Performance Analysis of MPI Collective Operations *

Jelena Pješivac-Grbović[1], Thara Angskun[1], George Bosilca[1],
Graham E. Fagg[1], Edgar Gabriel[2], and Jack J. Dongarra[1]

[1] Innovative Computing Laboratory,
University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
{pjesa, angskun, bosilca, fagg, dongarra}@cs.utk.edu,
[2] Department of Computer Science, University of Houston,
501 Philip G. Hoffman Hall, Houston, TX 77204-3010
gabriel@cs.uh.edu

Abstract. Previous studies of application usage show that the performance of collective communications are critical for high-performance computing and are often overlooked when compared to the point-to-point performance. In this paper, we analyze and attempt to improve intra-cluster collective communication in the context of the widely deployed MPI programming paradigm by extending accepted models of point-to-point communication, such as Hockney, LogP/LogGP, and PLogP. The predictions from the models were compared to the experimentally gathered data and our findings were used to optimize the implementation of collective operations in the FT-MPI library. Additionally, we also introduce a new form of optimized tree-based Broadcast algorithm.

1 Introduction

Previous studies of application usage show that the performance of collective communications are critical to high-performance computing (*HPC*). Profiling study [1] showed that some applications spend more than eighty percent of a transfer time in collective operations. Given this fact, it is essential for MPI implementations to provide high-performance collective operations. However, collective operation performance is often overlooked when compared to the point-to-point performance. Collective operations (*collectives*) encompass a wide range of possible algorithms, topologies, and methods. The optimal[3] implementation of a collective for a given system depends on many factors, including for example, physical topology of the system, number of processes involved, message sizes, and the location of the root node (where applicable). Furthermore, many algorithms allow explicit segmentation of the message that is being transmitted, in which case the performance of the algorithm also depends on the used segment size. Some collective operations involve local computation (e.g. reduction operations), in which case the local characteristics of each node need to be considered as they could affect our decision on how to overlap communication with computation.

* This material is based upon work supported by the Department of Energy under Contract No. DE-FG02-02ER25536.

[3] We define "optimal implementation" in the following way: given a set of available algorithms for the collective, optimal implementation will use the best performing algorithm for the particular combination of parameters (message size, communicator size, root, etc.).

Simple, yet time consuming way to find even a semi-optimal implementation of an individual collective operation is to run an extensive set of tests over a parameter space for the collective on a dedicated system. However, running such detailed tests even on relatively small clusters (32 - 64 nodes), can take a substantial amount of time [2][4]. If one were to analyze all of the MPI collectives in a similar manner, the tuning process could take days. Still, many of current MPI implementations use "extensive" testing to determine switching points between the algorithms. The decision of which algorithm to use is semi-static and based on predetermined parameters that do not model all possible target systems.

Alternatives to the static decisions include running a limited number of performance and system evaluation tests. This information can be combined with predictions from parallel communication models to make run-time decisions to select near-optimal algorithms and segment sizes for given operation, communicator, message size, and the rank of the root process.

There are many parallel communication models that predict performance of any given collective operation based on standardized system parameters. Hockney [3], LogP [4], LogGP [5], and PLogP [6] models are frequently used to analyze parallel algorithm performance. Assessing the parameters for these models within local area network is relatively straightforward and the methods to approximate them have already been established and are well understood [7][6].

The major contribution of this paper is the direct comparison of Hockney, LogP, LogGP, and PLogP based parallel communication models applied to optimization of intra-cluster MPI collective operations. We quantitatively compare the predictions of the models to experimentally gathered data and use models to obtain optimal implementation of broadcast collective. We assess the performance penalty of using model generated decision functions versus the ones generated by exhaustive testing of the system. Indirectly, this work was used to implement and optimize the collective operation subsystem of the FT-MPI [8] library.

The rest of this paper proceeds as follows. Section 2 discusses related work. Section 3 provides background information on parallel communication models of interest; Section 4 discusses the Optimized Collective Communication (OCC) library and explains some of the algorithms it currently provides; Section 5 provides details about the collective algorithm modeling; Section 6 presents the experimental evaluation of our study; and Section 7 is discussion and future work.

2 Related work

Performance of MPI collective operations has been an active area of research in recent years. An important aspect of collective algorithm optimizations is understanding the algorithm performance in terms of different parallel communication models.

Thakur et al. [9] and Rabenseifner et al. [10] use Hockney model to analyze the performance of different collective operation algorithms. Kielmann et al. [11] use PLogP model to find optimal algorithm and parameters for topology-aware collective operations incorporated in the MagPIe library. Bell et al. [12]

[4] For example, profiling the linear scatter algorithm on 8 nodes took more than three hours[2].

use extensions of LogP and LogGP models to evaluate high performance networks. Bernaschi et al. [13] analyze the efficiency of reduce-scatter collective using LogGP model. Vadhiyar et al. [2] used a modified LogP model which took into account the number of pending requests that had been queued. Tables 1 through 4 point the reader to the relevant work related to the algorithms in question.

3 Background

Our work is built upon mathematical models of parallel communication. For better understanding of how we use these models we describe them in more detail below. Since MPI collective operations consist of communication and computation part of the algorithm, both network and computation aspects of the collective need to be modeled for any meaningful analysis.

3.1 Modeling network performance

In modeling communication aspects of collective algorithms, we employ the models most-frequently used by the message-passing community:

Hockney model. Hockney model [3] assumes that the time to send a message of size m between two nodes is $\alpha + \beta m$, where α is the latency for each message, and β is the transfer time per byte or reciprocal of network bandwidth. We altered Hockney model such that α and β are functions of message size. Congestion cannot be modeled using this model.

LogP/LogGP models. LogP model [4] describes a network in terms of latency, L, overhead, o, gap per message, g, and number of nodes involved in communication, P. The time to send a message between two nodes according to LogP model is $L + 2o$. LogP assumes that only constant-size, small messages are communicated between the nodes. In this model, the network allows transmission of at most $\lfloor L/g \rfloor$ messages simultaneously. LogGP [5] is an extension of the LogP model that additionally allows for large messages by introducing the gap per byte parameter, G. LogGP model predicts the time to send a message of size m between two nodes as $L + 2o + (m-1)G$. In both LogP and LogGP model, the sender is able to initiate a new message after time g.

PLogP model. PLogP model [6] is an extension of the LogP model. PLogP model is defined in terms of end-to-end latency L, sender and receiver overheads, $o_s(m)$ and $o_r(m)$ respectively, gap per message $g(m)$, and number of nodes involved in communication P. In this model sender and receiver overheads and gap per message depend on the message size. Notion of latency and gap in the PLogP model slightly differs from that of the LogP/LogGP model. Latency in the PLogP model includes all contributing factors, such as copying data to and from network interfaces, in addition to the message transfer time. Gap parameter in the PLogP model is defined as the minimum time interval between consecutive message transmissions or

receptions, implying that at all times $g(m) \geq o_s(m)$ and $g(m) \geq o_r(m)$. Time to send a message of size m between two nodes in the PLogP model is $L + g(m)$. If $g(m)$ is a linear function of message size m and L excludes the sender overhead, then the PLogP model is identical to LogGP model which distinguishes between sender and receiver overheads.

