LogGP in Theory and Practice - An In-depth Analysis of Modern Interconnection Networks and Benchmarking Methods for Collective Operations

Torsten Hoefler, Timo Schneider, Andrew Lumsdaine

Open Systems Laboratory, Indiana University, 150 S Woodlawn Ave, Bloomington, IN 47405 USA, {htor,timoschn,lums}@cs.indiana.edu

Abstract

Accurate measurement and modeling of network performance is important for predicting and optimizing the running time of high-performance computing applications. Although the LogP family of models has proven to be a valuable tool for assessing the communication performance of parallel architectures, non-intrusive LogP parameter assessment of real systems remains a difficult task. Based on an analysis of accuracy and contention properties of existing measurement methods, we develop a new low-overhead measurement method which also assesses protocol changes in the underlying transport layers. We use the gathered parameters to simulate LogGP models of collective operations and demonstrate the errors in common benchmarking methods for collective operations. The simulations provide new insight into the nature of collective algorithms and their pipelining properties. We show that the error grows linearly with the system size.

Key words: LogP, LogGP, Network Modeling, Benchmarking, Simulation, Collective Operations

1 Introduction

Network performance prediction is very important for assessing the quality of parallel algorithms and predict their runtime on future architectures. We propose a low-overhead measurement method to assess LogGP parameters accurately and to detect network protocol changes which are often introduced by high-level communication libraries, such as MPI [1]. We then use our results to simulate collective algorithms to show possible sources of errors when such operations are benchmarked.

Various network models have been proposed in the past. One class of models targets specific hardware, network architectures, or the shared memory paradigm [2,3]. Other models are general-purpose and attempt to be architecture-independent, for example the PRAM [4,5], BSP [6], or the LogP [7] models. Several studies have compared the accuracy of those models [8,9]. In general, it seems that the LogP model family bridges the complexity of real-world interconnection networks and the usability of an abstract network model fairly well.

The LogP model family comprises the original model proposed by Culler et al. as well as a number of different extensions intended to improve prediction accuracy by considering different network effects. The LogGP model by Alexandrov et al. [10] models large messages with the new G parameter that indicates bulk-transfer rates. Using the LogGP model instead of the simpler Hockney model [11] separates the CPU and network times and thus enables detailed measurement of the communication overheads. This distinction between CPU overhead and network parameters enables researchers to model overlap of communication and computation efficiently. We use this ability to assess the overlap potential of different network interconnect architectures and to optimize the implementation of our non-blocking collective operations library LibNBC [12]. The models of the LogP family have been used by different research groups to derive new algorithms for parallel computing, predict the performance of existing algorithms, or prove an algorithm's optimality [13–18]. While the derivation of new algorithms and the proof of optimality can be done without actual parameter values, accurate measurement methods for each parameter are necessary to predict performance of the parallel algorithms. LogGP can also be used to adaptively change algorithms to react to parameter changes, for example, in wide-area networks. Another possible application is message scheduling, i.e., schedule messages across multiple, probably homogeneous, network interfaces to minimize the cumulative transmission time. All those methods need to assess the model parameters while the application runs. Such applications require a low-overhead measurement method that avoids network flooding or saturation.

One difficulty with simple direct parameter measurement is the fact that most modern communication systems use message-size dependent protocols to optimize communication (e.g., [19,20]). Small messages are often copied to prepared (or pre-registered) local send or remote receive buffers to speed up the communication. This method is commonly named *eager protocol*. Larger messages can not be copied directly to the receiver and are handled with a synchronizing protocol, often called *rendezvous protocol*. More protocol types can be introduced by the developer of the communication subsystem as needed. The switch between those protocol types is usually transparent to the user. Our measurement method is able to recognize protocol switches automatically because changes in the transmission parameters can be detected. We compute

individual parameter sets for all identified protocol ranges.

1.1 Background and Related Work

The LogGP model as described by Alexandrov et al. in [10] consists of the following parameters: **L** is an upper bound on the *Latency* of a send operation from one processor to another. **o** is the *overhead*, i.e., the time that the host processor is engaged in the transmission or reception of a message. **g** is the *gap* between two consecutive messages. It defines the minimum time-interval between two message sends or receptions. **G** is the *Gap* per byte for long messages. It defines the time needed to transmit a single byte for the bulk-transfer of long messages. **P** is the number of involved *Processors*. Several studies, including ours, often differentiate between in o_s and o_r as send and receive overhead, however, when citing other works we sometimes use o to adhere to the original notation. In the following, we review some of the well-known approaches, discussing the sources of error in those approaches.

The first measurement method for the LogP model was proposed by Culler et al. in [21]. Culler splits the overhead o into o_s on the sender side and o_r on the receiver side. To assess o_s, the time to issue a small number (n) of send operations is measured and divided by n. We note that this technique could be problematic on modern architectures which tend to copy messages to a temporary buffer for later transmission (e.g., TCP/IP sockets or Message Passing Interface (MPI) eager messages). As a result, the measurement of o_s would depend on n and only be accurate for very large number of messages when all buffers are filled. However, it is clear that a large number of messages would measure g. The receive overhead o_r is measured by sending multiple messages with a delay, bigger than the round trip time (RTT), after each send. This enables a computation of o_r based on the delay and the measured o_s. However, this makes the accuracy o_r dependent on the accuracy of o_s and introduces additional errors. The g parameter is simply benchmarked by flooding the network with many small messages and dividing the time by the number of messages. Finally, L can be computed from the other parameters with $L = {}^{RTT}/_2 - o_r - o_s$. A similar approach was used by Ianello et al. in [22], to assess the LogP parameters for Myrinet.

