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[134]

[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
[3] Oak Ridge National Laboratory
Computer Science and Mathematics Division
[4] University of Manchester

Abstract. Previous studies of application usage show that the performance of collective communications are critical for high-performance computing. Despite active research in the field, both general and feasible solution to the optimization of collective communication problem is still missing.

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, to collective operations. We compare the predictions from models against the experimentally gathered data and using these results, construct optimal decision function for broadcast collective. We quantitatively compare the quality of the model-based decision functions to the experimentally-optimal one. Additionally, in this work, we also introduce a new form of an optimized tree-based broadcast algorithm, splitted-binary.

Our results show that all of the models can provide useful insights into various aspects of the different algorithms as well as their relative performance. Still, based on our findings, we believe that the complete reliance on models would not yield optimal results. In addition, our experimental

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

results have identified the gap parameter as being the most critical for accurate modeling of both the classical point-to-point-based pipeline and our extensions to fan-out topologies.

1 Introduction

Previous studies of application usage show that the performance of collective communications are critical to high-performance computing (*HPC*). A profiling study [1] showed that some applications spend more than eighty percent of a transfer time in collective operations. Thus, it is essential for MPI implementations to provide high-performance collective operations. Collective operations (*collectives*) encompass a wide range of possible algorithms, topologies, and methods. The optimal[5] 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 segment size in use. 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.

Simple, yet time consuming way to find even a semi-optimal implementation of a 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][6]. 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

[5] The "optimal implementation" is defined 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.).

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

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. Additionally, we introduce a new form of optimized tree-based broadcast algorithm called splitted-binary. 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.

Grama et al. in [9] use Hockney model to perform cost analysis of different collective algorithms on various network topologies (such as torus, hypercube, etc). In [10], Thakur et al. discuss optimizations of their MPICH-2 MPI implementation. They use Hockney model to assess the performance of collectives and determine whether a particular algorithm would perform better for small or large message sizes. Using this analysis coupled with extensive testing they determine switching points between algorithms based on message size and whether the number of involved processors is exact power of two or not. Similarly, Chan et al. [11], use Hockney model to evaluate the performance of different collective algorithms on $c \times r$ mesh topology. Hockney model was used by Rabenseifner et al. in [12] to estimate performance of tree-based reduce algorithm optimized for large messages.

Kielmann et al. [13] use PLogP model to find optimal algorithm and parameters for topology-aware collective operations incorporated in the MagPIe library. The MagPIe library provides collective communication operations optimized for wide area systems. Across high-latency, wide-area links MagPIe selects segmented linear algorithms for collectives, while various tree-based algorithms are used in low-latency environment. Barchet-Estefanel et al. [14] use PLogP model to evaluate performance of broadcast and scatter operation on intra-cluster communication.

Bell et al. [15] use extensions of LogP and LogGP models to evaluate performance of small and large messages on contemporary super-computing networks. Similarly to PLogP, their extension of LogP/LogGP model accounts for the end-to-end latency instead of the transport latency. Additionally, they evaluate the potential for overlapping communication and computation on their systems. Bernaschi et al. [16] analyze the efficiency of reduce-scatter collective using LogGP model.

Vadhiyar et al. [2] use a modified LogP model which takes into account the number of pending requests that have been queued. Using this model coupled with modified hill-descent heuristics, they reduce the total number of tests necessary to tune the broadcast, scatter, gather, reduce, all-reduce, and all-gather collective on their systems.

The work in this paper is closest to the work published by Barchet-Estefanel et al. in [14]. Like them, we try to improve performance intra-cluster collective communication operations using parallel communication models. Unlike them, we consider wider range of collective operations, multiple communication models, and quantify the performance penalties which would arise from using models in place of extensive testing. Tables 1 through 4 point the reader to the relevant work related to the algorithms in question.

