# A Practical Approach for Measuring LogP Parameters

M. B. Ibáñez[§], A. Di Serio[†]

[§]Universidad Carlos III de Madrid,
Departamento de Telemática, Av. Universidad 30,
28911 Leganés (Madrid), Spain
[†]Universidad Simón Bolívar,
Departamento de Computación y Tecnología de la Información,
Sartenejas, Baruta, Apartado 89000,
Venezuela

**Abstract.** Performance modeling is used for the design and implementation of efficient parallel algorithms and runtime systems. This paper describes a methodology to measure the LogP communication parameters that characterize the performance of the communication hardware on a parallel machine. We use MPI as message passing platform and work with messages of any size in a homogeneous architecture. The main focus of our work is to minimize the number of trials of micro-benchmarks used to estimate the LogP parameters.

## 1 Introduction

Performance modeling is used for the design and implementation of efficient parallel algorithms and runtime systems. Considerable efforts have been done to develop models of parallel computation. PRAM (Parallel Random Access Machine) [6], a simple shared memory model, has proven to be useful in identifying inherent parallelism in problems, but it is unrealistic because it assumes that all processors work synchronously and that interprocessor communication is free. Other models such as Phase PRAM [7] and APRAM [3] incorporate some notion of asynchronous execution, but still they lack mechanisms to consider the latency of communication as well as memory and network contention. The bulk-synchronous parallel model (BSP) developed by Valiant [8] accounts for communication and abstracts the architectural features in a small number of parameters.

Another parallel model having received considerable attention is the LogP model [4], this model lacks explicit synchronization. LogP captures the essential features of the communication system by measuring only four parameters. Culler et al. [5] present a set of communication micro-benchmarks to extract the LogP performance parameters for small messages. Several works have extended the original LogP model, among the most relevant we can cite LogGP [1] that incorporates long messages into the LogP model and an extension for a parallel computation model for heterogeneous clusters [2].

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 785-792, 2008.

Although the LogP model has a broad applicability, it has two main drawbacks, measuring of the gap parameter is time consuming and highly intrusive. In this paper, we present a method to measure LogP parameters minimizing the number of trials of micro-benchmarks that measure the communication parameters. Our work is based on the Culler et al. approach presented in [5] over low overhead layers.

The remainder of this paper is organized as follows. In Section 2 we show a description of the communication parameters and the micro-benchmarks we use. Section 3 presents the results of our calibration and validation of the LogP communication parameters. Section 4 describes our measurement methodology. Finally, in Section 5 we present the conclusions of the paper.

## 2 Micro-benchmarks for measuring LogP parameters

LogP model [4] characterizes the behavior of communications using four parameters: the *latency* $L$ representing the maximum delay associated with delivering a message of size zero from its source to its target processor; the *overhead* $o = o_s + o_r$ representing the amount of time for which a processor is busy during the transmission $o_s$ or reception $o_r$ of a message; the *gap* $g$ representing a lower bound on the time between the transmission of successive messages and the number $P$ of processors in the machine.

Cullet et al. [5] base their measures on the behavior of the ping-pong benchmark repetition. They state that when the master starts to send a set of messages, there is a period of time where the sender spends all its time sending messages ($o_s$ parameter). The micro-benchmark $o_s$ at Figure 1 computes the time the master spends sending repeatedly a message, its results allow to measure $o_s$.

```
Master
MPI_Barrier();
t_start = MPI_Wtime();
for (i=0; i<N; i++)
        MPI_Send(n bytes to slave);
t_stop = MPI_Wtime();
t_os = (t_stop - t_start) / N;
for (i=0; i<N; i++)
        MPI_Recv(0 bytes from slave);
```

```
Slave
MPI_Barrier();
for (i=0; i<N; i++){
        MPI_Recv(n bytes from master);
        MPI_Send(0 bytes to master);
}
```

**Fig. 1.** Microbenchmark for measuring $o_s$

In the general case, there is no direct way to measure the time the slave spends receiving a message ($o_r$), nevertheless MPI has the routine (MPI_Probe) that

provides a blocking test for a message. Once MPI_Probe has been completed, the message has arrived and thus it is possible to compute $o_r$. The micro-benchmark $o_r$ at Figure 2 shows how we measure $o_r$ using MPI_Probe.