Kielmann et al. introduces the new pLogP (parametrized LogP) model and presents techniques to measure the parameter in [23]. He also shows how to reduce pLogP parameters to LogGP parameters. The send overhead o_s is simply measured as the time to send a single message. This measurement is influenced by caching effects similar to the original idea in [21]. The receive overhead o_r is defined as the time to copy the message from the receive buffer, which neglects the time in which the system is busy to receive the message to

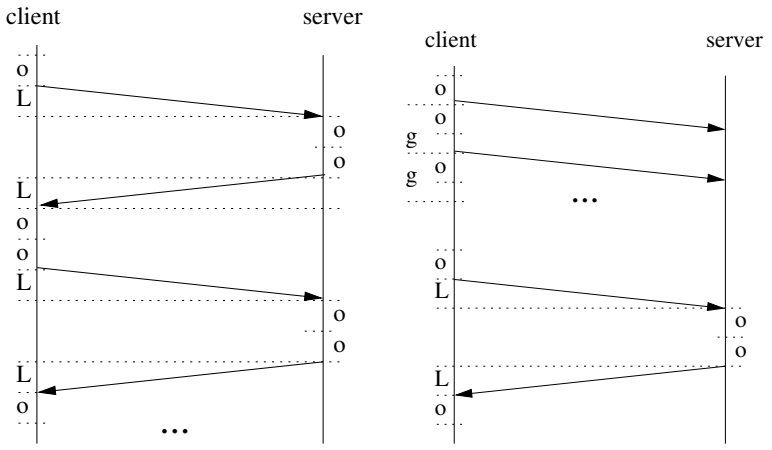
the temporary buffer (cf. TCP/IP sockets). The gap g is measured by sending n messages to a peer and the peer sends a single message back after it has received all n messages. The time between the first send and the reception of the final answer divided by n is reported as g. The n messages and a single reply message need $(n \cdot g + L) + (L + g)$ in pLogP and an error of $(2L + g)/n$ is made if one simply divides this sum by n. The impact of this error can be reduced if n is large enough so that $(2L + g)/n \ll g$. If we try to reach 1% accuracy, we need $n > (2L + g)/(g \cdot 0.01)$, which results in $n > 19640$ if we use the LogGP parameters for TCP/IP (see Section 3). The pLogP latency L is simply computed from the RTT of a zero-byte message $L = (RTT(0) - 2g(0))/2$. The fact that every parameter depends on the message size improves the accuracy of the model but also significantly complicates the model and the predictions.

The most recent work, and the only one that assesses all LogGP parameters besides L, was proposed by Bell et al. in [24]. The parameter o_s is measured with a delay between message sends. This delay d is adjusted until $d + o$ fits $g + (s - 1)G$ for a specific message size s exactly. This requires multiple measurement steps to adjust the correct d. The send overhead o_s is computed via $g + (s-1)G - d$, which relies on the correctness of g and G. The method to assess o_r is similar to the method in [21], but the transmission of the answer is delayed on the receiver side. A technique similar to pLogP is used to measure g, which suffers from the same problem that a huge number of packets (n) must be sent to get an accurate measurement and the network is effectively flooded. L can not be measured because modern networks tend to start the message transmission before the CPU is done (L is started before o ends). Bell et al. introduce the more practically oriented end-to-end latency (EEL) which denotes the RTT for a very small packet.

All proposed schemes use only single message sizes to derive parameters, which could be inaccurate for some networks that show anomalies at specific message sizes. The second problem with some methods is that the accuracy depends on the number of sent messages, which makes network flooding necessary to achieve good predictions. However, flooding causes unnecessary network contention and should not be used during application runs. We propose a new measurement scheme that avoids flooding as much as possible and delivers accurate parameters. The following section describes the working principle of our new measurement method.

2 Accurately Measuring LogGP Parameters

Modeling network transmissions can be helpful to optimize communication during application run time. However, most environments are changing under different loads (e.g., wide-area networks or oversubscribed communication



(a) Ping Pong (b) Ping Ping

Fig. 1. Different microbenchmarking schemes.

networks). Thus, network parameters are changing over time and must be reassessed during the runtime of the application. Run-time assessment should not introduce significant overhead. We describe a new low-overhead LogGP parameter assessment method, based on our findings in [25], in the following.

2.1 System Constraints

Many parallel systems do not have an accurately synchronized clock with a resolution that is high enough to measure network transmissions (in the order of microseconds). As a result, developers of network benchmarks must perform all time measurements locally on each machine. Many benchmarks (e.g., Netpipe, Pallas Micro Benchmarks (PMB [26])) use a so-called ping-pong scheme to benchmark message round trip times (RTT). This scheme uses two hosts: the client that initiates the communication and measures RTTs, and the server that mirrors all received packets back to the client. This common scheme is shown in Figure 1(a). Other schemes, such as ping-ping (originally mentioned in [26]), shown in Figure 1(b), can be used to measure the performance of multiple consecutive message sends. However, one has to be aware that a ping-ping with many packets quickly saturates the network and introduces contention. An additional degree of freedom for the benchmark is a ping-ping scheme with an artificial delay between each message send. Such a delay can be achieved by a fixed-size computation on the CPU.

2.2 Definitions

We combine the previously mentioned possibilities and use them to assess all LogGP parameters as unintrusively as possible. We introduce the notion

of the parametrized round trip time ($PRTT$) to define a specific parameter combination for the RTT. The possible parameters are the number of ping-ping packets (n), the delay between each packet (d) and the message size (s). A measurement result of a specific combination of n, d and s is denoted as *PRTT(n,d,s)*. The following subsections show that the notion of *PRTT(n,d,s)* is sufficient to assess all LogGP parameters accurately without network flooding or unnecessary contention.

The parametrized round trip time for a single ping-ping message without delay can be expressed in terms of the LogGP model as follows:

$$PRTT(1,0,s) = 2 \cdot (L + o_s + o_r + (s-1)G). \tag{1}$$

If we define the cumulative hardware gap $G_{all} = g + (s-1)G$, then n ping-ping messages can be modeled as (the original LogGP model defines $o < G_{all}$)

$$PRTT(n,0,s) = 2 \cdot (L + o_s + o_r + (s-1) \cdot G) + (n-1) \cdot G_{all} \ .$$

With (1), we get

$$PRTT(n,0,s) = PRTT(1,0,s) + (n-1) \cdot G_{all} \ . \tag{2}$$

This equation can easily be extended to the general case with a variable delay d as

$$PRTT(n,d,s) = PRTT(1,0,s) + (n-1) \cdot max\{o_s + d, G_{all}\} \ . \tag{3}$$

In the following sections, we explain how to use the parametrized roundtrip time to assess the LogGP parameters of different network interfaces.