3 Summary of related models and parameters

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 equivalent 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 a 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[7]. 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 pro-

[7] 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"

ducer 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, 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, all processes involved in the barrier 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 [17] algorithm. In flat-tree/linear fan-in-fan-out algorithm all nodes report to a preselected root; once every node 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. The 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. All of these algorithms support message segmentation which potentially allows for overlap of concurrent communications. 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 at the root and going towards the leaf nodes through intermediate nodes. In the splitted-binary tree algorithm[8], 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.

Reduce. The 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 a 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, the 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

[8] To the best of our knowledge, no other group implemented or discussed this algorithm so far.

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, the root process posts (P-1) non-blocking receives followed by a 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.

6 Results and analysis

6.1 Experiment setup

The measurements were obtained on two dedicated[9] clusters provided by the SInRG project at the University of Tennessee at Knoxville. The first cluster,

[9] The micro-benchmark was the only user process executing on either cluster during the measurement.

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.

Boba, consists of 32 Dell Precision 530s nodes, each with Dual Pentium IV Xeon 2.4 GHz processors, 512 KB Cache, 2 GB memory, connected via Gigabit Ethernet. The second cluster, Frodo, consist of 32 nodes, each containing dual Opteron processor, 2 GB memory, connected via 100 Mbps Ethernet and Myrinet. In the results presented in this paper, we did not utilize the Myrinet interconnect on the Frodo cluster.

Model parameters. We measured the model parameters using various MPI implementations. Most of the collected data was obtained using FT-MPI [8], MPICH-1.2.6, and MPICH-2.0.97 [19]. 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 pa-

Broadcast	Model	Duration	Related work
Linear	*Hockney*	$T = n_s \cdot (P-1) \cdot (\alpha(m_s) + m_s \cdot \beta(m_s))$	[10], [11]
Linear	*LogP/LogGP*	$T = L + 2 \cdot o - g + n_s \times (P-1) \times ((m_s - 1)G + g)$	
Linear	*PLogP*	$T = L + n_s \cdot (P-1) \cdot g(m_s)$	[18], [14]
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*	$T = (P-1) \times (L + g(m_s)) + (n_s - 1) \times g(m_s)$	[14]
Binomial	*Hockney*	$T = \lceil log_2(P) \rceil \times n_s \times (\alpha(m_s) + m_s \cdot \beta(m_s))$	[10], [11]
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*	$T = \lceil log_2(P) \rceil \times (L + n_s \times g(m_s))$	[18], [14]
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*	$T = \begin{array}{l}(\lceil log_2(P+1) \rceil - 1) \cdot (L + 2 \cdot g(m_s)) + \\ (n_s - 1) \times \max\{2 \cdot g(m_s), o_r(m_s) + g(m_s) + o_s(m_s)\}\end{array}$	[18], [14]
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.

per, 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 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 [20] to ensure the reproducibility of the measured results. We minimize the effects of pipelining by forcing a "report-to-

Reduce	Model	Duration	Related work
Flat Tree	*Hockney*	$T = n_s \times (P-1) \times (\alpha(m_s) + \beta(m_s)m_s + \gamma m_s)$	[10], [11]
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*	$T = L + (P-1) \times n_s \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}$	[18]
Pipeline	*Hockney*	$T = (P + n_s - 2) \times (\alpha(m_s) + \beta(m_s)m_s + \gamma m_s)$	
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*	$T = n_s \times \lceil log_2(P) \rceil \times (\alpha(m_s) + \beta(m_s)m_s + \gamma m_s)$	[10], [11]
Binomial	*LogP/LogGP*	$T = \lceil log_2 P \rceil \times \Big(o + L + n_s \times ((m_s-1)G + \max\{g, o + \gamma m_s\}) \Big)$	[4], [5]
Binomial	*PLogP*	$T = \lceil log_2 P \rceil \times (L + n_s \times \max\{g(m_s), o_r(m_s) + \gamma m_s + o_s(m_s)\})$	
Binary	*Hockney*	$T = 2 \cdot (\lceil \log_2(P+1) \rceil + n_s - 2) \times (\alpha(m_s) + \beta(m_s)m_s + \gamma m_S)$	[10], [11]
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*	$T = \begin{array}{l} (\lceil log_2(P+1) \rceil - 1) \times (L + 2 \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}) + \\ (n_s - 1) \times (o_s(m_s) + 2 \times \max\{g(m_s), o_r(m_s) + \gamma m_s\}) \end{array}$	

Table 3. Analysis of different reduce algorithms.