Since we have $o_s$ and $o_r$, in order to obtain the latency $L$, we just need to measure the round-trip time (RTT) associated with a single request-reply operation and solve $RTT = 2(o_s + L + o_r)$.

```
Master                                  Slave
MPI_Barrier();                          MPI_Barrier();
t_or = 0;
for (i=0; i<N; i++){                    for (i=0; i<N; i++){
        MPI_Send(0 bytes to slave);             MPI_Recv(0 bytes from master);
        MPI_Probe();                            MPI_Send(n bytes to master);
        t_start = MPI_Wtime();          }
        MPI_Recv(n bytes from slave);
        t_stop = MPI_Wtime();
        t_or = (t_stop - t_start) + t_or;
}
t_or = t_or / N;
```

**Fig. 2.** Micro-benchmark for measuring $o_r$

The *gap* $g$ is defined as the minimum time interval between two consecutive message transmissions or consecutive message receptions at a processor. It is computed as $g = o_s + o_r + Idle$. Culler approach consists on saturating the network with messages that are sent after a controlled amount of computation $\Delta$ between messages. When Idle becomes greater than $\Delta$ we have found $g$. Figure 3 presents the micro-benchmark $g$ that shows the use of $\Delta$ that allows to measure $g$. In Section 3 we show how to choose $\Delta$ and how to decrease the number of iterations.

## 3 Measurement of LogP Parameters

In this section we present the approach followed to compute the LogP parameters. Our experimental platform is a Linux cluster of 24 dual nodes Pentium III, runs Linux with 2.6.12 kernel and use Fast Ethernet network. We use MPI as communication library (MPICH implementation).

### 3.1 Measuring $o_s$ and $o_r$ for any message size

In Figure 4 we observe that $o_s$ and $o_r$ grow linearly with the size of the message, although the receive overhead is increased to a rate smaller than the send overhead.

```
Master
MPI_Barrier();
t_start = MPI_Wtime();
for (i=0; i<N; i++) {
        delay(Δ);
        MPI_Send(n bytes to slave);
}
t_stop = MPI_Wtime();
t_g = (t_stop - t_start) / N;
for (i=0; i<N; i++)
        MPI_Recv(0 bytes from slave);
```

```
Slave
MPI_Barrier();
for (i=0; i<N; i++){
        MPI_Recv(n bytes from master);
        MPI_Send(0 bytes to master);
}
```

**Fig. 3.** Micro-benchmark for measuring g

In Figure 5 we show the first 20 measures of the send and receive overhead respectively for messages varying from 0 to 5KB. The first values in Fig. 5(a) show simply the cost of sending messages without receiving any replies then, a fraction of the replies start to arrive and the message cost increase, thus $o_s$ is taken from the first set of measures. In Figure 5(a), we observe that as we work with messages of greater size, we need a smaller number of measures and we verified that the number of measures is less than $round\_trip\_time/o_s$ [5]. In Figure 5(b), we observe that $o_r$ is almost constant for a given message size, and 4 or 5 measures are enough to obtain the value of $o_r$.

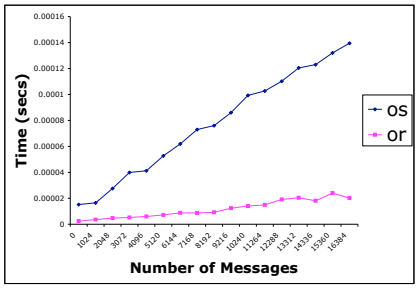
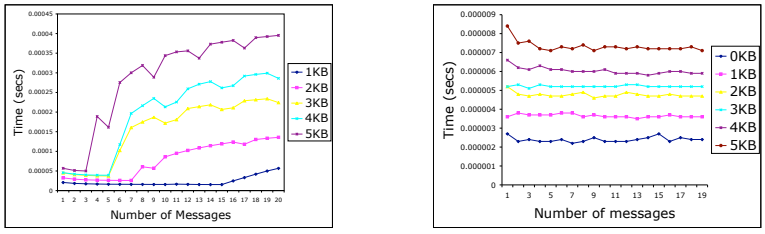