2.3 Assessment of the Overheads o_s and o_r

Previous works have shown that the overhead might not be constant with varying message sizes. Thus, we define a method to accurately measure the overheads for every message size. If we rewrite Equation (3) to

$$\frac{PRTT(n,d,s) - PRTT(1,0,s)}{n-1} = max\{o_s + d, G_{all}\} \ ,$$

and choose d_G, such that $d_G > G_{all}$, we get

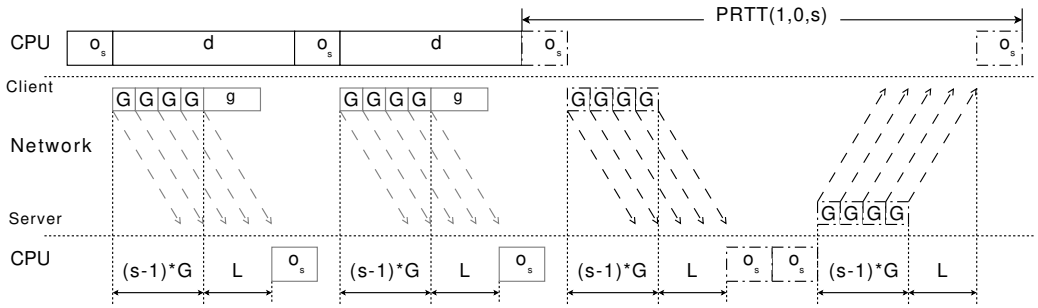


Fig. 2. Measurement Method for o for $n = 3$. This figure shows the components of $PRTT(n, d, s)$.

$$\frac{PRTT(n, d_G, s) - PRTT(1, 0, s)}{n - 1} = o_s + d_G \ . \tag{4}$$

This enables us to compute o from the measured $PRTT(n, d_G, s)$ and $PRTT(1, 0, s)$. We chose $PRTT(1, 0, s)$ for d_G to ensure that $d_G > G_{all}$. This assumption has been proven to be valid for all tested networks. However, if a network with a very low latency L and a very high gap g exists, one can fall back to $d_G = PRTT(2, 0, s)$ to guarantee $d_G > G_{all}$. We chose $PRTT(1, 0, s)$ to avoid unnecessary long benchmark times.

The measurement of o_s for $n = 3$ is illustrated in Figure 2. The entire figure represents a LogGP model for $PRTT(3, d_G, s)$ and it is easy to see that the last part is a simple $PRTT(1, 0, s)$. If we subtract $PRTT(1, 0, s)$ from $PRTT(3, d_G, s)$, we get $2d_G + 2o_s$ which equals to $(n-1)(d_G + o_s)$ (recall that $n = 3$ in our example) as shown in Equation (4).

This measurement method enables us to obtain a fairly accurate value of o_s for each message size s. It needs only a small number of messages (we used $n = 10$ in our tests) and thus does not saturate or flood the network to measure o_s. Furthermore, we are able to compute o_s directly from a single measurement and without depending on other LogGP parameters which would increase the measurement error. We do also not need to adjust d stepwise to fit other values. The measurement of o_r can be performed as proposed by Kielmann [23]. We are aware that this method will not produce accurate results. However, empirical experiments show that this method measures reasonable receive overheads for all networks that we investigated.

2.4 Assessment of the Gap Parameters g,G

Using Equation (2), we get a linear function of the form $f(s) = G \cdot (s-1) + g$:

$$G(s-1)+g=\frac{PRTT(n,0,s)-PRTT(1,0,s)}{n-1} \tag{5}$$

One could simply measure $PRTT(n,0,s)$ and $PRTT(1,0,s)$ for two different s and solve the resulting system of linear equations directly. However, several networks have anomalies or a huge deviation between different data sizes. Another problem is that this method would not allow us to detect protocol changes in the lower levels that influence the LogGP parameters.

Instead, we chose to measure $PRTT(n,0,s)$ and $PRTT(1,0,s)$ for many different s and fit a linear function to these values. The function value for $s=1$ is our g and the slope of this function represents our G.

We use the least squares method [27], which can be solved directly for the needed two degrees of freedom (g and G), to perform the fit. It provides us an accurate tool to assess g and G with multiple different message sizes and to detect protocol/parameter changes in the underlying transport layers (see Section 2.6). We use only a small n to benchmark every single message size. Thus, we do not need to flood or overload the network and our results are not influenced by anomalies for specific message sizes (like we experienced with TCP). Furthermore, we are able to use our method to detect changes in the underlying communication protocol, as described in Section 2.6. A graphical representation of our method with Open MPI over InfiniBand is shown in Figure 3.

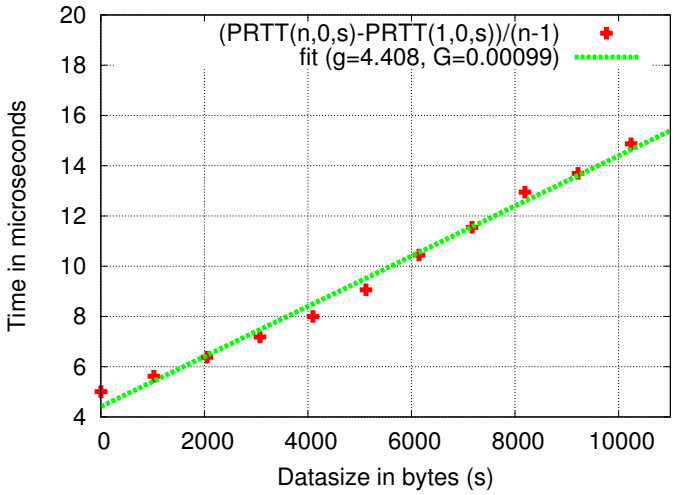


Fig. 3. Parameter Benchmark and Fit for Open MPI over InfiniBand.