3.2 Modeling computation

We assume that the time spent in computation on data in a message of size m is γm, where γ is computation time per byte. This linear model ignores effects caused by memory access patterns and cache behavior, but is able to provide a lower limit on time spent in computation.

4 Optimized collective communication

We have developed a framework for functional method verification and performance testing known as the Optimized Collective Communication library (*OCC*). OCC is an MPI collective library built on top of point-to-point operations. OCC consists of three modules: methods, verification, and performance-testing modules. The methods module provides a simple interface for addition of new collective algorithms. The verification module provides basic verification tools for the existing methods. The performance module provides set of micro-benchmarks for the library. A method is defined by an algorithm and parameters it needs, such as virtual topology and segment size[5]. Currently, the methods module contains various implementations of the following subset of MPI collective operations: MPI_Barrier, MPI_Bcast, MPI_Reduce, MPI_Scatter, and MPI_Alltoall. These particular routines were chosen as representative of the commonly used collective operations in MPI programs [1].

4.1 Virtual topologies

MPI collective operations can be classified as either one-to-many/many-to-one (single producer or consumer) or many-to-many (every participant is both producer and consumer) operations. For example, Broadcast, Reduce, Scatter(v), and Gather(v) follow the one-to-many communication pattern, while Barrier, Alltoall, Allreduce, and Allgather(v) employ many-to-many communication patterns.

Generalized version of the one-to-many/many-to-one type of collectives can be expressed as *i)* receive data from preceding node(s), *ii)* process data, if required, *iii)* send data to succeeding node(s). The data flow for this type of algorithm is unidirectional. Virtual topologies can be used to determine the preceding and succeeding nodes in the algorithm.

Currently, the OCC library supports five different virtual topologies: flat-tree(linear,) pipeline (single chain), binomial tree, binary tree, and k-chain tree. Our experiments show that given a collective operation,

[5] Even though the definition of method is precise, in this paper, we will sometimes refer to method as algorithm: instead of referring to "generalized broadcast method with binary topology and 32KB segments," we may abbreviate long name to "binary algorithm with 32KB segments"

message size, and number of processes, each of the topologies can be beneficial for some combination of input parameters.

4.2 Available algorithms

This section describes the currently available algorithms in OCC for barrier, broadcast, reduce and alltoall operations. Due to space constraints and since it is outside the scope of this paper, we will not discuss the algorithms in great details.

Barrier. Barrier is a collective operation used to synchronize a group of nodes. It guarantees that by the end of the operation, the remaining nodes have at least entered the barrier. We implemented four different algorithms for the Barrier collective: flat-tree/linear fan-in-fan-out, double ring, recursive doubling, and Bruck [14] algorithm. In flat-tree/linear fan-in-fan-out algorithm all nodes report to a preselected root; once every node has has reported to the root, the root sends a releasing message to all participants. In the double ring algorithm, a zero-byte message is sent from a preselected root circularly to the right. A node can leave Barrier only after it receives the message for the second time. Both linear and double ring algorithms require $O(P)$ communication steps. Bruck algorithm requires $\lceil log_2 P \rceil$ communication steps. At step k, node r receives a zero-byte message from and sends message to node $(r-2^k)$ and $(r+2^k)$ node (with wrap around) respectively. The recursive doubling algorithm requires $log_2 P$ steps if P is a power of 2, and $\lfloor log_2 P \rfloor + 2$ steps if not. At step k, node r exchanges message with node (r XOR 2^k). If the number of nodes P is not a power 2, we need two extra steps to handle remaining nodes.

Broadcast. Broadcast operation transmits an identical message from the root process to all processes of the group. At the end of the call, the contents of the root's communication buffer is copied to all other processes. We implemented the following algorithms for this collective: flat-tree/linear, pipeline, binomial tree, binary tree, and splitted-binary tree. In flat-tree/linear algorithm root node sends an individual message to all participating nodes. In pipeline algorithm, messages are propagated from the root left to right in a linear fashion. In binomial and binary tree algorithms, messages traverse the tree starting from the root, and going towards the leaf nodes through intermediate nodes. In the splitted-binary tree algorithm[6], the original message is split into two parts, and the "left" half of the message is sent down the left half of the binary tree, and the "right" half of the message is sent down the right half of the tree. In the final phase of the algorithm, every node exchanges message with their "pair" from the opposite side of the binary tree. In the case when the tree has even number of nodes, the leaf without the pairwise partner, receives the second half of the message from the root. All of the broadcast algorithms allow for message segmentation which potentially allows for overlap of concurrent communications.

[6] To the best of our knowledge, no other group implemented or discussed this algorithm so far. *Or, After an extensive search of the literature, no other form of this algorithm has been located*

Reduce. Reduce operation combines elements provided in the input buffer of each process within a group using the specified operation, and returns the combined value in the output buffer of the root process. We have implemented generalized Reduce operation that can use all available virtual topologies: flat-tree/linear, pipeline, binomial tree, binary tree, and k-chain tree. At this time, OCC library works only with the predefined MPI operations. As in the case of Broadcast, our actual implementation overlaps multiple communications with computation.

Alltoall. Alltoall is used to exchange data among all processes in a group. The operation is equivalent to all processes executing the scatter operation on their local buffer. We have implemented linear and pairwise exchange algorithms for this collective. In the linear alltoall algorithm at step i, the i^{th} node sends a message to all other nodes. The $(i+1)^{th}$ node is able to proceed and start sending as soon as it receives the complete message from the i^{th} node. We allow for segmentation of messages being sent. In the pairwise exchange algorithm, at step i, node with rank r sends a message to node $(r+i)$ and receives a message from the $(r-i)^{th}$ node, with wrap around. We do not segment messages in this algorithm. At any given step in this algorithm, a single incoming and outgoing communication exists at every node.

5 Modeling collective operations

For each of the implemented algorithms we have created a numeric reference model based on a point-to-point communication models previously discussed in Section 3. We assume a full-duplex network which allows us to exchange and send-receive a message in the same amount of time as completing a single receive.

Tables 1, 2, 3, and 4 show formulas for Barrier, Broadcast, Reduce, and Alltoall collectives respectively. If applicable, the displayed formulas account for message segmentation. Message segmentation allows us to divide a message of size m into a number of segments, n_s, of segment size m_s. In the Hockney and PLogP models parameter values depend on the message size. The LogP formulas can be obtained from LogGP by setting the gap per byte parameter, G to zero. The specified tables also provide references to relevant and similar work done by other groups.

The model of the flat-tree barrier algorithm performance in Table 1 requires additional explanation. The conservative model of flat-tree barrier algorithm would include time to receive (P-1) messages sent in parallel to the same node, and the time to send (P-1) messages from the root. In the first phase, root process posts (P-1) non-blocking receives followed by single waitall call. Our experiments show that on our systems, all MPI implementations we examined were able to deliver (P-1) zero-byte messages sent in parallel to the root in close to the time to deliver a single message. Thus we model the total duration of this algorithm as the time it takes to receive a single zero-byte message plus the time to send (P-1) zero-byte messages.

