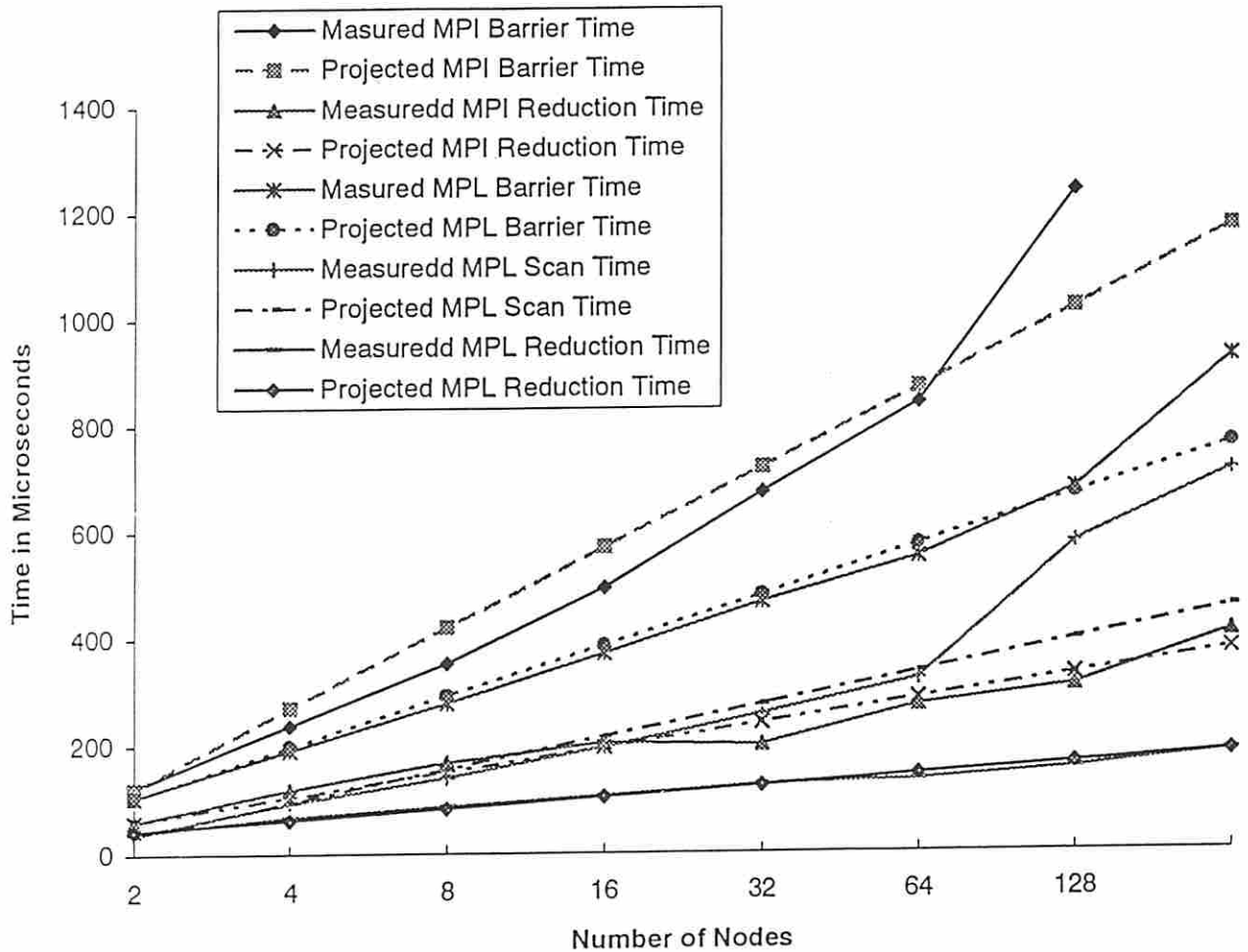


Figure 7. Measured Versus Projected Times for Barrier, Reduction, and Scan

computation operations. In both cases, the time for a collective computation operation is a linear function of the logarithm of the number of nodes n, i.e., $t=c\log n+d$. The coefficient c is the important constant, which indicates how many more microseconds are needed when the number of nodes is doubled. Our results show that when the number of nodes is doubled, an MPL barrier will take 94 more μs, an MPL parallel prefix will take 60 more μs, and a MPL reduction will take only 20 more μs. An MPI barrier takes 1.6 times as long as an MPL barrier. Similarly, an MPI reduction takes 2.25 times as long as an MPL reduction.

