Abstract—The time spent in communication operations is a major factor in determining the scalability of parallel applications. Tuning the parameters of a communication library can be used to adapt its characteristics to a particular platform, minimizing the communication time of an application. The goal of this paper is to improve theoretical and practical understanding of how performance improvements of point-to-point operations propagate to collective communication operations. We derive formulas to determine the expected improvement of a collective operation based on the improvement observed for a point-to-point communication using Hockney’s model and the LogGP model. Our results indicate that many collective algorithms will inherently see a lower performance improvements compared to the improvement observed for point-to-point operations. Our evaluation shows for most test cases a good match between the predictions made by our models and the observed data, but also identifies multiple reasons for potential disparity between theory and practice.

I. INTRODUCTION

The performance and scalability of parallel applications strongly depends on the time it spends in communication operations. End-users often spend enormous efforts to tune the communication aspects of their application. In addition to optimizations performed on the application level, communication libraries such as Open MPI [5] have a significant number of parameters that influence the performance of communication operations. Examples for these parameters are the cross-over point between eager and rendezvous protocol for point-to-point operations or switching points between different implementations of collective operations. The focus of this paper is on the second aspect, namely tuning parameters of a communication library.

Tuning a communication library for a particular application requires multiple steps, namely: i) identifying the set of individual and collective operations used by an application ii) identifying the message lengths used by the application for a particular application scenario; and iii) tuning parameters of the occurring individual and collective operations for the message length observed. The latter can be done semi-automatically using tools such as the Open Tool for Parameter Optimization (OTPO) [2].

The goal of this work is to establish better understanding both theoretically and practically on how improvements in the communication time of individual data transfer operations propagate to collective operations. The starting point of the analysis is a set of parameters for the Open MPI communication library that improves the execution time of a point-to-point communication operation. Using this improvement theoretical boundaries for each collective operation and algorithm have been derived for Hockney’s [7] and the LogGP communication model [1]. The paper also evaluates whether the models developed match actual observations by using measurements performed on an InfiniBand cluster.

Using performance models to evaluate collective operations has been been part of virtually all publications on performance of collective operations [3, 14, 15]. Combining performance modeling and tuning is however a relatively new approach complementing tuning approaches such as [10] or [6].

The remainder of the paper is organized as follows: Section II provides theoretical boundaries for multiple algorithms and collective operations, while section III compares the results obtained to actual measurements. Finally, section IV summarizes the paper and presents ongoing work in the area.

II. CONCEPT

Tuning the performance of communication operations in a parallel application requires multiple steps. In the first step, users have to generate a profile of their application in order to identify the most commonly used individual and collective operations as well as the dominant message lengths used in a particular application scenario. The second step consists of tuning parameters of the communication library for this particular application (scenario) and retrieve a set of parameters that minimizes the communication time. For the actual production runs, the user provides the parameter sets that were deemed to be optimal to the application manually or in a semi-automatic manner [9].

The Open MPI communication library accepts runtime parameters as an argument to $\text{mpirun}$, environment variables, or in configuration files. While most end-users depend on the Open MPI developers to set reasonable default values for each parameter, sophisticated end-users can (and do) change values of certain parameters to optimize for certain certain platforms/scenarios.

The Open Tool for Parameter Optimization (OTPO) [2] is designed to systematically explore the parameter space spawned by the Open MPI parameters. OTPO requires a configuration file which contains the list of parameters as
well as the range of values to be explored for each parameter. Furthermore, the user has to specify the benchmark or application to be executed with OTPO. OTPO launches the benchmark with various combinations of parameter values in order to determine the combination that leads to the lowest overall execution time. The result of the tuning process is a list of the parameter combinations that resulted in the best performance for the benchmark specified.

Tuning the parameters of Open MPI poses however multiple challenges. Deciding which of the over 400 parameters to tune requires some knowledge of the internals of the communication library and platform, since despite of using advanced search algorithms, tuning all parameters is not feasible. Furthermore, applications will have more than one relevant message length, each of which would lead to a separate 'optimal' set of parameters for the communication library. Open MPI can however only handle one set of parameters within a job, i.e. changing the value of a parameter after the job has been launched is except for very few parameters not an option. Thus, the benchmark has to utilize all the message lengths used by the application. This could be achieved by using the application itself for the tuning. This is in the vast majority of the cases however unrealistic, since the tuning step requires the re-execution of the benchmark/application hundreds or even thousands of times, necessitating benchmarks that take a few seconds per execution at most to keep the time spend in the tuning procedure within reasonable limits. Hence, most tuning tools – including OTPO – rely on simple communication benchmarks such as NetPipe [13] for point-to-point operations and SkaMPI [12] for collective operations.