Barrier	Model	Duration
Flat-Tree	*Hockney*	$T = (P-1) \times \alpha$
Flat-Tree	*LogP/LogGP*	$T_{min} = (P-2) \times g + 2 \times (L + 2 \times o)$ $T_{max} = (P-2) \times (g+o) + 2 \times (L + 2 \times o)$
Flat-Tree	*PLogP*	$T_{min} = P \times g + 2 \times L$ $T_{max} = P \times (g + o_r) + 2 \times (L - o_r)$
Double Ring	*Hockney*	$T = 2 \times P \times \alpha$
Double Ring	*LogP/LogGP*	$T = 2 \times P \times (L + o + g)$
Double Ring	*PLogP*	$T = 2 \times P \times (L + g)$
Recursive Doubling	*Hockney*	$T = \log_2(P) \times \alpha$, if P is exact power of 2 $T = (\log_2(P) + 2) \times \alpha$, otherwise
Recursive Doubling	*LogP/LogGP*	$T = \log_2(P) \times (L + o + g)$, if P is exact power of 2 $T = (\lfloor \log_2(P) \rfloor + 2) \times (L + o + g)$, otherwise
Recursive Doubling	*PLogP*	$T = \log_2(P) \times (L + g)$, if P is exact power of 2 $T = (\lfloor \log_2(P) \rfloor + 2) \times (L + g)$, otherwise
Bruck	*Hockney*	$T = \lceil \log_2(P) \rceil \times \alpha$
Bruck	*LogP/LogGP*	$T = \lceil \log_2(P) \rceil \times (L + o + g)$
Bruck	*PLogP*	$T = \lceil \log_2(P) \rceil \times (L + g)$

Table 1. Analysis of different Barrier algorithms.

6 Results and analysis

6.1 Experiment setup

The measurements were obtained on several dedicated clusters provided by the SInRG project at the University of Tennessee at Knoxville. The first cluster, Boba, consists of 32 Dell Precision 530s nodes, each with Dual Pentium IV Xeon 2.4 GHz processors, 512 KB Cache, 2 GB Ram, connected via Gigabit Ethernet. The second cluster, Frodo, consist of 32 nodes, each containing dual Opteron processor, 2 GB Ram, connected via 100 Mbps Ethernet and Myrinet. In the results presented in this paper we utilized only Ethernet interconnect on the Frodo cluster.

Model parameters. We measured the model parameters using various MPI implementations. Most of the collected data was generated using FT-MPI [8], MPICH-1.2.6, and MPICH-2.0.97 [17]. Parameter values measured using MPICH-1 had higher latency and gap values with lower bandwidth than both FT-MPI and MPICH-2. FT-MPI and MPICH-2 had similar values for these parameters on both systems.

Hockney model parameters were measured directly using point-to-point tests.

To measure PLogP model parameters we used the `logp_mpi` software suite provided by Kielmann et al. [6]. Measured parameter values were obtained by averaging the values obtained between different communication points in the same system. For this model we also experimented with directly fitting model parameters to the experimental data, and applying those parameter values to model other collective operations. Parameter fitting was done under the assumption that the sender and receiver overheads do not depend on the network behavior, and as such we used values measured by the `log_mpi` library. In this paper, we obtained fitted PLogP parameters by analyzing the performance of the non-segmented pipelined broadcast and flat-tree barrier algorithm over various communicator and message sizes. We chose to fit

Broadcast	Model	Duration	Related work
Linear	*Hockney*	$T = n_s \cdot (P-1) \cdot (\alpha(m_s) + m_s \cdot \beta(m_s))$	[9], [15]
Linear	*LogP/LogGP*	$T = L + 2 \cdot o - g + n_s \times (P-1) \times ((m_s-1)G + g)$	
Linear	*PLogP*	$T = L + n_s \cdot (P-1) \cdot g(m_s)$	[16]
Pipeline	*Hockney*	$T = (P + n_s - 2) \times (\alpha(m_s) + m_s \cdot \beta(m_s))$	
Pipeline	*LogP/LogGP*	$T = \begin{array}{l}(P-1) \times (L + 2 \cdot o + (m_s-1)G) + \\ (n_s-1) \times (g + (m_s-1)G)\end{array}$	
Pipeline	*PLogP*	$T = (P-1) \times (L + g(m_s)) + (n_s-1) \times g(m_s)$	
Binomial	*Hockney*	$T = \lceil log_2(P) \rceil \times n_s \times (\alpha(m_s) + m_s \cdot \beta(m_s))$	[9], [15]
Binomial	*LogP/LogGP*	$T = \lceil log_2(P) \rceil \times \begin{pmatrix} L + 2 \cdot o + (m_s-1)G + \\ (n_s-1) \times (g + (m_s-1)G) \end{pmatrix}$	[4], [5]
Binomial	*PLogP*	$T = \lceil log_2(P) \rceil \times (L + n_s \times g(m_s))$	[16]
Binary	*Hockney*	$T = (\lceil log_2(P+1) \rceil + n_s - 2) \times (2 \times \alpha(m_s) + m_s \cdot \beta(m_s))$	
Binary	*LogP/LogGP*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil - 1) \times (L + 2 \times (o + (m_s-1)G + g)) + \\ 2 \times ((m_s-1)G + g)\end{array}$	[4], [5]
Binary	*PLogP*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil - 1) \cdot (L + 2 \cdot g(m_s)) + \\ (n_s-1) \times max\{2 \cdot g(m_s), o_r(m_s) + g(m_s) + o_s(m_s)\}\end{array}$	[16]
Splitted-Binary	*Hockney*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil + \lceil \frac{n_s}{2} \rceil - 2) \times (2 \times \alpha(m_s) + m_s \cdot \beta(m_s)) + \\ \alpha(\frac{m}{2}) + \frac{m}{2} \cdot \beta(\frac{m}{2})\end{array}$	
Splitted-Binary	*LogP/LogGP*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil - 1) \times (L + g + 2 \cdot (o + (m_s-1)G)) + \\ 2 \times (\lceil \frac{n_s}{2} \rceil - 1) \times (g + (m_s-1)G) + \\ L + 2 \cdot o + (\frac{m}{2} - 1)G\end{array}$	
Splitted-Binary	*PLogP*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil - 1) \times (L + 2 \cdot g(m_s)) + \\ (\frac{n_s}{2} - 1) \cdot max\{2 \cdot g(m_s), o_r(m_s) + g(m_s) + o_s(m_s)\}\end{array}$	

Table 2. Analysis of different Broadcast algorithms.

model parameters to these algorithms as the communication pattern of non-segmented pipelined broadcast's data algorithm (linear sending and receiving message) is the closest match to the point-to-point tests used to measure model parameters in the `logp_mpi` and similar libraries. At the same time, flat-tree barrier formulas in Table 1 provide the most direct way of computing the gap per message parameter for zero-byte messages for PLogP and LogP/LogGP models. Results obtained using these values matched more closely the overall experimental data, thus all PLogP model results in this paper were obtained using fitted parameters.