Alltoall	Model	Duration	Related work
Linear	*Hockney*	$T = \begin{array}{l} P \times (\alpha(m_s) + \beta(m_s)m_s) + \\ (P-1) \times (n_s \times P + 1 - \frac{P}{2}) \times \alpha(m_s) \end{array}$	[10]
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*	$T = (P-1) \times (\alpha(m) + \beta(m)m)$	[10]
Pairwise exchange	*LogP/LogGP*	$T = (P-1) \times (L + o + (m-1) \times G + g)$	
Pairwise exchange	*PLogP*	$T = (P-1) \times (L + g(m))$	

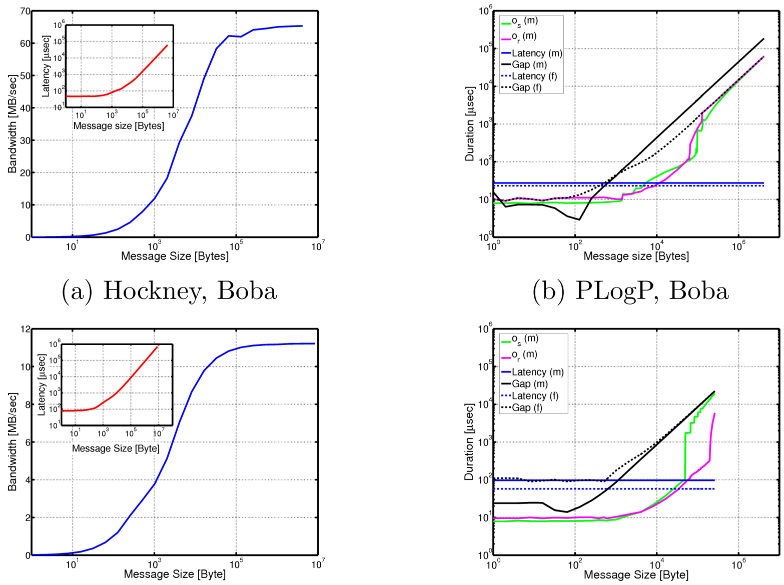
Table 4. Analysis of different alltoall algorithms.

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.

6.2 Performance of different collective algorithms

We executed performance tests on various algorithms for barrier, broadcast, reduce, and alltoall collective operations using FT-MPI, MPICH-1, and MPICH-2. We then analyzed the algorithm performance and the optimal implementation of various collective operations using parallel communication models, Hockney, LogP/LogGP, and PLogP. When predicting performance of collective operations that exchanged actual data (message size > 0) we did not consider pure LogP predictions, but used LogGP instead (See Section 3.1).

In our experiments, we found that the model of worst case performance of an algorithm is often too pessimistic, as in the case of the flat-tree/linear fan-



(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.

in-fan-out barrier algorithm. Our experience with the MPI implementations was that the algorithms performance was generally closer to the best case scenario. Thus, where applicable 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.

Experimental data for both Bruck and recursive doubling algorithms, while exhibiting trends, is not uniform. The reasons for this could be both our measurement procedure and the lock-step communication pattern of these algorithms. The "report-to-root" step in the performance measurement procedure takes time comparable to the time taken by Bruck and recursive doubling barrier algorithms. Thus, the reported measurement is affected more significantly by the variations in this step then it would be for longer running collectives. Moreover,

LogP/LogGP		***Boba cluster***	***Frodo cluster***
Latency	L	30.40 [μsec]	61.22 [μsec]
Overhead	o	8.15 [μsec]	8.2 [μsec]
Gap	g	8.683 [μsec]	23.8 [μsec]:
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.