Using microbenchmarks for the tuning step reduces the time spent in the tuning operation itself, the resulting parameter sets are however not necessarily optimal from the application perspective. We documented in our previous work [9] that optimizing parameters of the InfiniBand btl component of Open MPI for a given message lengths showed in very few scenarios the expected performance improvement, despite of significant performance improvements observed for a simple ping-pong benchmark. Even for relatively simple scenarios, e.g. a simple benchmark executing an All-to-all communication operation, performance benefits could be observed for some algorithms used to implement the collective operation, but not for others.

The goal of this work is to establish better understanding both theoretically and practically on how improvements in the communication time of individual data transfer operations propagate to collective operations. The focus of this work is on collective operations since they a) represent important building blocks of many application, b) are well understood multi-process communication patterns, and c) use a single message length and thus allows us to isolate the problem that we tackle from the multi message-length problem described above.

The starting point of the analysis is a parameter set for Open MPI that improves the execution time $t$ of a point-to-point communication operation of message length $m$ by a factor of $i^{2p}$. The problem that this paper addresses can be formulated as follows: given $m$, $i^{2p}$, and the number of processes $p$, what is the expected and the observed performance improvement for a collective operation for using the 'optimal' parameter set for the message length $m$ vs. using the default values. The paper focuses on three collective operations with multiple algorithms implementing the operation, namely:

- **broadcast**: chain, binary tree, binomial tree
- **all-gather**: ring, neighbor exchange, recursive doubling
- **all-to-all**: linear, pairwise exchange, Bruck’s algorithm

In the following, we give a brief overview of the communication models used in this analysis, derive formulas to estimate the improvement of a collective operation based on the NetPipe results observed, and derive furthermore the formula’s necessary to determine the tuned parameters of the communication operation.

### A. Description of Communication Models

Within the context of this work, we used the LogGP [11] communication model and Hockney’s model, although the second one is omitted from this paper for the sake of brevity. The LogGP model has four parameters, namely:

- **$L$**: communication delay, which is the hardware latency of the network
- **$\sigma$**: the software communication overhead i.e. the time spent in the MPI library before injecting data into the network
- **$g$**: the gap parameter, which is a hardware parameter dictating the minimum time before being able to inject two subsequent messages into the network
- **$G$**: Gap per byte, which is the reciprocal value of the network bandwidth

According to the LogGP model, the time to send a message of size $m$ between two nodes is given by $L + 2\sigma + (m - 1)G$.

The starting point of this work are the estimated costs of all algorithms mentioned previously, and is based on previously published work by multiple research groups [1, 4, 11]

### B. Estimating the Improvement of Collective Operations

In the following, we present an approach to derive an estimate for the improvement in the execution time of a collective communication operation given the improvement of the execution time of a point-to-point as a result of the tuning step. The ubiquitous formula for computing a relative improvement in a data point from an untuned value to a tuned value is

$$i = \frac{t_{untuned}(m) - t_{tuned}(m)}{t_{untuned}(m)},$$  \hspace{1cm} (1)$$

where $i$ is the performance improvement, $t_{tuned}(m)$ and $t_{untuned}(m)$ is the execution time of communication operation of a message of length $m$, using an optimized parameter set and the default settings respectively.

The binary tree broadcast operation is used to demonstrate how to derive the formula for the expected improvement $i_{coll}$. According to LogGP model the execution time of a binary
tree broadcast operation is
\[ t_{\text{untuned}} = (\lfloor \log_2(P + 1) \rfloor - 1) \cdot (L + g + 2(o + (m - 1)G)). \] (2)

The fundamental assumption that we make for the sake of simplicity is that the improvement observed when tuning a point-to-point operation only affects one parameter of the model at any given point in time. The LogGP model has four parameters that could be affected by the tuning. We focus in the paper on the overhead \( o \) and gap per byte \( G \) parameters for the sake of brevity. Let us denote the new latency and bandwidth as \( o^* \) and \( G^* \). Therefore, the tuned execution time is
\[ t_{\text{tuned}} = (\lfloor \log_2(P + 1) \rfloor - 1) \cdot (L + g + 2(o^* + (m - 1)G^*)). \] (3)