2.5 Assessment of the Latency Parameter L

Bell et al. discussed the interesting phenomenon that the occurrence of L and o is not ordered. It happens on modern interconnect networks that o and a part of L overlap (e.g., some message processing is done after the sending of the message is started). This is due to the fact that the network developers strive to minimize the round trip time and try to move all the bookkeeping after the message is sent. This effect does not allow us to measure a useful L (L may even be negative in certain situations). We take a similar approach as [24] and report half of the round trip time of a small message as latency. We use $PRTT(1, 0, 1)/2$ for this purpose.

2.6 Detection of Protocol Changes

Modern network APIs are complex systems and try to deliver highest performance to the user. This requires to use different transport protocols for different message sizes. It is obvious that each transport protocol has its own unique set of LogGP parameters. The problem is that the network APIs aim to be transparent to the user and do often not indicate protocol switches directly. These facts can make LogGP benchmarks very inaccurate if one does not differentiate between the used transport modes. Our approach is to detect those protocol changes automatically and provide a different set of LogGP parameters for each transport type to the user.

We define the mean least squares deviation from measurement point k to l and the fit-function $f(s) = G \cdot s + g$ as

$$lsq(k, l) = \frac{\sum_{i=k}^{l} (G \cdot size(i) + g - val(i))^2}{l - k - 2} \ , \tag{6}$$

where $val(i)$ is the measured value at point i and $size(i)$ is the message-size at point i. We substract 2 in the denominator because we have 2 degrees of freedom for the solution of the least squares problem.

We take an x point look-ahead method and compare the mean least squares deviation of the intervals $[lastchange : current]$ with the deviation of the interval $[lastchange : current + 1]$, $[lastchange : current + 2]$, ..., $[lastchange : current + x]$. We define *lastchange* as the first point of the actual protocol (the point after the last protocol change, initially 0) and *current* as our current point to test for a protocol change. If *current* is the last measured value of a protocol, and a new protocol begins at $current + 1$, the mean least squares deviation rises from this point on. We consider the next x (typically 3-

5) points to reduce the effect of single outliers. If $lsq(lastchange, current + j)$ $\forall 1 \leq j \leq x$ is larger than $lsq(lastchange, current) \cdot pfact$, then we assume that a protocol change happened at *current*. The factor *pfact* determines the sensitivity of this method. Empirical studies unveiled that $pfact = 2.0$ was a reasonable value for our experiments. However, this factor is highly network dependent and further network-specific tuning may be necessary to detect all protocol changes accurately.

3 Applying the Method

We implemented our approach as a new *communication pattern* in the extensible open source Netgauge tool [28]. Netgauge is a modular network benchmarking tool that uses high-precision timers (e.g., the x86 RDTSC instruction [29]) to benchmark times accurately. An important difference between Netgauge and other tools like NetPipe [30], coNCePTuaL [31] or the Pallas Micro Benchmarks (PMB) [26] is that the Netgauge framework offers the ability to use MPI as infrastructure to distribute needed protocol or connection information for other low-level APIs (e.g. Sockets, InfiniBand, SCI, Myrinet/GM ...) or to benchmark MPI_Send/MPI_Recv itself. This capability is important for comparing low-level performance with MPI performance and enables the user to assess the quality and overheads of specific MPI implementations. Our LogGP communication pattern enables a detailed analysis of the introduced software overhead.

Transmission modules for MPI, TCP, UDP, InfiniBand, and several other networks are included in Netgauge and enable us to compare the performance of MPI with the underlying low-level API's performance. More low-level modules (e.g., SCI, Cell B.E.) are under development. The newest version of Netgauge supports LogGP parameter measurement as described in this paper. We followed the useful hints provided by Gropp et al. [32] to achieve reproducible and accurate benchmark results.

As explained in Section 2, the *PRTT(n,d,s)* is sufficient to derive the LogGP parameters. The choice of n is critical to benchmark performance (i.e., congestion avoidance) and accuracy. It should be as small as possible to enable fast measurements and still retain accuracy. We conducted several benchmarks with different n and used the least squares deviation of the parameter fit as base for our choice. Figure 4 shows the least squares deviations for different values of n with Open MPI on Gigabit Ethernet. We see that $n = 10$ results in the lowest average deviation on our available networks. Thus, we chose $n = 10$ as default in our implementation in Netgauge. The delay d is implemented as a busy loop that checks high-performance timers (e.g., RDTSC [29]) repeatedly until the d has elapsed. However, the choice of d is also not trivial because it

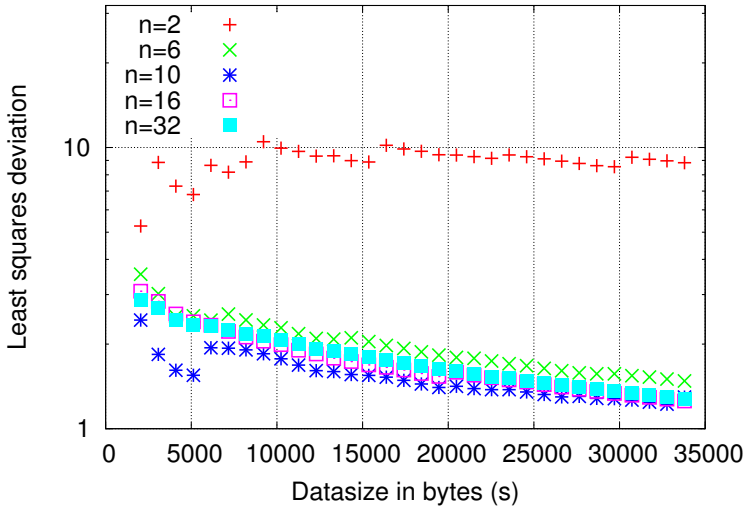


Fig. 4. Least squares deviation for different values of n.

should be small to ensure fast measurements but it must also be larger than G_{all}. We chose $d = PRTT(1, 0, s)$ in our implementation because we found that this usually fulfills these criteria. However, a check is added after every parameter calculation and a warning is printed if $G_{all} > d$. In case of a failure, the user can easily fall back to $d = PRTT(2, 0, s)$ which is guaranteed to fulfill the criteria, but increases the measurement time.