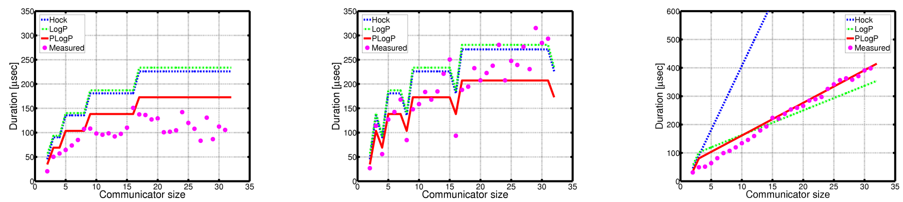
as these algorithms communicate with different processes in lock-step manner, delay on a single process would affect whole operation. At the same time, the flat-tree/linear fan-in-fan-out barrier which takes slightly longer to complete and has a more regular communication pattern does not exhibit this problem.

The 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 displayed in Figure 2 were obtained using MPICH2, but the behavior was consistent with results obtained using FT-MPI on the Frodo system (See discussion on optimal broadcast decision function). This implies that the underlying system (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, the models captured relative performance of these barrier algorithms sufficiently correctly.

Reduce performance. 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. Results indicate that for small message sizes, non-segmented binomial algorithm outperforms pipeline algorithm, while for large message sizes, the segmented pipeline algorithm would have best performance.

Experimental data for non-segmented binomial and pipeline reduce algorithms exhibits non-linear increase in duration for the message sizes in the range from 1KB to 10KB. The similar increase can be observed for large message sizes

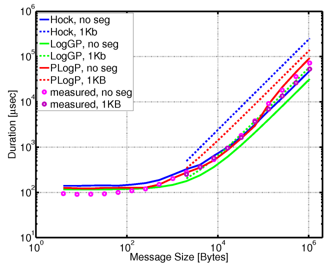


(a) Bruck (b) Recursive Doubling (c) Flat-tree

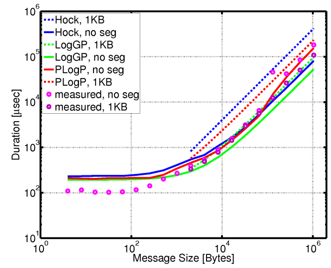
Fig. 2. Performance of barrier algorithms: Experimentally measured values are indicated by circles. (MPICH-2, Boba cluster, GigE).

($>$ 100KB) on non-segmented binomial algorithm. All three models were able to capture relative performance of non-segmented algorithms in question. However, LogP/LogGP failed to capture non-linear increase in duration for the intermediate sized messages. PLogP was the only model which captured non-linear increase in duration of non-segmented binomial algorithm for large message sizes. We can explain these shortcomings by considering the model parameters. The LogP/LogGP model assumes linear dependence between the time to send/receive a message and message size. However, the results in Figure 3 show that in general, this is certainly not the case. The Hockney model is able to capture the first non-linear behavior because its parameters are function of message size. However, once the message size exceeds 100KB, the transfer rate, β parameter, reaches its asymptotic value and effectively becomes an constant, thus preventing the Hockney model to capture the non-linear behavior in that message size range. The gap and parameters of PLogP model are a function of message size, so some of the nonlinear effects can be accounted for. We believe that non-linear changes in values of sender and receiver overheads (Figure 1) enabled PLogP to capture performance of these methods.