When the machine size becomes very large, such as greater than 128 nodes, the projection deviates far from the measured results for the barrier and the scan (parallel prefix) operations. This was caused by unreliable timing from interferences to the HPS in SP2, when more nodes are involved in the measurement. If these interferences were completely removed, the deviation can be also eliminated accordingly.

8. Concluding Remarks

We have shown that the performance of all three types of communications can be modeled by the generalized Hockney equation (4), where the latency and the asymptotic bandwidth are simple functions of the number of nodes n. The forms of these functions for both MPI and MPL are summarized in Table 10.

Most of the metric items for MPL make sense from the architecture angle. For instance, the latencies of all collective operations are O(log n). This is understandable since the Omega network in the HPS has O(log n) stages. For large messages, the time for a gather (or scatter) should be O(mn), because n messages, each m bytes, need to be received (sent) by one node sequentially. The $O(n^{-1.29})$ bandwidth for total exchange is quite respectable, as the much less complex gather operation can only achieve O(n).

The boldfaced items need improvement. There is only one such item for MPL: The

asymptotic bandwidth for broadcast should be a constant on a pipelined Omega network, not decreasing for larger machine size. The 1/O(log$n$) asymptotic bandwidth suggests that either the current MPL implementation does not exploit pipelining in the HPS or the effect of pipelining is canceled out by other factors.

Table 10. Summary of Hockney Parameters for MPI and MPL

Metric / Operation	MPI		MPL	
	Latency t_0	Asymptotic Bandwidth r_∞	Latency t_0	Asymptotic Bandwidth r_∞
Point-to-Point	Constant	Constant	Constant	Constant
Collective Computation	O(log$n$)	Constant	O(log$n$)	Constant
Broadcast	O(log$n$)	1/O(log$n$)	O(log$n$)	1/O(log n)
Gather/Scatter	O(n)	1/O(n)	O(log$n$)	1/O(n)
Total Exchange	O(n)	O($n^{-1.29}$)	O(log$n$)	O($n^{-1.29}$)

MPI performs almost equally well for point-to-point communication. This is impressive giving that MPL is machine-specific, proprietary library, while MPI is a machine-independent, public domain library standard. However, MPI is still significantly slower than MPL for collective communication and collective computation operations. As Tables 4, 9, and 10 indicate, the latencies of all operations, and the bandwidth for barrier and total exchange, could be improved. We must stress that the version of MPI we used, MPICH, is a public domain, portable implementation. IBM has developed an experimental implementation, called MPI-F [1], which could have better performance.

In designing message passing libraries, two approaches have been taken in the past. The PVM approach provides simple point-to-point primitives. Not only is the implementation easier, but also users have less primitives to learn. If a user needs a more advanced operation (e.g., for

collective communication), he can always simulate it by using a number of point-to-point primitives. The MPI (and MPL) approach, on the other hand, provides collective primitives as part of the library. Our experiment show that the second approach is more advantageous in two aspects:

First, collective operations are frequently needed in parallel programs. Providing them directly through a library makes coding easier and less error-prone for the user. One should not, though, push this approach to the extreme by defining hundreds or thousands of functions as primitives. Overall, we think MPI and MPL achieved a balanced design.

Second, collective primitives have a performance edge over simulation through point-to-point primitives. As an example, consider broadcasting a 1-MB message on a 128-node system. Using the MPL broadcast primitive directly will take about 136,381 μs. However, if we want to simulate this collective communication operation with point-to-point send/receive, the best way is to use a fan-out tree algorithm in log128=7 steps, where each node performs two point-to-point communications per step. Each point-to-point communication of a 1MB message takes about 31,348 μs. The total time will then be 31348×14=438,872 μs, which is 2.22 times slower than the broadcast primitive.

The IBM SP2 is equipped with very powerful POWER2 processors. In contrast, the communication capability does not quite match the processing capability. This was also observed for other multicomputers such as Intel Paragon and Meiko CS-2 [3]. Listed in Table 11 are some performance data we have collected regarding the floating-point capability of one SP2 node. The Efficiency column shows the ratio of the real performance to the 266 MFLOPS peak performance, in percentage.

Let us consider the simplest communication operation: one task sends a 4B message to another task. By our measurement, such a point-to-point communication takes about 46 μs. Compared with applications with a slow MFLOPS rate (e.g., the pi program with 25 MFLOPS), a single communication operation takes all the time to perform 1,150 flops. Compared to fast

applications (e.g., beamforming with 199 MFLOPS), such a communication operation is equivalent to 9,154 flops. Thus SP2 is only suitable for coarse-grain parallel programs, where thousands or more computation operations are performed for each communication operation encountered.