Substituting (2) and (3) in (1) leads to
\[ i_{\text{coll}} = \frac{2(o + (m - 1) \cdot G) - 2(o^* + (m - 1)G^*)}{L + g + 2(o + (m - 1)G)}. \] (4)

In the first case, we assume that the gap per byte parameter \( G \) remains unchanged, and the improvement has been reflected entirely in the overhead \( o \), with \( o^* = x \cdot o \) and \( G^* = G \). Note, that \( x \) is not the observed improvement of the point-to-point operation \( i^{p2p} \). Section II-C shows how derive \( o^* \) and \( G^* \) given \( i^{p2p} \). The performance improvement is in this case
\[ i_{\text{LogGP}(o)}^{\text{coll}} = \frac{2o(1 - x)}{L + g + 2o + 2G(m - 1)}. \] (5)

In the second case, we assume that the overhead remains unchanged, and the improvement has been reflected entirely in the gap per byte, i.e. \( G^* = x \cdot G \) and \( o^* = o \), leading to
\[ i_{\text{LogGP}(G)}^{\text{coll}} = \frac{2G(m - 1)(1 - x)}{L + g + 2o + 2G(m - 1)}. \] (6)

C. Deriving the tuned parameter values

In this section we derive equations to determine the improved network parameter \( o^*, G^* \). Using the equation (1), we can conclude that
\[ t_{\text{untuned}}(m) = (1 - i) \times t_{\text{tuned}}(m). \] (7)

In the LogGP model the execution time of a point to point operation can be estimated by
\[ t(m) = L + 2o + (m - 1) \cdot G. \] (8)

From equations (8) and (7) we can derive that
\[ L + 2o^* + (m - 1)G^* = (1 - i^{p2p})(L + 2o + (m - 1)G). \] (9)

Assuming that overhead remains unchanged, i.e. \( o^* = o \), leads to
\[ G^* = G - \frac{i^{p2p} \cdot (L + 2o + (m - 1)G)}{m - 1} \] (10)

and assuming in the second case that \( G^* = G \)
\[ o^* = o - \frac{i^{p2p}}{2} (L + 2o + (m - 1) \cdot G) \] (11)

D. Performance improvement of Collective Operations in Terms of Point-to-point Improvement

Using the formulas derived in section II-C, one can directly determine the improvement factor \( x \) used in section II-B as the ratio of the tuned vs. untuned parameter value. Alternatively, one could use the tuned parameter values from section II-C to calculate the execution time of a collective operation using the base formulas presented in the literature, and determine the expected improvement of the collective operation by simply applying the formula shown in (1).

In this subsection, we would like to derive the expected improvement of a collective operation as a function of the improvement in the point-to-point operation, since this provides high-level information on the expected performance improvement.

The expected performance improvement of a binary tree broadcast operation can be derived from substituting (11) in (4) assuming that \( G^* = G \). This leads to
\[ i_{\text{LogGP}(o)}^{\text{coll}} = \frac{i^{p2p} \cdot L + 2o + (m - 1)G}{L + g + 2(o + (m - 1)G)} \] (12)

and for the second scenario, assuming \( o^* = o \), by substituting (10) in (4)
\[ i_{\text{LogGP}(G)}^{\text{coll}} = \frac{i^{p2p} \cdot 2(L + 2o + (m - 1)G)}{L + g + 2(o + (m - 1)G)} \] (13)

Assuming that the performance improvement of a point-to-point operation stems from improving the overhead \( o \), the expected performance benefit of a binary tree broadcast operation will be lower than the measured improvement of the point-to-point operation, since the denominator in (12) is always larger than the numerator. On the other hand, if the improvement comes from the parameter \( G \), as assumed in eq. (13), a collective operation could see a larger, equal or lower improvement than the improvement measured by the point-to-point operation, depending on whether \( L + 2o \) is larger, equal or less than \( g \). Based on LogGP parameter values that we observed using the Netgauge tool [8] on various platform, all three possibilities can occur in real life.