**Fig. 4.** $o_s$ and $o_r$ values for messages from 0 to 16KB




(a) $o_s$ (b) $o_r$

**Fig. 5.** Partial micro-benchmark signatures

### 3.2 Measuring L

In order to compute $L$, we need the value of the parameters $o_s$ and $o_r$ for messages of 0 bytes. We obtain $o_s = 15.2\mu secs$ and $o_r = 2.4\mu secs$ by using the micro-benchmarks presented at Figures 1 and 2. The round-trip time gives us the quantity $2(o_s + L + o_r)$ thus, $L = 74.4\mu secs$.

### 3.3 Measuring g

For each message size, the estimation of $g$ involves using the micro-benchmark at Figure 3 with different delay times ($\Delta$'s). The difficulty here is, not only the amount of measures needed to be taken, but also how to choose the delays.

We work with a sequence of delta values $\{\Delta_0, \Delta_1, ...\}$. We define:

- $\Delta_0$ as the asymptotic upper bound of the $o_s$ curve generated by the micro-benchmark presented at Fig. 1.
- $\Delta_{i+1} = \Delta_i + \Delta_0/10$.

The curve generated by micro-benchmark $g$ using $\Delta_i$ is called $o_{s_i}$ curve.

The imposed delay $\Delta_0$ is always greater than the maximum value of the $o_{s_0}$ curve and the $o_{s_{i+1}}$ curve is an upper bound curve of the $o_{s_i}$ curve. Thus, it is not necessary to generate the $o_{s_i}$ curve to find the $\Delta_i < Idle$ in $g = o_s + o_r + Idle$, the round trip time considering the time delay $\Delta_i$ of one message is enough to decide whether $\Delta_i$ is the right delay.

Figure 6 shows the $g$ values obtained following this approach. We observe that $g$ has a linear behavior and using this characteristic of $g$, we fitted a linear model using only two values of the data obtained from the benchmark. The Table 1 shows the percentage error of $g$ computed experimentally with the micro-benchmark $g$ showed at Figure 3 and the one obtained using the linear model of $g$. We notice that for small messages, it is necessary to obtain $g$ experimentally but for large messages, it is enough to compute the $g$ line.

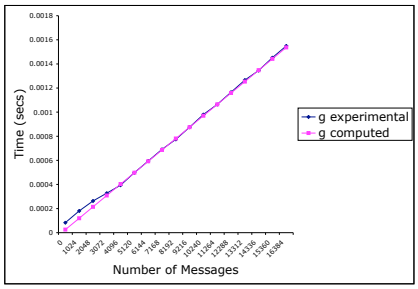



**Fig. 6.** g values for messages from 0 to 16KB

| nbytes | % error g experimental | % error g computed |
|---|---|---|
| 0 | 0.0 | 69.6 |
| 1024 | 1.0 | 33.7 |
| 2048 | 7.5 | 18.6 |
| 3072 | 8.8 | 5.8 |
| 4096 | 12.2 | 2.2 |
| 5120 | 6.3 | 0.04 |
| 6144 | 3.3 | 0.4 |
| 7168 | 3.0 | 0.7 |
| 8192 | 2.0 | 0.8 |
| 9216 | 0.3 | 0.0 |
| 10240 | 2.5 | 1.1 |
| 11264 | 1.8 | 0.0 |
| 12288 | 3.2 | 0.5 |
| 13312 | 4.6 | 1.0 |
| 14336 | 3.1 | 0.1 |
| 15360 | 4.7 | 0.7 |
| 16384 | 6.7 | 0.8 |

**Table 1.** Percentage of error in g experimental and computed

Finally, in Figure 7 we show the estimated and real values of communication for different size of messages. We observe that the estimated values obtained by using LogP parameters measured by our approach are very close to the real ones, the maximum error found for short messages was 12.2%, for long messages was 6,7% and the average was 4.2%.