3.1 Results

We analyzed different interconnect technologies and parallel systems to evaluate their performance in the LogGP model. The evaluated systems are described in Table 1. All presented results are very system dependent and we suggest to re-run the benchmark for every system configuration. For example, the overhead is heavily influenced by the performance of the main CPU and the I/O subsystem. Our test-systems have very different CPU speeds and chipset configurations, thus, the results can not serve as a network interconnect comparison. It has to be ensured that the testing environment is exactly identical in order to compare network interconnects.

We benchmarked TCP over Gigabit Ethernet and MPICH2 1.0.3, SCI with NMPI 1.2, Single Data Rate (SDR) InfiniBand with Open MPI 1.1.2/openib, Double Data Rate (DDR) ConnectX InfiniBand with Open MPI 1.2.8, Myrinet 2000 with Open MPI 1.1.2/GM, Myrinet 10G with Open MPI 1.2.8/MX and Chelsio 10 Gigabit iWARP adapters with Open MPI 1.2.6/openib. The graphs for $G \cdot (s-1) + g$ (cf. Equation (5)) for several configurations are shown in Figure 5.

Transport	CPU	Additional Information
MPICH2	Opteron 246, 2GHz	MPICH2 1.0.3, BCM5704 GigE Network Chip
NMPI/SCI	Xeon 2.4GHz	NMPI-1.2, SCI-Adapter PSB66 D33x
OMPI/IB (SDR)	Opteron 244	Open MPI 1.1.2, OFED-1.0, Mellanox MT25208
OMPI/10G	Xeon 5160 3.0GHz	Open MPI 1.2.6, Chelsio T3 iWARP, cxgb3_0
OMPI/IB (DDR)	Xeon L5420 2.5GHz	Open MPI 1.2.8, OFED-1.3, Mellanox ConnectX
OMPI/GM	Athlon MP 1.4GHz	Open MPI 1.1.2, GM 2.0.23, Myrinet 2000
OMPI/MX	Xeon L5420 2.5GHz	Open MPI 1.2.8, MX 1.4.3, Myrinet 10G

Table 1
Details about the test systems.

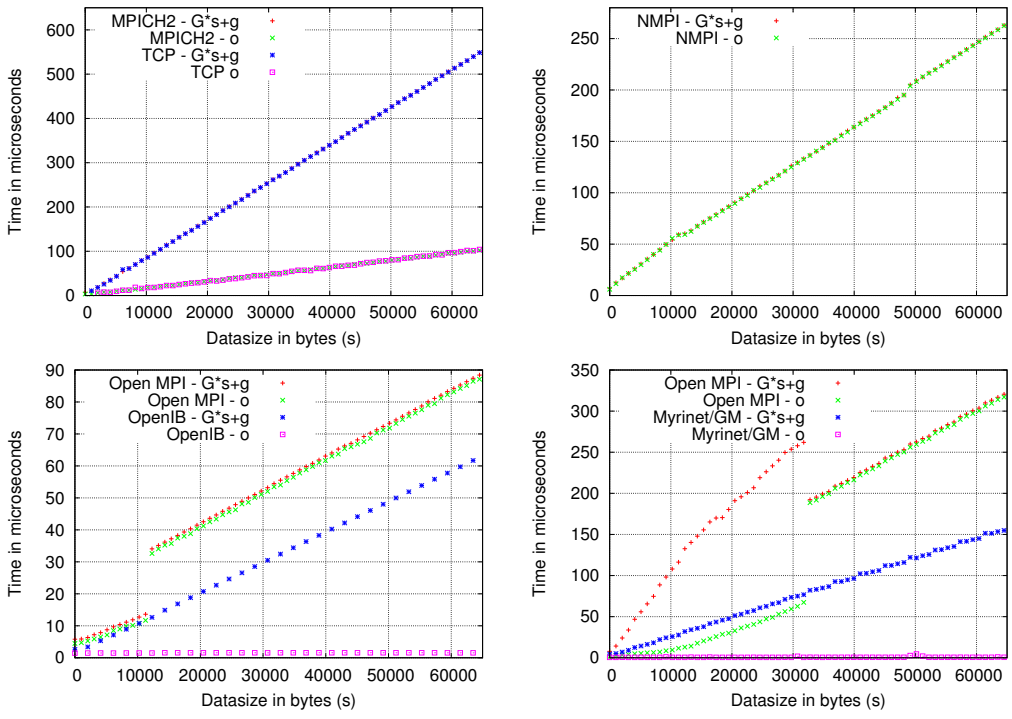


Fig. 5. Measurement results for GigE/TCP, SCI, InfiniBand (SDR) and Myrinet/GM (in this order). The graphs show $f(s) = G \cdot (s-1) + g$, such that the slope indicates G and $f(1) = g$ and the CPU overhead. The parameters of the fitted functions for g and G can be found in Table 2.

Table 2 shows the numerical results for the LogGP measurements on the different systems. Blocking communication was used to measure those values. The TCP results show that o_s, g and G are nearly identical for MPICH2 and RAW TCP (they are not distinguishable in the diagram because they lie practically on the same line). We also see that o_s is not constant as assumed in the LogGP model but has a linear slope. We encounter no protocol change for TCP in the interval $[1, 65536]$ bytes. The SCI results indicate that the implementation uses polling to send messages because $o_s \approx G_{all}$. We see three different protocol regions for the MPI implementation NMPI (cf. Table 2).

InfiniBand also shows interesting behavior. The Open MPI openib component uses polling to send or receive messages. A protocol change at approximately 12 kiB leads to a large increase of g. This is due to the rendezvous protocol which introduces an additional RTT of a small status message, which costs $\approx 2L + 4o$ in LogGP, before the actual transmission begins. The blocking MPI_Send charges this to g because it has to wait until a message is sent before it sends the next one. G is mainly identical across all message-sizes. The low-level openib API has a small g and shows no protocol change. The low-level overhead to post a send request is independent of the message size. Open MPI introduces additional CPU overhead which is due to the local copy (for eager send) or InfiniBand's memory registration.