Values of LogP and LogGP were obtained from the fitted PLogP values as explained by Kielmann et al. in [6].

Figure 1 shows parameter values for Hockney and PLogP models on both clusters. Table 5 summarizes the parameter values for LogP/LogGP model.

Performance tests. Our performance measuring methodology follows the recommendations given by Gropp et al. in [18] to ensure the reproducibility of the measured results. We minimize the effects of pipelining by forcing a "report-to-root" step after each collective operation. Each of the collected data points is a minimum value of 10-20 measurements in which the maximum value is excluded, and the standard deviation was less than 5% of the remaining points.

Reduce	Model	Duration	Related work
Flat Tree	*Hockney*	$T = n_s \times (P-1) \times (\alpha + \beta m_s + \gamma m_s)$	[9], [15]
Flat Tree	*LogP/LogGP*	$T = \begin{array}{l} o + (m_s-1)G + L + \\ n_s \times \max\{g, (P-1) \times (o + (m_s-1)G + \gamma m_s)\} \end{array}$	
Flat Tree	*PLogP*	$T = L + (P-1) \times n_s \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}$	[16]
Pipeline	*Hockney*	$T = (P + n_s - 2) \times (\alpha + \beta m_s + \gamma m_s)$	
Pipeline	*LogP/LogGP*	$T = \begin{array}{l} (P-1) \times (L + 2 \times o + (m_s-1)G + \gamma m_s) + \\ (n_s-1) \times \max\{g, 2 \times o + (m_s-1)G + \gamma m_s\} \end{array}$	
Pipeline	*PLogP*	$T = \begin{array}{l} (P-1) \times (L + \max\{g(m_s), o_r(m_s) + \gamma m_s\}) + \\ (n_s-1) \times (\max\{g(m_s), o_r(m_s) + \gamma m_s\} + o_s(m_s)) \end{array}$	
Binomial	*Hockney*	$T = n_s \times \lceil log_2(P) \rceil \times (\alpha + \beta m_s + \gamma m_s)$	[9], [15]
Binomial	*LogP/LogGP*	$T = \lceil log_2 P \rceil \times \left(\begin{array}{l} (n_s-1) \times \max\{(m_s-1)G + g, o + \gamma m_s\} + \\ o + L + \max\{(m_s-1)G, \gamma m_s\} \end{array} \right)$	[4], [5]
Binomial	*PLogP*	$T = \lceil log_2 P \rceil \times (L + n_s \times \max\{g(m_s), o_r(m_s) + \gamma m_s\})$	[16]
Binary	*Hockney*	$T = 2 \cdot (\lceil \log_2(P+1) \rceil + n_s - 2) \times (\alpha + \beta m_s + \gamma m_S)$	[9], [15]
Binary	*LogP/LogGP*	$T = \begin{array}{l} (\lceil log_2(P+1) \rceil - 1) \times ((L + 3 \times o + (m_s-1)G + 2\gamma m_s) + \\ (n_s-1) \times ((m_s-1)G + \max\{g, 3o + 2 \times \gamma m_s\})) \end{array}$	[4], [5]
Binary	*PLogP*	$T = \begin{array}{l} (\lceil log_2(P+1) \rceil - 1) \times (L + 2 \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}) + \\ (n_s-1) \times (o_s(m_s) + 2 \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}) \end{array}$	[16]

Table 3. Analysis of different Reduce algorithms.

Alltoall	Model	Duration	Related work
Linear	*Hockney*	$T = \begin{array}{l} P \times (\alpha + \beta m_s) + \\ (P-1) \times (n_s \times P + 1 - \frac{P}{2}) \times \alpha \end{array}$	[9]
Linear	*LogP/LogGP*	$T = \begin{array}{l} P \times (L + 2 \times o) + \\ (P-1) \times (n_s \times P + 1 - \frac{P}{2}) \times (g + (m_s-1)G) \end{array}$	[4]
Linear	*PLogP*	$T = P \times L + (P-1) \times (n_s \times P + 1 - \frac{P}{2}) \times g(m_s)$	
Pairwise exchange	*Hockney*	$T = (P-1) \times (\alpha + \beta m)$	[9]
Pairwise exchange	*LogP/LogGP*	$T = (P-1) \times (L + o + (m-1) \times G + g)$	
Pairwise exchange	*PLogP*	$T = (P-1) \times (L + g(m))$	

Table 4. Analysis of different Alltoall algorithms.

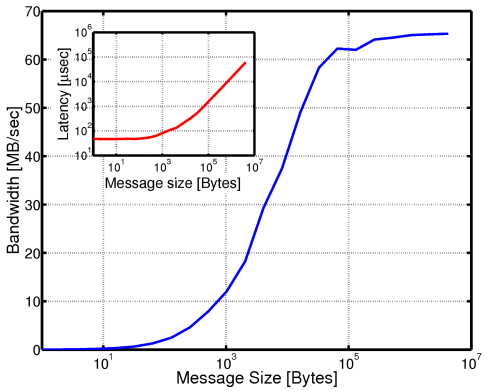
LogP/LogGP		***Boba cluster***	***Frodo cluster***
Latency	L	$30.45\ [\mu sec]$	$61.22\ [\mu sec]$
Overhead	o	$8.15\ [\mu sec]$	$8.2\ [\mu sec]$
Gap	g	$8.683\ [\mu sec]$	$23.8\ [\mu sec]$:
Gap-per-byte	G	$0.015\ [\frac{\mu sec}{byte}]$	$0.084\ [\frac{\mu sec}{byte}]$

Table 5. LogP/LogGP model parameters on both clusters.

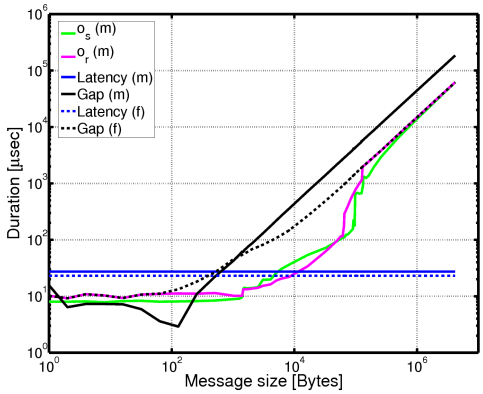
6.2 Empirical results and analysis

We executed performance tests for various Barrier, Broadcast, Reduce, Scatter, and Alltoall collective operations implementations using FT-MPI, MPICH-1, and MPICH-2. We then analyzed the algorithm performance and the optimal implementation of various collective operations. When predicting performance of collective operations that exchanged actual data (message size > 0) we did not consider pure LogP predictions, but used LogGP instead.