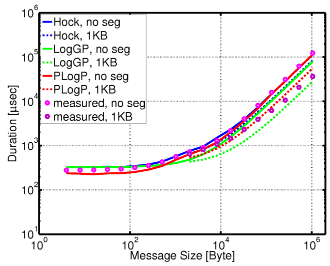
In the experiments in Figure 3, segmentation using 1KB segments improved performance of both pipeline and binomial reduce algorithms. While segmentation incurs overhead for managing multiple messages, it also enables higher bandwidth utilization due to increased number of concurrent messages; provides opportunity to overlap multiple communications and computation; and limits the size of internal buffers required by the algorithm. The models of pipeline reduce in Table 3 dictate that as the number of segments, n_s, increases (total message size increases), the algorithm should achieve asymptotically optimal performance. In the asymptotic case, the segmented pipeline reduce algorithm



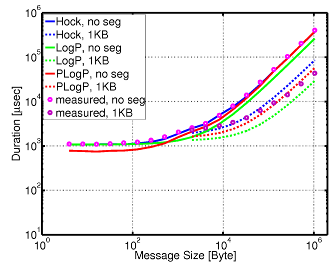
(a) Binomial reduce, 8 nodes



(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).

should take a constant amount of time for message of size m and should not depend on number of processes, P. The results in Figure 3 (c) and (d) are consistent with this observation: the duration of segmented pipeline reduce on 8 and 24 nodes takes around $4 \times 10^4 \mu sec$. All three models correctly captured the relative performance of segmented pipeline algorithm, and the PLogP model had best estimate of the absolute duration of the operation.

Modeling performance of segmented binomial reduce algorithm proved to be a challenge for all three models. Contrary to the measured results, the formulas in Table 3 seem to indicate that with increased number of segments the duration of binomial reduce operation should increase and model predictions in Figure 3 agree with that. However, to determine if models are capable of recognizing the benefit of segmentation for binomial reduce algorithm we have to analyze these formulas in more detail.

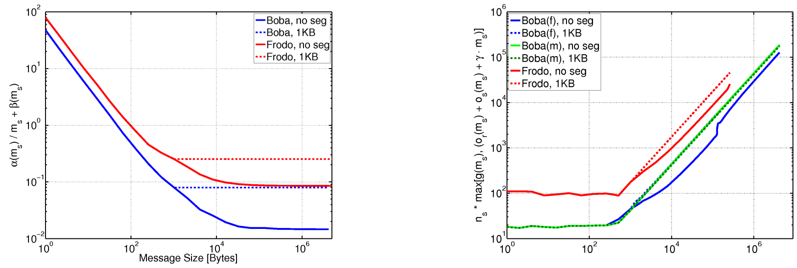
According to the Hockney model of segmented binomial reduce algorithm, the segmentation should improve operation performance when $(\alpha(m)+\beta(m)\cdot m+$

$\gamma \cdot m) > n_s \times (\alpha(m_s) + \beta(m_s) \cdot m_s + \gamma \cdot m_s)$. Taking in account that $m = n_s \cdot m_s$, we conclude that under the Hockney model, segmentation of binomial reduce algorithm would improve performance when $(\frac{\alpha(m)}{m} + \beta(m)) > (\frac{\alpha(m_s)}{m_s} + \beta(m_s))$. Figure 4 (a) shows how do the left- and right-hand sides of this condition depend on message size. On systems we considered, measured Hockney model parameter values were such that according to the model, segmentation should not improve the performance of binomial reduce.

In LogP/LogGP model of segmented binomial reduce algorithm, we can see that only condition under which segmentation would be beneficial is $(\max\{g, (o + \gamma \cdot m)\}) > ((n_s - 1) \cdot (m_s - 1) \cdot G + n_s \cdot \max\{g, (o + \gamma \cdot m_s)\})$, or equivalently $(\max\{g, (o + \gamma \cdot m)\}) > ((n_s - 1) \cdot (m_s - 1) \cdot G + \max\{n_s \cdot g, (n_s \cdot o + \gamma \cdot m)\})$. Noting that all LogP/LogGP parameters are positive numbers, as well as message size, segment size, and number of segments, we conclude that this condition is not possible to obtain. Thus, under LogP/LogGP model it is impossible to get result in which the message segmentation would improve performance of binomial reduce algorithm.