Table 11. Performance of Some Computations on One SP2 Node

Computation	MFLOPS	Efficiency %
FFT	47	17.61
Householder Transform	80	30.08
Beamforming	199	74.81
Matrix Multiplication	106	39.85
Inner Product	96	36.09
Computation of π	25	9.40
Bubble Sorting	50	18.80
Search	73	27.44

We do not imply that the IBM SP2 has inferior communication support. In fact, the communication capability of SP2 compares quite favorably over those of other contemporary distributed memory multicomputers. For instance, consider just one parameter, the latency in point-to-point communication. The SP2 has a user-level latency of 39 μs according to IBM and 46 μs according to our measurement. The latency is 86 μs on a TMC CM-5, and 164 μs on an Intel Paragon.

What we would like to point out is that although remarkable advances have been made in the communication capability of multicomputers, the computational capability increased even faster, as shown in Table 12. The Cosmic Cube, representing the first generation multicomputers, had a 0.047 MFLOPS per-node peak performance. It had a latency of 2000-6000 μs and an asymptotic bandwidth of 0.25 MB/s for point-to-point communication. Compared to Cosmic Cube, the SP2 has improved the latency by 51-154 times and the bandwidth 142 times. However, the peak speed has increased 5,675 times! Therefore communication overhead is still a significant problem with today's multicomputers.

Table 12. Comparison between First-Generation and Current Multicomputers

Metric	Cosmic Cube	SP2	Improvement
Per Node Peak Speed (MFLOPS)	0.047	266	5,675
Latency (μs)	2,000-6,000	39-46	51-154
Bandwidth (MB/s)	0.25	35.54	142

Acknowledgments

We would like to thank David Martinez and Robert Bond at MIT Lincoln Laboratory for their support. We are especially grateful to Peggy Williams, Racine Arnowitz, Blaise Barney, Tim Fahey, George Gusciora, and Lon Waters at Maui High-Performance Computing Center for their help. Craig Stunkel of IBM provided the IBM latency and bandwidth data for point-to-point communication.

References

(1) H. Franke, P. Hochschild, P. Pattnail, J.P. Prost, and M. Snir, "MPI on IBM SP1/SP2: Current Status and Future Directions", (contact: frankeh@watson.ibm.com), 1993.

(2) A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.

(3) R. W. Hockney, "The Communication Challenge for MPP: Intel Paragon and Meiko CS-2", *Parallel Computing*, 1994, 20():389-398.

(4) K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, and Programmability*, McGraw-Hill, New York, 1993.

(5) K. Hwang, Z. Xu, and M. Arakawa, "STAP Benchmarking on IBM SP2 in MHPCC", *Tech. Report*, University of Southern California, Dept. of EE-Systems, Los Angeles, Jan. 1995.

(6) IBM Corp., *IBM AIX Parallel Environment: Operation and Use (Release 2.0)*, SH26-7230-01, June 1994.

(7) IBM Corp., *IBM AIX Parallel Environment: Parallel Programming Subroutine Reference (Release 2.0)*, SH26-7228-01, June 1994.

(8) W. Gropp and E. Lusk, "Some Early Performance Results with MPI on the IBM SP1", Argonne National Laboratory, August 1994.

(9) W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, MA, 1994.

(10) MPI Forum, "MPI: A Message-Passing Interface Standard", *International Journal of Supercomputer Applications*, 1994, 8(3/4).

(11) MHPCC, *MHPCC 400-Node SP2 Environment*, Maui High-Performance Computing Center, October 1994.

(12) C. B. Stunkel, Private Communication, August 1994.

(13) T. von Eicken, D.E. Culler, S.C. Goldstein, K.E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation", *Proc. of 19th Int'l. Symp. on Computer Architecture*, May 1992.

(14) Z. Xu and K. Hwang, "Computation and Communication Characteristics of the IBM SP2 Multicomputer System", *Tech. Report*, University of Southern California, Dept. of EE-Systems, Los Angeles, December 1994.

(15) Z. Xu and K. Hwang, "Performance Characterization of MPI Communication Functions on the Maui IBM SP2 Multicomputer System", *Tech. Report*, University of Southern California, Dept. of EE-Systems, Los Angeles, December 1994.