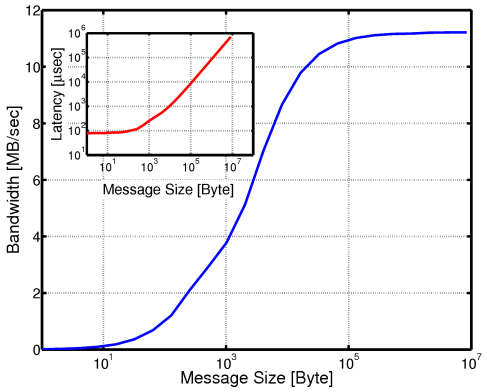
We found that the worst case for an algorithm performance is often too pessimistic, as in the case of the flat-tree/linear fan-in-fan-out barrier algorithm. Our experience with the MPI implementations was that



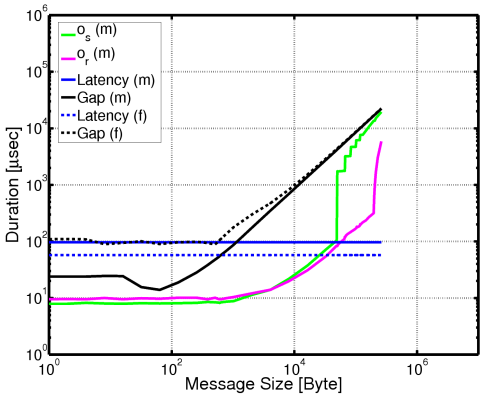
(a) Hockney, Boba



(b) PLogP, Boba



(c) Hockney, Frodo



(d) PLogP, Frodo

Fig. 1. Hockney and PLogP parameter values on the Boba and Frodo clusters. The Boba cluster utilized GigE interconnect, while the Frodo we utilized 100 Mbps Ethernet. On PLogP parameter graphs (b) and (d), (m) denotes measured values while (f) denotes fitted values of gap and latency.

the algorithms performance was generally closer to the best case scenario. Thus, where possible we chose to model algorithm performance using the best case scenario.

Barrier performance. Figure 2 illustrates measured and predicted performance of bruck, recursive doubling, and linear fan-in-fan-out barrier algorithms on Boba cluster (GigE).

Experimental data for both bruck and recursive doubling algorithms, while exhibiting trends, is not uniform. One of the possible explanations for this is due to their logarithmic behavior, the "report-to-root" step in the performance measurement procedure takes comparable amount of time to the duration of these algorithms. Thus, variations in the time to perform "report-to-root" on a given communicator could affect the measured value for these barrier methods more significantly than for other collective operations which take proportionally more time. The flat-tree/linear fan-in-fan-out barrier which takes slightly longer

to complete and has a more regular communication pattern does not exhibit this problem. The second possible explanation could lie in the irregular communication pattern of both methods which may cause some extra overhead in underlying run-time communication libraries.

The measured data for the flat-tree barrier algorithm displays some unexpected behavior. Based on the PLogP and LogP/LogGP models of performance showed in Table 1, the duration of this algorithm grows linearly with communicator size and the slope of the line is equal to the zero-byte gap. However, the experimental data implies that the slope decreases around 16 nodes. The results shown in Figure 2 were generated using MPICH2, but the behavior was consistent also when using FT-MPI on the Frodo system, as we will see in the case of Broadcast decision function. This implies that the underlying system (either MPI library, TCP/IP, or hardware) were able to further optimize communication when sending and receiving zero-byte messages to multiple nodes. Since the Hockney model assumes that the minimum time between sending two messages is equal to the latency, the prediction for this model for flat-tree barrier is largely overestimated.

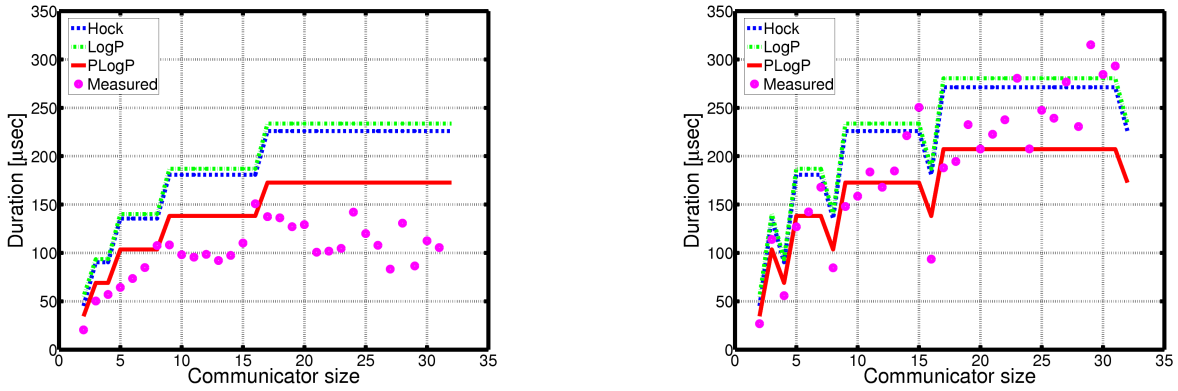
However, even accounting for all known discrepancies, all the models captured relative performance of these barrier algorithms sufficiently correctly.

Reduce performance. Considering one-to-many and many-to-one style of collectives which exchange data, such as Reduce, our experiments show that for sufficiently large messages, segmenting message into fixed-size chunks can improve performance. Some of the benefits of segmentation include increased number of concurrent messages which allows us to utilize bandwidth of the system more efficiently; ability to overlap multiple communications and computation; and limiting the size of internal buffers required by the algorithm.

Figure 3 displays measured and predicted performance of non-segmented and segmented versions of binomial and pipeline reduce algorithms for two communicator sizes on the Boba cluster.

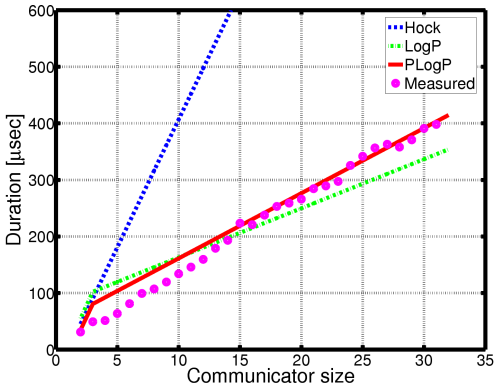
For small messages (less than 1KB,) all models were able to capture relative performance as well as satisfyingly predict absolute performance for these algorithms.