The PLogP model of segmented binomial reduce algorithm shows that the following condition is necessary for segmentation to improve performance of this algorithm: $(\max\{g(m), (o_r(m) + o_s(m) + \gamma \cdot m)\}) > \max\{(n_s \cdot g(m_s)), [n_s \cdot (o_r(m_s) + o_s(m_s)) + \gamma \cdot m]\})$. Figure 4 (b) illustrates this condition as function of message size. In case of fitted parameters on both clusters (See discussion from Section 6.1), the non-segmented version of binomial reduce algorithm always outperforms the one with 1KB segments. However, when the directly measured PLogP parameters on Boba cluster are used to evaluate the condition, the segmented version outperforms non-segmented version by a slight margin. Thus, using the measured parameters the PLogP model of binomial reduce algorithm would capture segmentation effect correctly.

The analysis of model parameters and effect of segmentation on binomial reduce shows that the models are fairly sensitive to parameter values. As observed in the case of the flat-tree barrier algorithm, the gap between messages depends on number of nodes we are communicating with, and for communicator sizes greater than 16 nodes it decreased in comparison to smaller communicator sizes. However, PLogP and LogP/LogGP models cannot include this dependence, and Hockney model does not even have notion of the gap. Additionally, 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



(a) Hockney (b) PLogP

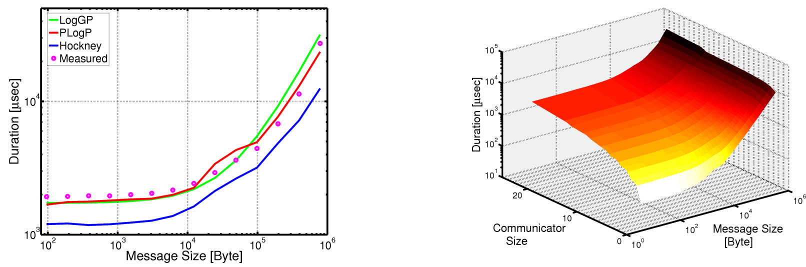
Fig. 4. Segmentation and binomial reduce algorithm. Figure displays necessary conditions under which segmentation of binomial reduce algorithm would improve algorithm performance using Hockney and PLogP models. LogP/LogGP models do not have condition under which the segmentation of this algorithm would improve performance. In this Figure we use parameters from Figure 1 and assume segment size of 1KB.

window could require resizing the window and an extra memory copy operation per pair of communicating parties (which in this case is $log_2(P)$ times). Only PLogP model considers sender and receiver overheads to depend on message size, LogP/LogGP and Hockney do not have this notion.

Alltoall performance. Figure 5 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 congestion 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. This indicates that in the test, the communication was scheduled correctly and we did not over-flood the switch.

6.3 Analysis of optimal broadcast implementation.

Figure 6 shows the optimal implementation of the broadcast collective using measured data and model predictions on the Frodo cluster. The optimal implementation of the collective is described by a *decision function*. Given the



(a) (b)

Fig. 5. 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).

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 Megabytes 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. This result is surprising but possible if we take in account the decrease in the gap per message parameter when communicating to more than 16 nodes. Not surprisingly, all models mispredict the optimal method for that section of the parameter space. For message sizes close to 1KB measured data suggests that all tree-based non-segmented algorithms can be optimal, i.e. 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.

The Hockney model broadcast decision function, Figure 6 (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, since gap determines whether it will be more cost effective to have a longer pipeline or a wider tree. The Hockney model which has no notion of gap parameter, favors tree-based algorithms as they decrease the latency term, and larger segment size because they lower the overall number of messages. However, the experimental results clearly show that in case of broadcast and reduce collectives segmented pipeline algorithms should be considered for large message sizes.