Myrinet/GM seems to use interrupts for small messages ($o < G_{all}$) and polling for messages larger than 32 kiB ($o \approx G_{all}$). The protocol change is again clearly visible in the graph and is correctly recognized by our method. The low-level API delivers a slightly lower G and a similar g in the measured interval. The overhead o of the GM API is constant (as it was for InfiniBand).

4 Simulating Collective Algorithms with LogGP

In order to show the importance of accurate measuring of LogGP parameters, we use our parameters for different interconnection networks to show the behavior of different collective algorithms. It has been shown that the LogGP model can be used to provide bounds on the communication time of collective algorithms. However, modeling a parallel application in this way is often tedious and error-prone. A binomial tree communication pattern for example has a complicated communication structure with regards to the finishing time of every process and a closed form for the runtime estimation can not be found easily. We show how to achieve accurate predictions using a discrete event-driven simulation methodology for the LogGP model.

Transport	**Protocol Interval (bytes)**	**L** (μs)	**o(1)** (μs)	**g** (μs)	**G** ($\mu s/b$)
MPICH2/TCP	$1 \leq s$	45.74	3.46	0.915	0.00849
NMPI/SCI	$1 \leq s < 12289$	5.48	6.10	7.78	0.0045
	$12289 \leq s$	5.48	6.10	13.34	0.0037
OMPI/IB (SDR)	$1 \leq s < 12289$	5.96	4.72	5.14	0.00073
	$12289 \leq s$	5.96	4.72	21.39	0.00103
OMPI/10G	$1 \leq s < 12289$	10.97	5.05	5.00	0.0023
	$12289 \leq s$	10.97	5.05	42.00	0.00101
OMPI/IB (DDR)	$1 \leq s < 12289$	2.50	1.49	1.08	0.00067
	$12289 \leq s$	2.50	1.49	11.90	0.00058
OMPI/GM	$1 \leq s < 32769$	10.53	1.27	9.44	0.0092
	$32769 \leq s$	10.53	1.27	52.01	0.0042
OMPI/MX	$1 \leq s < 32769$	2.98	1.48	2.00	0.00081
	$32769 \leq s$	2.98	1.48	33.10	0.00109

Table 2
LogGP Parameters for different Transport Protocols

4.1 Simulation Methodology

In this section we present a brief description of our simulator design which is similar to [33]. We model a parallel algorithm with a set of partially dependent send, receive and local transformation operations. Dependencies among them, e.g., a send of a data item can only start after a local transformation on this item is finished, are modeled as required by the parallel algorithm. Our simulator models each process with a queue of outstanding operations. Communication is by default modeled as non-blocking so that posted sends and receives finish independently (this eliminates deadlocks). Each send and receive is added to the queue and removed from the queue when matched. Receives are removed from the queue when a matching send arrived and a send is either removed immediately (eager protocol) or after it matched a receive (rendezvous protocol). Local times are updated according to the Lamport clock condition [34]. The simulator also generates a time-line diagram which can be used to visualize the simulation.

For example, consider a program where P processes communicate in a double-ring. Process i receives one data item from process $(i-1 \mod P)$ and sends to process $(i+1 \mod P)$ (forward ring) and, at the same time, it receives a

data item from process $(i+1 \mod P)$ and sends to process $(i-1 \mod P)$ (backward ring). The two data items are originating at process 0. Figure 6 shows the timeline diagram (simulation output) for this communication with $P = 4$. In this example, it becomes obvious why it is not easy to design explicit

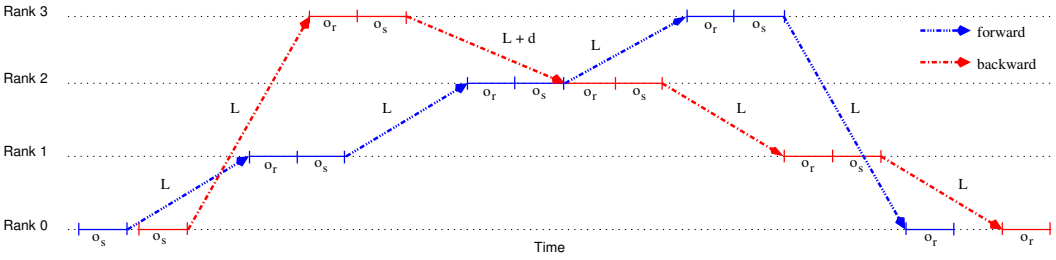


Fig. 6. LogGP timeline of double-ring communication pattern.

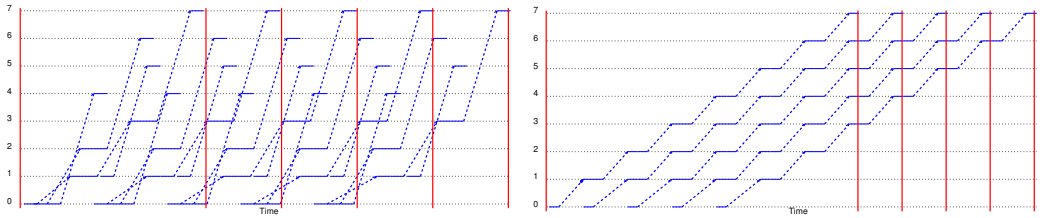
mathematical models for the running time of such algorithms. The problem in this example is that the message from rank 3 and the message from rank 1 arrive at the same times at rank 2 and want to consume CPU resources. Thus, the send from rank 3 is delayed until the CPU becomes available which delays the global communication accordingly. Mathematically modeling such effects for more complicated patterns is tedious and error-prone.