For larger messages (involving multiple segments,) some of the model limitations were exposed. The models of binomial and pipeline algorithms presented in Table 3 indicate that the segmenting of messages will have different effect on the performance of these algorithms. Segmenting of messages in binomial reduce algorithm even with proper communication-computation overlap should increase running time as the number of root's children in the binomial tree with P nodes is proportional to $log_2(P)$. Contrary to this, the formulas for modeling segmented pipelined reduce given in Table 3 indicate that as the number of segments n_s increases, the term that depends on number of processes P becomes less significant. In the asymptotic case, the segmented pipeline reduce takes a constant time for a given message size (m) and number of segments (n_s) regardless of any additional increase in the number of processes (P). The experimental data for binomial reduce algorithm shown in Figure 3 (a) and (b) disagree with the expected models: on both eight and twenty-four nodes, when the message size is greater than approximately 100KB



(a) Bruck

(b) Recursive Doubling



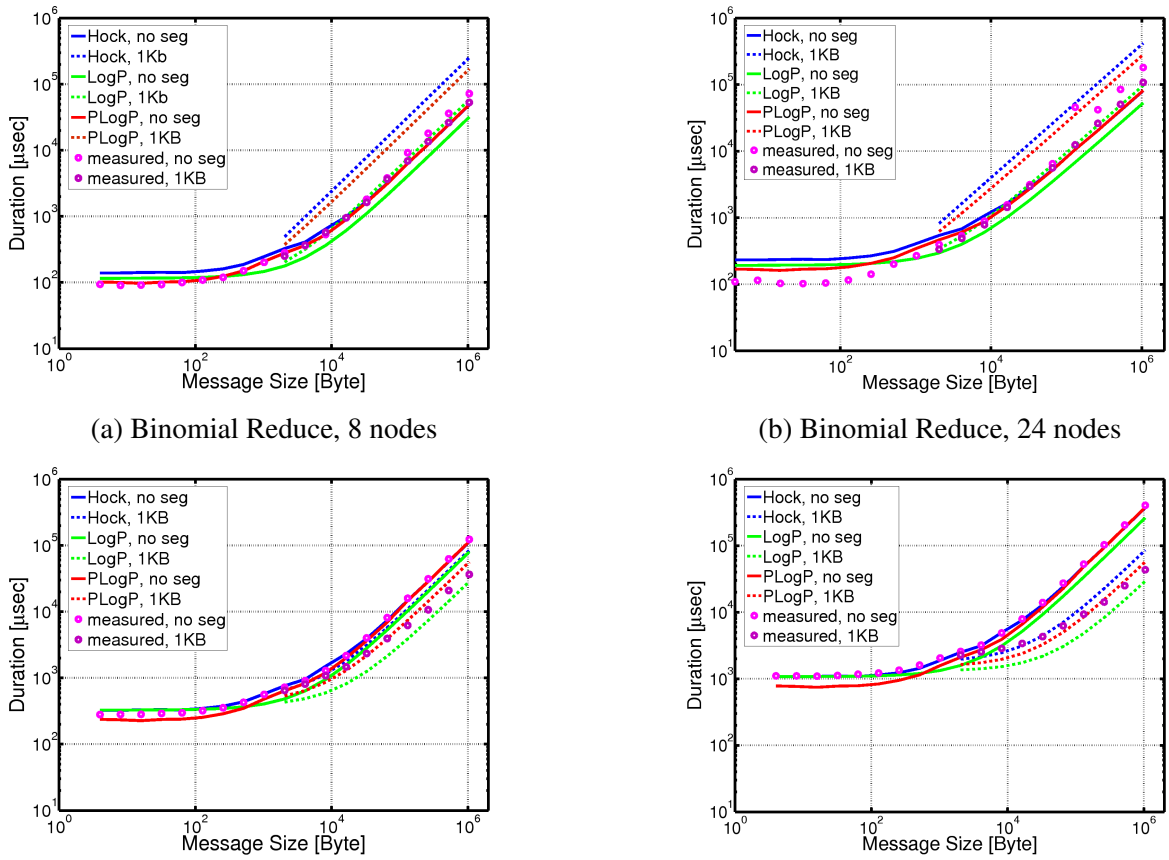
(b) Flat-tree

Fig. 2. Performance of Barrier algorithms: Experimentally measured values are indicated by circles. (MPICH-2, Boba cluster, GigE).

segmenting message into 1KB segments improves binomial reduce performance. However, the benefit is more noticeable on 24 nodes. There are two possible explanations for this behavior which cannot be captured with our current models. First, as we observed in the case of the flat-tree barrier algorithm, the gap between messages in PLogP and LogP/LogGP models depends on number of nodes we are communicating with, and for communicator sizes greater than sixteen nodes it decreased in comparison to smaller communicator sizes. Secondly, MPI libraries in our experiments used the TCP/IP stack. The TCP window size on our systems is 128KB. This means that sending messages larger than the TCP window will require resizing the window and an extra memory copy operation per pair of communicating parties (which in this case is $log_2(P)$ times). Both of the reasons would increase cost of non-segmented method while keeping the cost of the segmented method unchanged.

On the other hand, the measured performance of the segmented pipeline reduce algorithm agrees with our models very well. The asymptotic behavior is visible on both eight and twenty-four nodes as the

time it takes to reduce $1MB$ message is around 40msec in both cases. The version of the method without segmentation is accurately modeled using PLogP and Hockney models, and somewhat less accurately with LogP/LogGP model. Never the less, all three models are able to capture general performance. The duration of the pipelined reduce algorithm with segments of size 1KB are most accurately modeled using the PLogP model. Hockney model overestimates the measured values, meanwhile the LogP/LogGP underestimates them respectively.



(a) Binomial Reduce, 8 nodes

(b) Binomial Reduce, 24 nodes

(c) Pipeline Reduce, 8 nodes

(d) Pipeline Reduce, 24 nodes

Fig. 3. Performance of Segmented Binomial and Pipelined Reduce methods on 8 and 24 nodes. Fitted parameter values were used to make predictions for LogP/LogGP and PLogP models (MPICH-2, Boba cluster, GigE).

Alltoall performance. Figure 4 demonstrates the performance of the pairwise-exchange alltoall algorithm. The alltoall type of collectives can cause network flooding even when we attempt to carefully schedule communication between the nodes. Hockney model does not have the notion of network con-

gestion and this is one of the possible reasons why it significantly underestimates the completion time of collective operation. While we did not explicitly include a congestion component in the PLogP and LogGP model formulas, they were able to predict measured performance with reasonable accuracy.

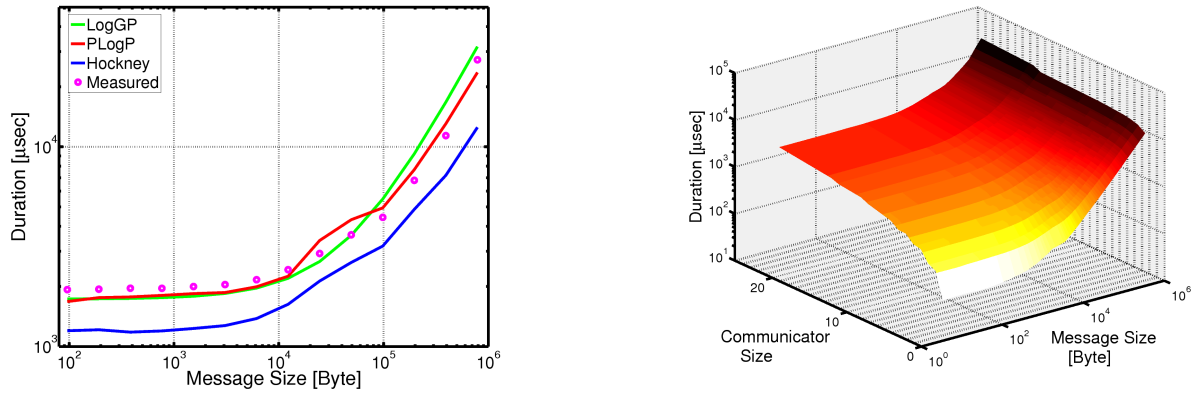


Fig. 4. Performance of Pairwise Exchange Alltoall algorithm: (a) Measured performance and predictions for 24 nodes, and (b) Measured performance on 2 to 24 nodes. The message size represents the total send buffer size (FT-MPI, Boba cluster, GigE).

Optimal broadcast implementation. Figure 5 shows the optimal implementation of the broadcast collective using measured data and model predictions for the Frodo cluster. The optimal implementation of the collective is described by a *decision function*. Given the collective operation, message and communicator size, the decision function determines which algorithm, topology, and segment size combination should be used.

The measured decision function was derived from exhaustive testing on the Frodo cluster. We considered sample message sizes from 1 byte up to 8 Gigabytes and every communicator size from 3 to 32 nodes. We examined linear, binomial, binary, splitted-binary, and pipeline algorithms with and without segmentation, with segment sizes of 1KB and 8KB. The model decision functions were computed by analyzing predicted performance of the measured methods on the identical message and communicator sizes. Then the best method according to the model was chosen, and the model decision function was constructed.

Examining the optimal measured broadcast method for small messages and larger communicator sizes (above 16 nodes) we observe that the non-segmented linear algorithm is the best option. Contrary to this, for smaller communicator sizes and small messages non-segmented binomial algorithm executed in the least time. Our only explanation for this behavior is that by taking into the account change in the gap per message parameter when communicating to more than 16 nodes. Not surprisingly, all models mispredict

the optimal method for that part of the parameter space. For message sizes close to 1KB measured data suggests that all tree-based non-segmented algorithms can be optimal, ie. binomial, binary, and splitted binary trees. Once the message size increases to a couple of kilobytes, splitted-binary method with 1KB segments outperforms the other two algorithms, and for large message sizes segmented pipeline methods dominate. It is important to notice that the switching points between methods for large message sizes appears to depend on communicator size.

Hockney model broadcast decision function, Figure 5 (b), reflects the fact that in the Hockney model we must wait a full latency before being able to send another message. For small messages, binomial tree algorithm is the algorithm of choice for all communicator sizes. Except for a message size range around 10KB where the splitted-binary method with 1KB segments is optimal, 8KB segment is used for sending larger messages either using splitted-binary or pipeline method.

The LogP/LogGP model broadcast decision function utilizes non-segmented versions of linear, binomial, and binary algorithms for small messages. For intermediate size messages, depending on communicator size, either splitted-binary with 1KB segments or pipeline with 1KB segments method should be used. For really large messages, pipeline with 8KB segments is the best performing method. While this captures the general shape of the measured decision function, the points at which we switch from 1KB to 8KB segments differ. The LogP/LogGP decision function switches "too early."

The PLogP model broadcast decision function uses non-segmented binomial method for small message sizes. This is the only model decision function which recognizes that the binary algorithm with 1KB segments can be beneficial for intermediate size messages. For larger messages, as in the case with the LogP/LogGP model and measured decision function, it utilizes splitted-binary algorithm with 1KB segments, followed by segmented pipeline with 1KB and 8KB segment sizes. However, the PLogP decision function switches from splitted-binary to pipeline and between 1KB and 8KB segments even "earlier" than the LogP/LogGP decision function.

Deciding the correct switching point is ultimately related to understanding the exact behavior of the gap parameter in the underlying model, as gap determines whether it will be more cost effective to have a longer pipeline or a wider tree.

Given the limitations of our models, it is reasonable to ask how useful are their predictions in building decision functions for real collective implementation. Additionally, what is the performance penalty the user will pay from using the model generated decision function instead of using a measured one? Figure 6 addresses this question. The performance penalty for not using the linear algorithm for broadcasting small messages on 16 through 32 nodes is largest with more than 300% performance penalty. For small numbers of nodes with small messages, Hockney and PLogP vary between 0% and 15% performance degradation, except in case when communicator size is 5. For messages of intermediate size (up to 10KB) the model decision functions pay a performance penalty between 0% and 50%, with Hockney model decision performing worst. For larger messages the performance penalty of LogP/LogGP decision function for mispredicted switching points does not go above 25%. But the PLogP decision function does pay higher

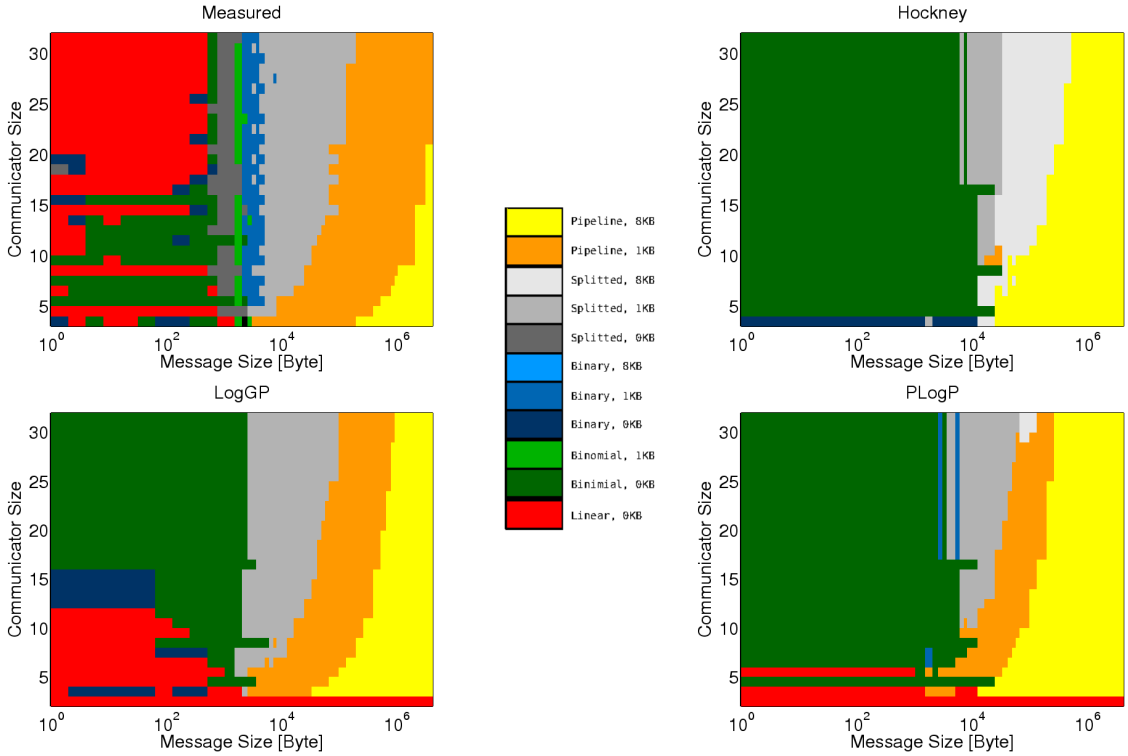
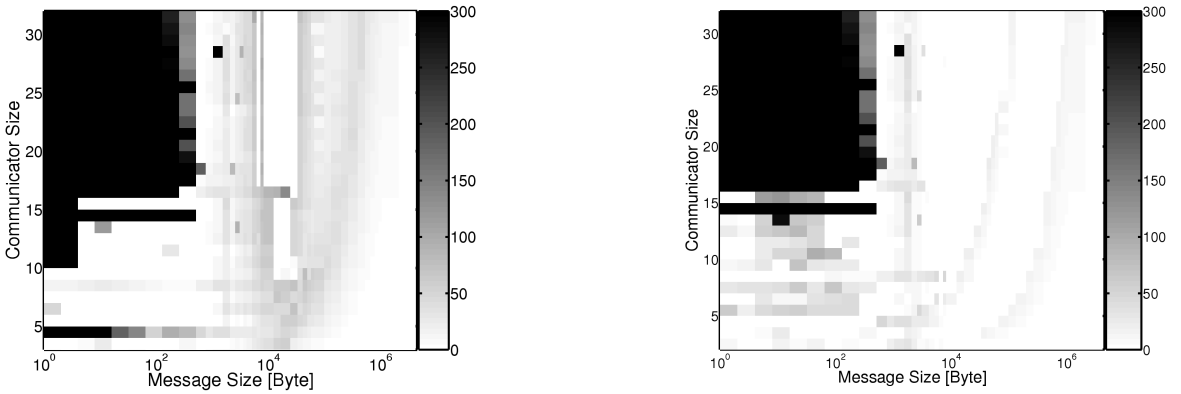