Table I lists the expected improvement for all operations and algorithms discussed in this paper. The last column indicates whether the expected improvement of the collective operation is equal (=), larger (>) or lower (<) than that of the point-to-point operation. For some formulas, the column is left blank, which indicates that all three options are possible. For a few algorithms, namely chain and binomial tree broadcast, and ring allgather, both models agree that the expected improvement in the performance of the collective operation should be equal to the improvement of the point-to-point operation.

III. COMPARISON TO ACTUAL MEASUREMENTS

The goal of this section is to evaluate how well the performance improvement models derived in section II match actually observed data. Tests in this section have been executed on the crill cluster at the University of Houston using Open MPI 1.8.3. The crill cluster consists of 16 nodes with four
12-core AMD Opteron (Magny Cours) processor cores each (48 cores per node, 768 cores total) and 64 GB of main memory per node. Each node has four 4X DDR InfiniBand Host Channel Adaptors (HCAs), although only one HCA has been used in the subsequent tests. All tests have been executed at least three times, and the average values are presented subsequently. Tests have been executed for 32 and 64 processes for message length of 128 bytes, 1 Kbyte, 12 KB, 16 KB, 32 KB and 64 KB. The tuning step involved the tuning of seven parameters of the openib btl component, the parameter names and values are shown in II. In our tests we enforced that all communication operations are going through the InfiniBand network by disabling shared memory btl components.

The evaluation consists of the following steps:

1) Using OTPO and the NetPipe benchmark, tune the initial set of parameters for each message length individually. The result of this step is one or multiple sets of parameters per message length $p_{set}(m)$ which lead to minimal execution time as reported by NetPipe.

2) Calculate the improvement $i_{2p}(m)$ for each message length individually by comparing the execution time reported by NetPipe with default parameter values vs. the tuned parameter set $p_{set}(m)$.

3) Measure the execution time for each collective operation and algorithm using the SkAmpi benchmark for each message length $m$ using the default parameter values as well as the top 5 parameter sets $p_{set}(m)$ determined in step 1. For the subsequent analysis, we choose the best result obtained with any of the top 5 parameter sets.

4) Calculate the expected improvement $i_{coll}(m)$ for each message length using the formulas shown in I. The parameters of the untuned model were determined using the NetPipe benchmark for Hockney's model, using the 0-byte data transfer costs for the message latency, and the asymptotic maximum bandwidth observed for the bandwidth. Those values were determined to be $l = 1.6 \mu s$ and $b = 1941.5 MB/s$ for the crill cluster. For the LogGP model, we used the NetGauge toolkit [8] version 2.1. The values used are: $L = 1.84 \mu s$, $o = 1.49 \mu s$, $g = 0.08 \mu s$ for $m \leq 32$ KB and $g = 11.9 \mu s$ for $m \geq 32$ KB, $G = 0.00067 \mu s$.

The most noteworthy observation of the results of performance improvement from the previous work [9] is that the performance improvement observed for point-to-point operations is not uniform across different message lengths. On this particular platform, the performance of very small and large messages could not be significantly improved by tuning the openib parameters used. However, for messages in the range of 12KB to 36KB, the performance of point-to-point operations could be improved by up to 13% compared to the

### TABLE I

**IMPROVEMENT FOR EACH COLLECTIVE OPERATION AND ALGORITHM FOR LogGP COMMUNICATION MODELS.**

<table>
<thead>
<tr>
<th>Collective</th>
<th>Algorithm</th>
<th>Condition</th>
<th>Performance improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast</td>
<td>Chain</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Binary</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Binomial</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Recursive Doubling</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Neighbor Exchange</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Alltoall</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Linear</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Pairwise</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
<tr>
<td></td>
<td>Bruck</td>
<td>$G^* = G$</td>
<td>$i_{coll} = i_{2p}$</td>
</tr>
</tbody>
</table>