4.2 Simulation Results

Based on the LogGP model, we argue in [35] that many of the current benchmarking strategies that are used for benchmarking MPI collective operations deliver inaccurate results. In this work, we substantiate this claim with simulation results that use real-world network parameters. Based on the availability of an accurate LogGP benchmarking method and our simulation framework, we are now able to assess the relative inaccuracy of common benchmarking schemes. A common measurement mistake is to perform the same collective operation n times in a loop and then report the total time divided by n. We argued in [35] Section 1.2 that such measurements often underestimate the time of a single collective operation due to message pipelining effects.

In the following, we demonstrate the effect with two different broadcast algorithms, a binomial tree and pipelined broadcast. Both algorithms are used in practice depending on the proportion of P and the message size s (cf. [9]). Figure 7 shows the influence of pipelining on a measurement with $P = 16$ and $n = 5$.

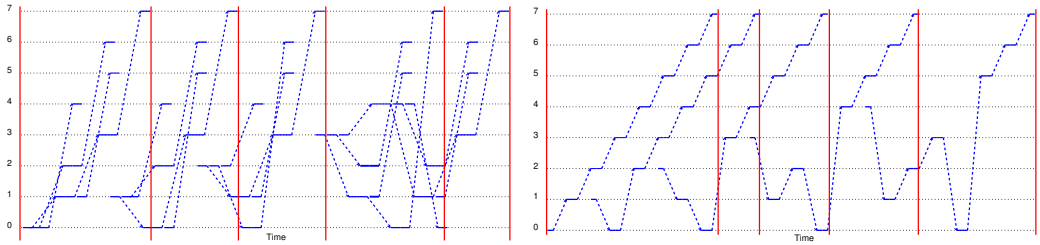
An intuitive fix, rotating the root, was proposed to avoid such pipelining techniques. However, we show that this fix does not solve the problem. A visualization of our two algorithms with root-rotation is shown in Figure 8. It only mitigates the measurement error and, for some algorithms, the asymptotic relative



(a) Binomial Tree (b) Pipeline

Fig. 7. LogGP Communication graphs for a common benchmarking scheme.

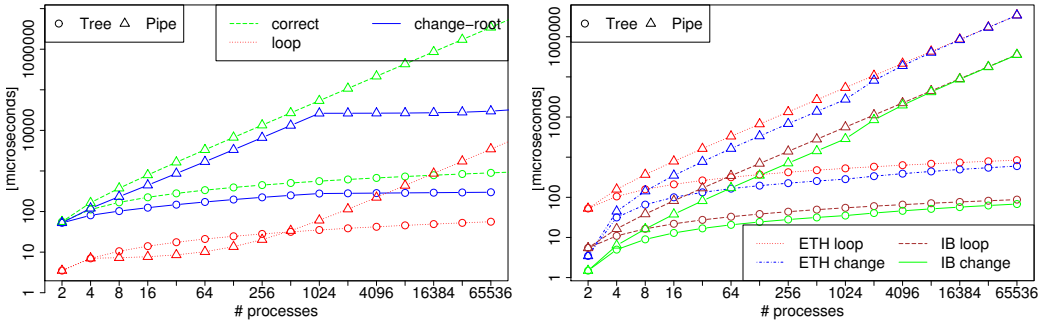
error in n is identical to the error without root rotation. Thus, we conclude that root rotation does not fix the problem. If we assume a fixed n ($n = 1000$



(a) Binomial Tree (b) Pipeline

Fig. 8. LogGP Communication graphs for change-root benchmarking scheme.

in our experiments) and we vary P, then we see that the measurement mistake accumulates depending on the network type as shown in Figure 9(a). The sim-



(a) Absolute Comparison for Ethernet (b) Error for Ethernet and InfiniBand

Fig. 9. Comparison of the different benchmarking schemes for two different broadcast algorithms: binomial tree and pipeline. The correct benchmarking scheme is compared with the simple loop benchmark (`loop`) and a change-root scheme (`change`) for Ethernet (`ETH`) and InfiniBand (`IB`).

ple loop scheme underestimates the actual broadcast time by several orders of magnitude in case of the pipelined broadcast. The binomial tree broadcast is also underestimated significantly. A change-root scheme mitigates the difference, but does not reach the quality of a synchronizing benchmark as proposed in [35]. The root rotation also fails when the number of measurements/rotations (n) is smaller than the number of processes. For example, the pipelined

scheme measures a quasi-constant time for $P > n$. We also see in Figure 9(a) that a wrong benchmarking scheme could lead to wrong decisions if the choice of algorithms depends on a previously benchmarked model (cf. [9]) because pipelined broadcasts erroneously seem superior to tree-based broadcasts up to 512 processes.

Figure 9(b) shows the scaling of the measurement error (the difference between correct and benchmarked result) for Ethernet and InfiniBand. We see that the error is much less with InfiniBand, however, the asymptotic behavior is similar. We also see that the error depends on the used collective algorithm. The error for binomial trees scales logarithmically with the number of processes while the pipelined algorithm has a linear error. Based on those results, we strongly suggest to benchmark each collective operation separately and not in a loop.

5 Conclusions and Future Work

We compared well known Log(G)P measurement methods and derived a new accurate LogGP parameter measurement scheme. Our method is able to detect protocol changes in the underlying communication subsystem. An open source implementation within the Netgauge framework is available at `http://www.unixer.de/research/netgauge/`

The precision with which we are able to assess LogGP parameters with our approach suggests further refinements to the LogGP models and to the measurement process. In particular, the overheads for receiving and sending (o_r and o_s respectively) are assumed to be constant and the same in current models. However, previous results and preliminary experiments with our framework have shown that this is often not the case. Models and corresponding measurement to account for this difference are the subject of future work.

We also presented a discrete event simulation framework for LogGP which uses the benchmarked values and simulates real-world communication systems. We utilized the simulation to assess the absolute error that is done by performing collective communication benchmarks in a loop. Our simulations help us to gain a better understanding of effects in the network (such as pipelining or endpoint contention). Our results show that simple techniques that have been proposed to increase the accuracy of benchmarking techniques fail at large scale. We showed that the absolute measurement error grows linearly or logarithmically with the system size for a pipelined and tree-based broadcast respectively.

We plan to investigate opportunities to simulate the influence of operating system noise to large-scale systems with our LogGP simulation environment.

Furthermore, we work on more detailed application simulation methods.

Acknowledgments

This work was supported by the Department of Energy project FASTOS II (LAB 07-23) and a grant from the Lilly Endowment.

References

[1] MPI Forum, MPI: A Message-Passing Interface Standard. Version 2.1, `www.mpi-forum.org` (September 4th 2008).

[2] J. R. Larus, S. Chandra, D. A. Wood, CICO: A Practical Shared-Memory Programming Performance Model, in: Ferrante, Hey (Eds.), Workshop on Portability and Performance for Parallel Processing, John Wiley & Sons, Southampton University, England, July 13 – 15, 1993.

[3] P. G. Gibbons, Y. Matias, V. Ramachandran, Can a shared memory model serve as a bridging model for parallel computation?, in: ACM Symposium on Parallel Algorithms and Architectures, 1997, pp. 72–83.

[4] S. Fortune, J. Wyllie, Parallelism in random access machines, in: STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing, ACM Press, 1978, pp. 114–118.

[5] R. M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. Leeuwen (Ed.), Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity, Elsevier, Amsterdam, 1990, pp. 869–941.

[6] L. G. Valiant, A bridging model for parallel computation, Commun. ACM 33 (8) (1990) 103–111.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: towards a realistic model of parallel computation, in: Principles Practice of Parallel Programming, 1993, pp. 1–12.

[8] G. Bilardi, K. T. Herley, A. Pietracaprina, BSP vs LogP, in: SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, ACM Press, 1996, pp. 25–32.

[9] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, J. J. Dongarra, Performance Analysis of MPI Collective Operations, in: 4th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 05), Denver, CO, 2005.

[10] A. Alexandrov, M. F. Ionescu, K. E. Schauser, C. Scheiman, LogGP: Incorporating Long Messages into the LogP Model, Journal of Parallel and Distributed Computing 44 (1) (1995) 71–79.

[11] R. Hockney, The communication challenge for MPP: Intel Paragon and Meiko CS-2, Parallel Computing 20 (3) (1994) 389–398.

[12] T. Hoefler, A. Lumsdaine, W. Rehm, Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI, in: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 2007.

[13] M. Bernaschi, G. Iannello, Quasi-Optimal Collective Communication Algorithms in the LogP Model, Tech. rep., University of Naples (DIS) (03 1995).

[14] D. E. Culler, A. Dusseau, R. Martin, K. E. Schauser, Fast Parallel Sorting under LogP: from Theory to Practice, in: Proceedings of the Workshop on Portability and Performance for Parallel Processing, Wiley, Southampton, England, 1993.

[15] T. Hoefler, L. Cerquetti, T. Mehlan, F. Mietke, W. Rehm, A practical Approach to the Rating of Barrier Algorithms using the LogP Model and Open MPI, in: Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPP'05), 2005, pp. 562–569.

[16] G. Iannello, Efficient Algorithms for the Reduce-Scatter Operation in LogGP, IEEE Trans. Parallel Distrib. Syst. 8 (9) (1997) 970–982.

[17] R. M. Karp, A. Sahay, E. Santos, Optimal Broadcast and Summation in the LogP Model, Tech. rep., Berkeley, CA, USA (1992).

[18] M. Bernaschi, G. Iannello, Collective communication operations: experimental results vs. theory, Concurrency - Practice and Experience 10 5 (1998) 359–386.

[19] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004.

[20] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (1996) 789–828.

[21] D. Culler, L. T. Liu, R. P. Martin, C. Yoshikawa, LogP Performance Assessment of Fast Network Interfaces, IEEE Micro.

[22] G. Iannello, M. Lauria, S. Mercolino, LogP performance characterization of Fast Messages atop Myrinet, in: In Proceedings of 6th Euromicro Workshop on Parallel and Distributed Processing, 1998, pp. 395–401.

[23] T. Kielmann, H. E. Bal, K. Verstoep, Fast Measurement of LogP Parameters for Message Passing Platforms, in: IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, Springer-Verlag, London, UK, 2000, pp. 1176–1183.

[24] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, K. Yelick, An evaluation of current high-performance networks, in: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IEEE Computer Society, Washington, DC, USA, 2003, p. 28.1.

[25] T. Hoefler, A. Lichei, W. Rehm, Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks, in: Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium, IEEE Computer Society, 2007.

[26] Pallas GmbH, Pallas MPI Benchmarks - PMB, Part MPI-1, Tech. rep., Pallas GmbH (2000).

[27] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, Philadelphia, 1996.

[28] T. Hoefler, T. Mehlan, A. Lumsdaine, W. Rehm, Netgauge: A Network Performance Measurement Framework, in: High Performance Computing and Communications, Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007, Proceedings, Vol. 4782, Springer, 2007, pp. 659–671.

[29] Intel Corporation, Intel Application Notes - Using the RDTSC Instruction for Performance Monitoring, Tech. rep., Intel (1997).

[30] D. Turner, X. Chen, Protocol-Dependent Message-Passing Performance on Linux Clusters, in: Cluster '02: Proceedings of the IEEE International Conference on Cluster Computing, IEEE Computer Society, Washington, DC, USA, 2002, p. 187.

[31] S. Pakin, Reproducible Network Benchmarks with coNCePTuaL., in: M. Danelutto, M. Vanneschi, D. Laforenza (Eds.), Euro-Par 2004 Parallel Processing, 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Proceedings, Springer, 2004, pp. 64–71.

[32] W. Gropp, E. L. Lusk, Reproducible Measurements of MPI Performance Characteristics, in: Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, London, UK, 1999, pp. 11–18.

[33] R. Rugina, K. E. Schauser, Predicting the Running Times of Parallel Programs by Simulation, in: 12th International Parallel Processing Symp., 1998, p. 654.

[34] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565.

[35] T. Hoefler, T. Schneider, A. Lumsdaine, Accurately Measuring Collective Operations at Massive Scale, in: Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS), Workshop PMEO, 2008.