Fig. 5. Broadcast decision function. Graphs in this figure should be read in the following way: the color at point (m, P) represents the best broadcast method for message size m and communicator size P. Label with 0KB segment size denotes a non-segmented version of the algorithm. (FT-MPI, Frodo cluster, 100Mbps).

performance penalty (up to 50% for bordering points) for it switches algorithms even earlier. The fact that Hockney model would utilize splitted-binary broadcast algorithm with 8KB segments over the pipeline algorithm with 1KB segments would cost around 30% in performance over that part of parameter space. Still, one needs to be careful when interpreting the relative performance of decision functions, since the measured performance in this case was only result of a micro-benchmark. Individual, real-application performance and performance loss or gain could vary greatly depending on applications.

7 Discussion and future work

We compared the Hockney, LogP, LogGP, and PLogP parallel communication models applied to inter-cluster MPI collective operations on two systems at University of Tennessee. We showed that even when we do not model network congestion directly, all of the models can provide useful insights into various aspects of the different algorithms and their relative performance. We also demonstrated importance of accurate modeling of the gap parameter between sending two consecutive messages to a single destination and to a set of different destination processes. Unfortunately, neither of models was able to completely



(a) Hockney

(b) LogP/LogGP

(b) PLogP

Fig. 6. Performance penalty from using decision functions generated by models. Graphs in this figure should be read in the following way: the shade at point (m, P) represents the percent of the relative performance cost. The colorbar at the right of every graph shows the percentage range: from 0 to 300%. (FT-MPI, Frodo cluster, 100Mbps)

accurately describe this parameter. This shortcoming was reflected in the inaccurate prediction of switching points between available broadcast methods for large messages.

This work was used to implement and optimize the collective operation subsystem of the FT-MPI library by changing the static method-selecting decision function, but can be used as a library for any MPI implementation. For example, this work is currently being used to produce a new tuned collective module in the open source OpenMPI Implementation [19]. In FT-MPI experimental and analytical analysis of collective algorithm performance was used to determine switching points between available methods. At run time, based on a static table of values, a particular method is selected depending on the number of processes in the communicator, message size, and the rank of the root process.

Possible application of the models is to help avoid exhaustive testing of a particular system, as one can use the information generated by the models to narrow down the number of physical tests that need to be executed so we could perform focused tuning of the collectives.

We plan to extend this study in the following directions: addition of new algorithms and collective operations to the OCC library; making the algorithm selection process at run-time fully automated rather than hard-coded at compile time[7]; and building decision function refinement capability which would use parallel computation model decision function as a starting point to generate a list of physical tests to be executed on a given system.

Additionally, this analysis can be extended to hierarchical systems consisting of multiple clusters. In order to model performance of collective operations in such environments, we would have to include more details about the underlying network topology.

8 Acknowledgments

The infrastructure used in this work was supported by the NSF CISE Research Infrastructure program, EIA-9972889.

References

1. Rolf Rabenseifner. Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512. In *Proceedings of the Message Passing Interface Developer's and User's Conference*, pages 77–85, 1999.
2. Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra. Automatically tuned collective communications. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 3. IEEE Computer Society, 2000.
3. R.W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3):389–398, March 1994.
4. David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 1–12. ACM Press, 1993.
5. Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating long messages into the LogP model. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 95–105. ACM Press, 1995.
6. T. Kielmann, H.E. Bal, and K. Verstoep. Fast measurement of LogP parameters for message passing platforms. In José D. P. Rolim, editor, *IPDPS Workshops*, volume 1800 of *Lecture Notes in Computer Science*, pages 1176–1183, Cancun, Mexico, May 2000. Springer-Verlag.
7. D. Culler, L. T. Liu, R. P. Martin, and C. Yoshikawa. Assessing fast network interfaces. *IEEE Micro*, 16:35–43, 1996.
8. Graham E. Fagg, Edgar Gabriel, Zizhong Chen, Thara Angskun, George Bosilca, Antonin Bukovsky, and Jack J. Dongarra. Fault tolerant communication library and applications for high performance computing. In *LACSI Symposium*, 2003.
9. Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number 2840 in LNCS, pages 257–267. Springer Verlag, 2003. 10th European PVM/MPI User's Group Meeting, Venice, Italy.

[7] This is already being done in OpenMPI

10. Rolf Rabenseifner and Jesper Larsson Träff. More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems. In *Proceedings of EuroPVM/MPI*, Lecture Notes in Computer Science. Springer-Verlag, 2004.
11. Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 131–140. ACM Press, 1999.
12. Christian Bell, Dan Bonachea, Yannick Cote, Jason Duell, Paul Hargrove, Parry Husbands, Costin Iancu, Michael Welcome, and Katherine Yelick. An evaluation of current high-performance networks. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 28.1. IEEE Computer Society, 2003.
13. Massimo Bernaschi, Giulio Iannello, and Mario Lauria. Efficient implementation of reduce-scatter in MPI. *J. Syst. Archit.*, 49(3):89–108, 2003.
14. Jehoshua Bruck, Ching-Tien Ho, Schlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, November 1997.
15. Ernie W. Chan, Marcel F. Heimlich, Avi Purkayastha, and Robert M. van de Geijn. On optimizing of collective communication. In *Cluster*, 2004.
16. Thilo Kielmann, Henri E. Bal, Sergei Gorlatch, Kees Verstoep, and Rutger F.H. Hofman. Network performance-aware collective communication for clustered wide-area systems. *Parallel Computing*, 27(11):1431–1456, 2001.
17. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.
18. William Gropp and Ewing L. Lusk. Reproducible measurements of MPI performance characteristics. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in PVM and MPI*, pages 11–18. Springer-Verlag, 1999.
19. Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.