### TABLE II

**OPEN MPI PARAMETERS TUNED IN THE SENSITIVITY ANALYSIS**

<table>
<thead>
<tr>
<th>Parameter (btl_openib)</th>
<th>Default value</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>eager_limit</td>
<td>12K</td>
<td>1K-48K-72K</td>
</tr>
<tr>
<td>rdma_num</td>
<td>16</td>
<td>1-32:*2</td>
</tr>
<tr>
<td>rdma_threshold</td>
<td>16</td>
<td>4-32:*2</td>
</tr>
<tr>
<td>use_eager_rdma</td>
<td>1</td>
<td>0,1</td>
</tr>
<tr>
<td>use_message_coalescing</td>
<td>1</td>
<td>0,1</td>
</tr>
<tr>
<td>free_list_num</td>
<td>8</td>
<td>2-32:*2</td>
</tr>
<tr>
<td>free_list_inc</td>
<td>32</td>
<td>8-64:*2</td>
</tr>
</tbody>
</table>
A. Results of Tuning Collective Operations

Figures 1 and 2 present a subset of the results obtained for the all-gather, broadcast and all-to-all operations. Each graph contains 6 lines, namely:
- **Netpipe**: the improvement observed for point-to-point operations for the corresponding message length using the NetPipe benchmark.
- **<algorithm name>**: the measured improvement of a given algorithm (e.g. binary bcast, chain bcast, etc).
- **Hockney(l) and Hockney(b)**: the predicted performance improvement of a collective operation using Hockney’s model assuming that the improvement can be attributed to the latency only / bandwidth only.
- **LogGP(G) and LogGP(o)**: the predicted performance improvement of a collective operation using LogGP model assuming that the improvement can be attributed to the gap per byte parameter or the overhead parameter respectively.

Note that on some graphs (allgather ring, bcast binomial, bcast chain), all models are identical and equal to $i^{2p}(m)$. Thus, only two lines are visible, namely the measured and the predicted performance.

The main result shown in these graphs is that for the all-gather and the broadcast operations, there is a good correlation between the observed and the predicted improvement of the collective operation, i.e. the trend of the observed performance improvement matches at least one, but typically multiple of the predicted lines. The observed improvement of the all-gather neighbor exchange and all-gather recursive doubling algorithm favor the models that assume the latency related parameters are affected by the tuning, i.e. they follow the predictions made by the Hockney(l) and LogGP(o) models.

The all-to-all results reemphasize the value and the limitations of performance models. While performance models are useful to understand fundamental properties of the algorithms and thus guide the expectation of end-users, they do represent simplifications compared to the real-world. Some data points for the linear and pairwise all-to-all algorithm represent perfect examples for that, since the observed performance benefit is much larger than predicted by any of the models. One has to keep in mind however the number of assumptions, simplifications and potential inaccuracies made in any of the models and the measurements, including handle network congestion or protocol switch from eager to rendezvous protocol. Considering all of these aspects, our results represent a reasonable good match between the models and observed data with occasional outliers, very similar to many other performance analysis papers using these models. Out of the nine algorithms used in this study, eight had a very small Mean Squared Error between the most optimal prediction model and the observed improvement (between .0002 to .011) for the 64 process test cases, indicating a good overall match. In the following, we discuss an interesting sub-topic that also contribute towards limitations of the modeling aspect.
B. Impact of number of processes on optimal parameter values

When tuning parameters of Open MPI using a point-to-point benchmark, one makes fundamentally the assumption that the optimal value for a parameter does not change with the number of processes used. This is assumption is not always correct. Consider for example an analysis performed with one of the parameters listed in Table II, namely btl_openib_eager_limit. This parameter defines the threshold value starting from which Open MPI will use a rendezvous protocol for the data transfer over an InfiniBand network instead of the eager protocol. Since the range of values for this parameter was set to be between 4KB and 48 KB, one would expect no impact of this parameter for 64KB message length, since the message is expected to be transferred in rendezvous mode independent of the eager value used. This is however not necessarily what one observes.

Table III shows the execution time obtained for an eager limit of 12KB (default value) and 48 KB for a message length of 64KB using the NetPipe benchmark and for binary tree broadcast operation using SkaMPI for the same message length. As expected, changing the eager limit to 48KB did not have an impact on the NetPipe benchmark, which operates one message at a time. In a collective operation however, increasing the eager limit lead to significant performance degradation. It is beyond the scope of this paper to detail the reasons for this behavior, the main message is however that a parameter setting for high performance computing. This work can be extended by including other collective operations and network interconnects. Since the ultimate goal is however to make a statement about performance improvements of an entire application, an interesting challenge will be how to incorporate multiple message length in this analysis.

Acknowledgments. Partial support for this work was provided by the National Science Foundation under Award No. S12-1148052 and S12-1339763. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES