Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks

Torsten Hoefler,[1,2] Andre Lichei,[1] and Wolfgang Rehm[1]

[1]Dept. of Computer Science
Technical University of Chemnitz
Strasse der Nationen 62
Chemnitz, 09107 GERMANY
{htor,lica,rehm}@cs.tu-chemnitz.de

[2]Open Systems Laboratory
Indiana University
501 N. Morton Street
Bloomington, IN 47404 USA
htor@cs.indiana.edu

Abstract

Network performance measurement and prediction is very important to predict the running time of high performance computing applications. The LogP model family has been proven to be a viable tool to assess the communication performance of parallel architectures. However, non-intrusive LogP parameter assessment is still a very difficult task. We compare well known measurement methods for Log(G)P parameters and discuss their accuracy and network contention. Based on this, a new theoretically exact measurement method that does not saturate the network is derived and explained in detail. Our method only uses benchmarked values instead of computed parameters to compute other parameters to avoid propagation of first-order errors. A methodology to detect protocol changes in the underlying communication subsystem is also proposed. The applicability of our method and the protocol change detection is shown for the low-level API as well as MPI implementations of different modern high performance interconnection networks. The whole method is implemented in the tool Netgauge and it is available as open source to the public.

1 Introduction

Network performance prediction is very important to assess the quality of parallel algorithms. We propose a low-overhead measurement method and implementation to assess LogGP parameters accurately and to detect network protocol changes which are often introduced by high-level communication libraries, such as MPI [30, 31].

1-4244-0910-1/07/$20.00

Different network models have been proposed in the past. Many are used to model a specific hardware or network architecture [26, 6] or the shared memory paradigm [25, 12]. There are also some general-purpose parallel models which try to stay architecture independent like PRAM [10, 22], BSP [39], C^3 [16] or the LogP [7] model. Several studies compare the accuracy of those models [28, 15, 4, 37, 17]. The LogP model family, originally proposed by Culler et al., has been found to be the most accurate model for many modern interconnection networks.

The different LogP model extensions aim at improving the prediction accuracy of the standard LogP model by taking different network effects into consideration. The LogGP model by Alexandrov et al. [1] models large messages with the new G parameter that indicates bulk-transfer rates. The LoGPC model by Moritz et al. [33] discusses the effects of network contention in the LogGP model. Synchronization overhead during the sending of large messages in high-level communication libraries such as MPI is modeled in the LogGPS model. The LogfP model [19] adds the new parameter f, which represents the number of consecutive small messages that can be sent for "free", to address the hardware parallelism (e.g., pipelining, super-scalar principles) in current high-performance networks like InfiniBand™. All those different models can be combined to predict the performance of a specific network. We decided to use the LogGP model for this work because it has been proven to be accurate and it is general enough to keep the applicability of our measurement method for many networks.

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 [3, 9, 18, 20, 23, 29]. While the derivation of new algorithms and the proof of op-

timality can be done without the real parameter values, accurate measurement methods for the single parameters are necessary to predict performance of algorithms or message transmission processes.

Network models can also be used to modify algorithms in runtime-changing environments adaptively. This is very important for wide-area networks as they are typically used in grid computing. Another application field is multi-NIC 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 during the running time of the application. This makes a low-overhead measurement method for the LogGP parameters necessary (a method that avoids network flooding or saturation).

Another pitfall for the parameter measurement is the fact that most modern communication systems use message-size dependent protocols to optimize communication (e.g., [27, 11, 13]). Small messages are often copied to prepared (e.g., preregistered in case of InfiniBand™ cf. [32]) local send or remote receive buffers to speed up the communication. This method is commonly named "eager protocol". Larger messages can not be copied to a buffer on the receiver side (because there may not be enough space), and a local copy would introduce too much overhead. Those messages force a synchronization, and the protocol type is 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, i.e., he does not realize it explicitly. Our method is able to recognize protocol switches automatically since changes in the message transmission times can be detected. We compute own parameter sets for all identified protocol ranges.

The following section describes related work to assess LogGP parameters and discusses positive and negative effects of the proposed methods. Section 3 introduces our new low-overhead measurement approach and its implications to the accuracy of the resulting LogGP parameters. Measurement results for different low-level interfaces and MPI implementations and a short comparison are presented in Section 4. The last Section 5 summarizes our contribution and points out future work in this area.

2 Existing Approaches and Related Work

The LogGP model as described by Alexandrov et al. in [1] consists of the following parameters: **L** is an upper bound on the *L*atency of a send operation from one processor to another. **o** is the *o*verhead, i.e., the time that the host processor is engaged in the transmission or reception of a message and can not perform other operations. **g** is the *g*ap between two consecutive messages. It defines the minimum time-interval between two message sends or receptions. **G** is the *G*ap per byte for long messages. It defines the needed time to transmit a single byte for the bulk-transfer of long messages. **P** is the number of involved *P*rocessors.

Previous works used different strategies and changes of the original model to assess the single parameters as accurately as possible. We discuss the well-known approaches in the following.

The first measurement method for the LogP model has been proposed in [8] by Culler et al., the author of the original LogP model. He differentiates between o_s on the sender side and o_r on the receiver side which complicates the model slightly. To assess o_s, he measured the time to issue a small number (n) of send operations and divided it by n. This could be problematic on modern architectures, because they tend to copy the message to a temporary buffer and send it later (e.g. TCP). This would make the measurement of o_s depend on n and only realistic for very large n when all buffers are filled. But this is not possible because a large number of n would benchmark g. Culler et al. use a delay between messages that is larger than a single round trip time (RTT) to assess o_r, based on the measurement of o_s, which makes the result dependent on the accuracy of o_s and introduces second-order 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 $L = RTT/2 - o_r - o_s$.

A second, similar approach was used by Ianello et al. in [21]. He uses similar techniques to assess the LogP parameters for Myrinet.

Kielmann et al. used heavy model changes and proposed a solution to assess parameters for his pLogP (parametrized LogP) model in [24]. He uses the time for a single send operation to assess o_s. This could be influenced by caching effects similar to the original idea in [8]. He defines o_r as the time to copy the message from the receive buffer. This clearly neglects the time in which the system is busy to receive the message to the temporary buffer (cf. TCP). The g assessment sends n messages to a peer and the peer sends a single message back after it received n. The time between the first send and the reception of the final answer divided by n is used as g. Those n messages and a single reply message need $(n \cdot g + L) + (L + g)$ in pLogP. 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 that $(2L + g)/n << g$. If we try to reach 1% accuracy, we need $n > (2L + g)/(g \cdot 0.01)$, which is 19640 for the LogGP parameters gained for TCP (see Section 4). L is 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 complicates the model and the predictions fairly.

The latest work, the only one that assesses all LogGP pa-

rameters besides L, was proposed by Bell et al. in [2]. 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. Now, 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 [8], but he delays the transmission of the answer on the receiver side. He uses a similar technique as Kielmann to measure g, which suffers from the same problem that he has to send a huge number of packets (n) 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 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.

3 Low Overhead Parameter Measurement

We describe a new low-overhead LogGP parameter assessment method in the following. We implemented our approach as a new "communication pattern" in the extensible open source Netgauge tool [34]. Netgauge is a modular network benchmarking tool that uses high-precision timers to benchmark network times. The difference to other tools like NetPipe [38], coNCePTuaL [35] or the Pallas Micro Benchmarks (PMB) [36] is that the the framework offers the possibility 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 is important to compare low-level performance with MPI performance and enables the user to assess the quality and overheads of a specific MPI implementation. 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) 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. [14] to achieve reproducible and accurate benchmark results.

3.1 General Definitions

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). This forces developers of network benchmarks to do all time measurement on only one machine and measure a round trip time (from now on RTT). Many benchmarks (e.g. Netpipe, PMB) use the so called ping-pong scheme. This scheme uses two hosts, the client that initiates the communication and measures needed RTTs and the server that mirrors all received packets back to the client. This common scheme is depicted in the left part of Figure 1. Other schemes as our simplified ping-ping (originally mentioned in [36]), depicted in the right part of Figure 1, can be used to get the performance of multiple consecutive message sends. However, one has to be aware that a ping-ping with many

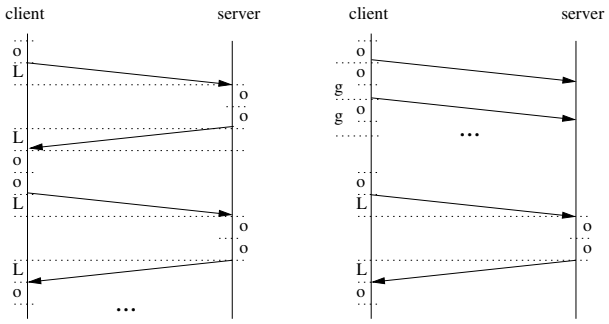

Figure 1. Left: ping-pong micro-benchmark scheme for 1 byte messages in the LogGP model; Right: ping-ping micro-benchmark scheme for 1 byte messages in the LogGP model.

packets is able to saturate the network and introduce contention easily. An additional possibility to influence the benchmark is a ping-ping scheme with an artificial delay between each message send. The delay can easily be achieved with calculation on the CPU.

We combine all those possibilities and use them to assess all LogGP parameters as less intrusively as possible. We introduce the notion of the parametrized round trip time (from now on *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)*.

A pseudocode for server and client to measure a sin-

```
void server(int n, int s) {
  for(int i=0; i<n; i++)
    /* receive s bytes from client */
    recv(client, s);
  /* send s bytes to client */
  send(client, s);
}

double client(int n, int d, int s) {
  t = -time(); /* get time */
  /* send s bytes to server */
  send(server, s);
  for(int i=0; i<n-1; i++) {
    wait(d); /* wait d microseconds */
    /* send s bytes to server */
    send(server, s);
  }
  /* receive s bytes from server */
  recv(server, s);
  t += time(); /* get time */
  return t;
}
```

Listing 1. Pseudocode to measure *PRTT(n,d,s)*

gle ***PRTT(n,d,s)*** is given in Listing 1. 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:

$$PRTT(1,0,s) \quad = \quad 2 \cdot (L + 2o + (s-1)G). \quad (1)$$

If we define the cumulative hardware gap G_{all} as

$$G_{all} \quad = \quad g + (s-1)G\,, \quad (2)$$

n ping-ping messages can be modeled as (remember that the LogGP model defines $o < G_{all}$)

$$PRTT(n,0,s) \quad = \quad 2 \cdot (L + 2o + (s-1) \cdot G) + (n-1) \cdot G_{all}\,. \quad (3)$$

With (1), we get

$$PRTT(n,0,s) \quad = \quad PRTT(1,0,s) + (n-1) \cdot G_{all}\,. \quad (4)$$

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

$$PRTT(n,d,s) \quad = \quad PRTT(1,0,s) + (n-1) \cdot max\{o+d, G_{all}\}\,. \quad (5)$$

The following section uses the PRTT to assess the LogGP parameters of different network interfaces.

3.2 Assessment of the Overhead o

If we rewrite Equation 5 to

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

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

$$\frac{PRTT(n,d,s) - PRTT(1,0,s)}{n-1} \quad = \quad o + d\,. \quad (6)$$

This enables us to compute o from the measured $PRTT(n,d,s)$ and $PRTT(1,0,s)$. We chose $PRTT(1,0,s)$ for d to ensure that $d > G_{all}$ 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 $d = PRTT(2,0,s)$ to guarantee $d > G_{all}$. We chose $PRTT(1,0,s)$ to avoid unnecessary long benchmark times.

The measurement of o for $n = 3$ is illustrated in Figure 2. The whole figure represents a LogGP model for $PRTT(3,d,s)$, it is easy to see that the last part is a simple $PRTT(1,0,s)$. If we subtract $PRTT(1,0,s)$ from $PRTT(3,d,s)$, we get $2d+2o$ which equals to $(n-1)(d+o)$ (remember that $n = 3$ in our example) as shown in Equation (6).

This measurement method enables us to get a pretty accurate value of o for each message size s. It needs only a small number of messages (we used $n = 16$ in our tests) that does not saturate or flood the network unnecessarily to measure o. Furthermore, we are able to compute o directly from a single measurement and without inter-dependencies to other LogGP parameters (that are computed themselves and contain already an error of first order). We do also not need to adjust d stepwise to fit other values.

3.3 Assessment of the Gap Parameters g,G

Using Equation (4), we get a polynomial of degree one of the form $f(s) = G \cdot s + g$:

$$G(s-1) + g \quad = \quad \frac{PRTT(n,0,s) - PRTT(1,0,s)}{n-1} \quad (7)$$

This represents a linear function. 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

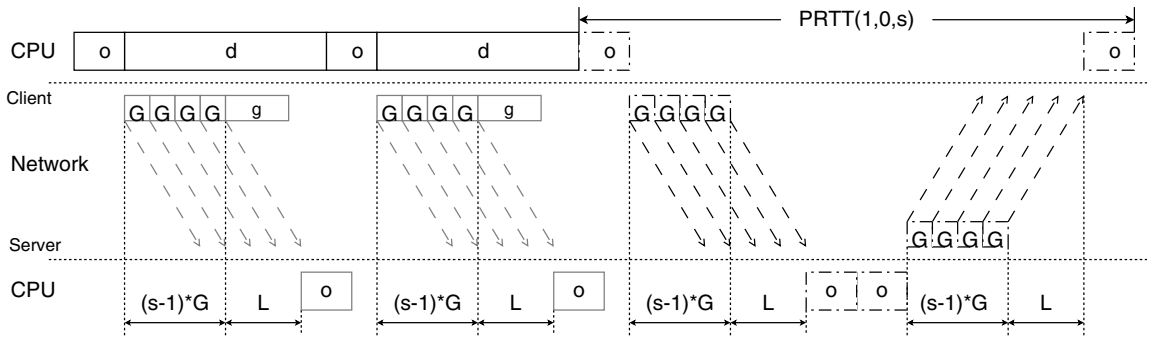

Figure 2. Measurement Method for o for $n = 3$. This figure shows the components of $PRTT(n, d, s)$.

also that this method would not allow us to detect protocol changes in the lower levels that influence the LogGP parameters.

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 [5], which can be solved directly for the needed two degrees of freedom (g and G), to perform the fit. This gives 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 3.5). This helps to avoid mispredictions and to detect anomalies at specific message sizes.

We use only a small n ($n = 16$ for our tests) 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 (as we experienced with TCP). Furthermore, we are able to use our method to detect changes in the underlying communication protocol, as described in Section 3.5.

A graphical representation of our method with Open MPI over InfiniBand™ is shown in Figure 3.

3.4 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 (some message processing is done after the sending of the message is started). This is due to the fact that the network developers want to minimize the round trip time and try to move all the bookkeeping after the message send. 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 [2] and report half of the round trip time (EEL in [2]) of a small message as latency. We use $PRTT(1, 0, 1)/2$ for this purpose.

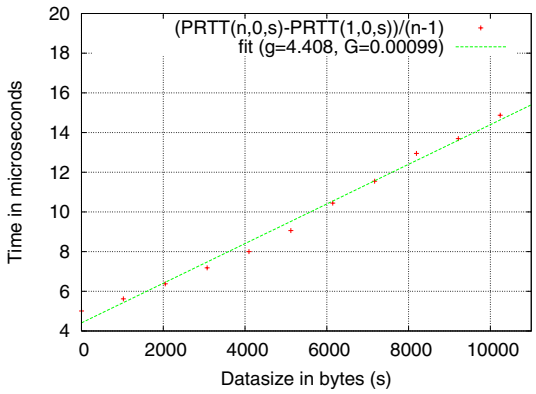

Figure 3. Parameter Benchmark and Fit for Open MPI over InfiniBand™.

3.5 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

$$lsq(k, l) = \frac{\sum_{i=k}^{l} (G \cdot size(i) + g - val(i))^2}{l - k - 2}, \quad (8)$$

where $val(i)$ is the measured value at point i and $size(i)$

is the message-size at point i. The subtraction of 2 in the denominator is 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 $lsq(lastchange, current) \cdot pfact$, 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.

4 Applying the Method

This section discusses first results of the application of our measurement methods with the tool Netgauge [34]. We analyzed different interconnect technologies and parallel systems to evaluate their performance in the LogGP model. The used test-systems are described in Table 1.

We benchmarked TCP over Gigabit Ethernet and MPICH2 1.0.3, SCI with NMPI 1.2, InfiniBand™ with Open MPI/OpenIB and Myrinet with Open MPI/GM. The graphs for $G \cdot (s-1)+g$ (cf. Equation (7)) for InfiniBand™ and Myrinet are shown in Figure 4.

Table 2 shows the numerical results for the LogGP measurements on the different systems. We uses blocking MPI_SEND and MPI_RECV to measure those values. The TCP results show that o, 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 do also see that o 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 and receive messages because $o \approx G_{all}$.

InfiniBand™ shows also a very interesting behavior. The Open MPI OpenIB component uses also polling to send or receive messages. A protocol change at approximately $12kb$ 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 accounts this to g because it has to wait

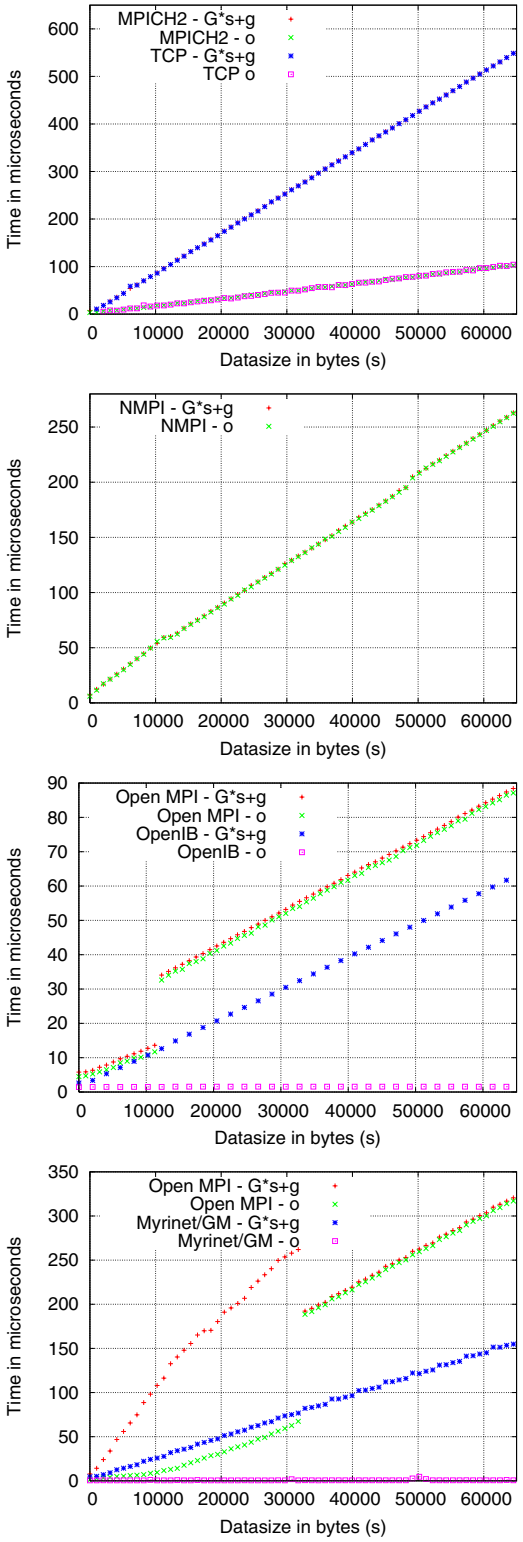

Figure 4. Measurement results for GigE/TCP, SCI, InfiniBand™ and Myrinet/GM (in this order). The graphs show $f(s) = G \cdot (s-1) + g$, such that the slope indicates G and $f(1) = g$. The parameters of the fitted functions for g and G can be found in Table 2.

Transport	CPU	OS	Additional Information
MPICH2	Opteron 246, 2GHz	Linux 2.6.9	MPICH2 1.0.2, BCM5704 GigE Network Chip
NMPI/SCI	Xeon 2.4GHz	Linux 2.6.9	NMPI-1.2, SCI-Adapter PSB66 D33x
Open MPI/OpenIB	Opteron 244	Linux 2.6.9	Open MPI 1.1.2, OFED-1.0, Mellanox MT25208
Open MPI/gm	Athlon MP 1.4GHz	Linux 2.6.9	Open MPI 1.1.2, GM 2.0.23, Myrinet 2000

Table 1. Details about the test systems.

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 G which exhibits no protocol change. The low-level overhead to post a send request is independent of the message size. Open MPI introduces an additional overhead which is due to the local copy (for eager send) or InfiniBand™ memory registration ([32], for rendezvous).

Myrinet appears to be using no polling for small messages ($o < G_{all}$) and polling for messages larger than 32kB ($o \approx G_{all}$). 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 for InfiniBand™.

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/` for public use. This implementation has been tested extensively with different modern MPI implementations and low-level networking APIs.

We plan to extend Netgauge to use more low-level communication modules and to analyze more different interconnects and MPI implementations. The evaluation of this scheme for guiding a homogeneous multi-NIC scheduler for Open MPI has already begun and seems very promising. Later versions of Netgauge will also support the parameter benchmarking for non-blocking communication.

References

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

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

[3] M. Bernaschi and G. Iannello. Quasi-Optimal Collective Communication Algorithms in the LogP Model. Technical report, University of Naples (DIS), 03 1995.

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

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

[6] G. Blelloch. Scans as Primitive Operations. In *Proc. of the International Conference on Parallel Processing*, pages 355–362, August 1987.

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

[8] D. Culler, L. T. Liu, R. P. Martin, and C. Yoshikawa. LogP Performance Assessment of Fast Network Interfaces. *IEEE Micro*, February 1996.

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

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

[11] 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, and 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, September 2004.

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

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

[14] W. Gropp and 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*, pages 11–18, London, UK, 1999. Springer-Verlag.

Transport	Protocol Interval (bytes)	L (μs)	o(1) (μs)	g (μs)	G ($\mu s/byte$)
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
Open MPI/openib	$1 \leq s < 12289$	5.96	4.72	5.14	0.00073
	$12289 \leq s$	5.96	4.72	21.39	0.00103
Open MPI/gm	$1 \leq s < 32769$	10.53	1.27	9.44	0.0092
	$32769 \leq s$	10.53	1.27	52.01	0.0042

Table 2. LogGP Parameters for different Transport Protocols

[15] S. E. Hambrusch. Models for parallel computation. In *ICPP Workshop*, pages 92–95, 1996.

[16] S. E. Hambrusch and A. A. Khokhar. An architecture-independent model for coarse grained parallel machines. In *Proceedings of the 6-th IEEE Symposium on Parallel and Distributed Processing*, 1994.

[17] T. Hoefler. Evaluation of publicly available Barrier-Algorithms and Improvement of the Barrier-Operation for large-scale Cluster-Systems with special Attention on InfiniBand™Networks. Master's thesis, TU-Chemnitz, 2004. url: http://archiv.tu-chemnitz.de/pub/2005/0073/data/diploma.pdf.

[18] T. Hoefler, L. Cerquetti, T. Mehlan, F. Mietke, and 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)*, pages 562–569, June 2005.

[19] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. LogfP - A Model for small Messages in InfiniBand. In *Proceedings, 20th International Parallel and Distributed Processing Symposium IPDPS 2006 (PMEO-PDS 06)*, April 2006.

[20] G. Iannello. Efficient algorithms for the reduce-scatter operation in loggp. *IEEE Trans. Parallel Distrib. Syst.*, 8(9):970–982, 1997.

[21] G. Iannello, M. Lauria, and S. Mercolino. Logp performance characterization of fast messages atop myrinet, 1998.

[22] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity*, pages 869–941. Elsevier, Amsterdam, 1990.

[23] R. M. Karp, A. Sahay, and E. Santos. Optimal broadcast and summation in the logp model. Technical report, Berkeley, CA, USA, 1992.

[24] T. Kielmann, H. E. Bal, and 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*, pages 1176–1183, London, UK, 2000. Springer-Verlag.

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

[26] F. T. Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., 1992.

[27] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. *Int'l Journal of Parallel Programming, 2004*, 2004.

[28] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of Parallel Computation: A Survey and Synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70, 1995.

[29] G. I. Massimo Bernaschi. Collective communication operations: experimental results vs. theory. *Concurrency - Practice and Experience 10*, 5:359–386, 1998.

[30] Message Passing Interface Forum. MPI: A Message Passing Interface Standard. 1995.

[31] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. Technical Report, University of Tennessee, Knoxville, 1997.

[32] F. Mietke, R. Baumgartl, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm. Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack. 8 2006. Accepted for publication at Euro-Par 2006 Conference.

[33] C. A. Moritz and M. I. Frank. LoGPC: Modelling Network Contention in Message-Passing Programs. *IEEE Transactions on Parallel and Distributed Systems*, 12(4):404, 2001.

[34] Netgauge. http://www.unixer.de/research/netgauge/, 2006.

[35] S. Pakin. Reproducible network benchmarks with conceptual. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, *Euro-Par 2004 Parallel Processing, 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Proceedings*, pages 64–71. Springer, 2004.

[36] Pallas GmbH. Pallas MPI Benchmarks - PMB, Part MPI-1. Technical report, Pallas GmbH, 2000.

[37] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance Analysis of MPI Collective Operations. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium, 4th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 05)*, Denver, CO, April 2005.

[38] D. Turner and X. Chen. Protocol-dependent message-passing performance on linux clusters. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, page 187, Washington, DC, USA, 2002. IEEE Computer Society.

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