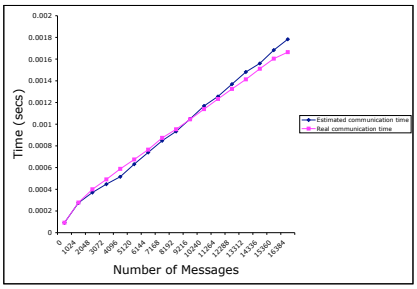



**Fig. 7.** Communication times real and estimated communication times for messages from 0 to 16KB

## 4 Methodology

Based on the results obtained in the previous section, we propose the following methodology for obtain the LogP parameters for messages using a small number of experiments.

$o_s$ Use the benchmark $o_s$ showed at Figure 1 with $N = RTT/o_s$. Compute $\Delta_0$ as the asymptotic upper bound of the $o_s$ curve for every message size.

$o_r$ Use the benchmark showed at Figure 2, repeat the micro-benchmark until you obtain an accuracy of $\pm 5\%$.

**L** Compute the time for a round trip time, $o_s$ and $o_r$ for messages of size 0 and use the formula $L = \frac{RTT}{2} - (o_s + o_r)$.

**g** Use the benchmark showed at Figure 3 with N = 1 and $\Delta_0$ for messages smaller than 4KB. Use the benchmark in Figure 3 with N = 1 for two different large messages and compute the $g$ line.

## 5 Conclusions

In this paper we have presented a methodology to measure the parameters of the LogP model for any size of message that minimizes the number of trials of micro-benchmarks. The parameters $o_s$ and $L$ are computed following the Culler approach stated at [5], the $o_r$ parameter is measured directly using the MPI_Probe routine.

The *gap* is traditionally very time consuming to measure due to the difficulty to find a suitable $\Delta$. We have found that the initial $\Delta$ is the asymptotic upper bound of the $o_{s_0}$ curve generated by the micro-benchmark presented at Fig. 1. With our approach, it is not necessary to repeat the transmission of the message, one measure is enough to determine whether a $\Delta_i$ is the Idle time in $g = o_s + o_r + Idle$. Since $g$ has a linear behavior, for long messages it is not necessary

to use the micro-benchmark $g$ in order to compute $g$. The value of $g$ can be obtained from the $g$ line. We have found that for short messages, it is convenient to follow the traditional approach.

With this approach we minimize the intrusive behavior of the benchmarks, and the LogP parameters can be estimated with few iterations. We have validated our methodology comparing the real communication time with the estimated using the LogP parameters and found that our approach computes accurately the LogP parameters.

## References

1. Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating Long Messages into the LogP Model — One step closer towards a realistic model for parallel computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.
2. Jose Luis Bosque and Luis Pastor. A Parallel Computational Model for Heterogeneous Clusters. *IEEE Trans. Parallel Distrib. Syst.*, 17(12):1390–1400, 2006.
3. Richard Cole and Ofer Zajicek. The APRAM: Incorporating asynchrony into the PRAM model. In *Proceedings of the Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 169–178, New York, NY, USA, 1989. ACM Press.
4. David E. Culler, Richard M. Karp, David A. Patterson, Adhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thosten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *SIGPLAN Not.*, 28(7):1–12, 1993.
5. David E. Culler, Lok Tin Liu, Richard P. Martin, and Chad O. Yoshikawa. Assessing Fast Network Interfaces. *IEEE Micro*, 16(1):35–43, 1996.
6. Steven Fortune and James Wyllie. Parallel in Random Access Machines. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118, New York, NY, USA, 1978. ACM Press.
7. P.B. Gibbons. A more practical PRAM model. In *SPAA'89: Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pages 158–168, New York, NY, USA, 1989. ACM Press.
8. Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, 1990.