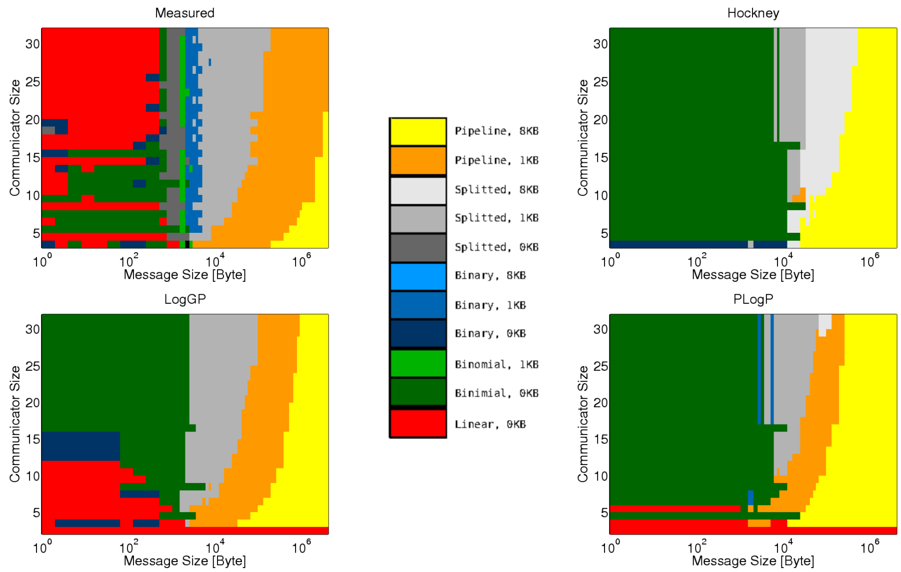
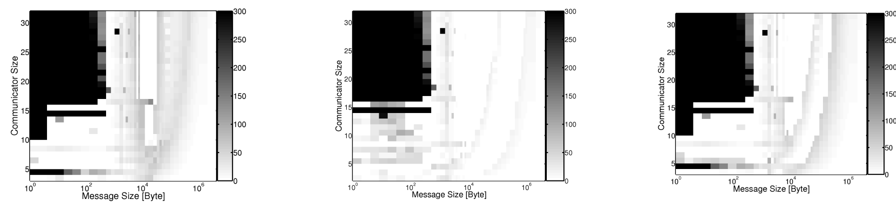


Fig. 6. Broadcast decision function. Graphs in this figure should be read in the following way: the color at point (m, P) represents the best broadcast method for message size m and communicator size P. Label with 0KB segment size denotes a non-segmented version of the algorithm. (FT-MPI, Frodo cluster, 100Mbps).

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 by using the model generated decision function instead of using a measured one? Figure 7 addresses this question. The performance penalty of 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 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 the result of a micro-benchmark. Individual, real-world applications' performance and their performance losses or gains, could vary greatly depending on application communication patterns.



(a) Hockney (b) LogP/LogGP (c) PLogP

Fig. 7. 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 color-bar at the right of every graph shows the percentage range: from 0 to 300%, white color means less than 5%. (FT-MPI, Frodo cluster, 100Mbps)

7 Discussion and future work

We compare the Hockney, LogP/LogGP, and PLogP parallel communication models applied to inter-cluster MPI collective operations on two systems at the University of Tennessee. Our results indicate that all of the models can provide useful insights into various aspects of the collective algorithms and their relative performance. We also demonstrate the importance of accurate modeling of the gap between sending consecutive messages to a single destination and to a set of different destination processes. Experiments show that the value of gap between consecutive send operations depends on the number of unique destination nodes. Unfortunately, neither of the models is able to capture this behavior correctly. This shortcoming is reflected in underestimating the benefit of using segmentation for binomial reduce algorithm and the inaccurate prediction of switching points between available broadcast methods for large messages. Additionally, neither of the point-to-point models used in this paper, considers network congestion directly. Nonetheless, for the communicator and the message size range we consider, PLogP and LogP/LogGP models are able to model pairwise-exchange alltoall algorithm successfully.

We believe that parallel communication models can still be used to perform focused tuning of collective operations. Based on measured parameter values coupled with small number of test runs which would be used to verify predictions and adjust model parameters, one could use the models to decrease the number of physical tests needed to construct semi-optimal decision function for a particular collective. The work in this paper could be further improved by extending existing models to include gap parameter which depends on both message size and number of nodes we are communicating with, as well as the contention.

The performance analysis of different collective methods presented in this paper 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 decision function within the tuned collective module in the Open MPI library [21]. 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.

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[10]; and building decision function refinement capability which would use a 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 additional information about the underlying network topology.

8 Acknowledgments

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

[10] This is already included in tuned collective module in Open MPI

References

1. Rabenseifner, R.: 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. (1999) 77–85
2. Vadhiyar, S.S., Fagg, G.E., Dongarra, J.J.: Automatically tuned collective communications. In: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), IEEE Computer Society (2000) 3
3. Hockney, R.: The communication challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Computing **20**(3) (1994) 389–398
4. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: Towards a realistic model of parallel computation. In: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, ACM Press (1993) 1–12
5. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: LogGP: Incorporating long messages into the LogP model. In: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, ACM Press (1995) 95–105
6. Kielmann, T., Bal, H., Verstoep, K.: Fast measurement of LogP parameters for message passing platforms. In Rolim, J.D.P., ed.: IPDPS Workshops. Volume 1800 of Lecture Notes in Computer Science., London, UK, Springer-Verlag (2000) 1176–1183
7. Culler, D., Liu, L.T., Martin, R.P., Yoshikawa, C.: Assessing fast network interfaces. IEEE Micro **16** (1996) 35–43
8. Fagg, G.E., Gabriel, E., Chen, Z., Angskun, T., Bosilca, G., Bukovsky, A., Dongarra, J.J.: Fault tolerant communication library and applications for high performance computing. In: LACSI Symposium. (2003)
9. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. Second edition edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
10. Thakur, R., Gropp, W.: Improving the performance of collective operations in MPICH. In Dongarra, J., Laforenza, D., Orlando, S., eds.: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2840 in LNCS, Springer Verlag (2003) 257–267 10th European PVM/MPI User's Group Meeting, Venice, Italy.
11. Chan, E.W., Heimlich, M.F., Purkayastha, A., van de Geijn, R.A.: On optimizing collective communication. In: CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing, Washington, DC, USA, IEEE Computer Society (2004) 145–155

12. Rabenseifner, R., Träff, J.L.: 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)
13. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: 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, New York, NY, USA, ACM Press (1999) 131–140
14. Barchet-Estefanel, L.A., Mounié, G.: Fast tuning of intra-cluster collective communications. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (2004) 28–35
15. Bell, C., Bonachea, D., Cote, Y., Duell, J., Hargrove, P., Husbands, P., Iancu, C., Welcome, M., Yelick, K.: An evaluation of current high-performance networks. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, IEEE Computer Society (2003) 28.1
16. Bernaschi, M., Iannello, G., Lauria, M.: Efficient implementation of reduce-scatter in MPI. J. Syst. Archit. **49**(3) (2003) 89–108
17. Bruck, J., Ho, C.T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Transactions on Parallel and Distributed Systems **8**(11) (1997) 1143–1156
18. Kielmann, T., Bal, H.E., Gorlatch, S., Verstoep, K., Hofman, R.F.: Network performance-aware collective communication for clustered wide-area systems. Parallel Computing **27**(11) (2001) 1431–1456
19. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing **22**(6) (1996) 789–828
20. Gropp, W., Lusk, E.L.: Reproducible measurements of MPI performance characteristics. In: Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in PVM and MPI, London, UK, Springer-Verlag (1999) 11–18
21. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (2004) 97–104