LogGPS: A Parallel Computational Model for Synchronization Analysis

Fumihiko Ino, Noriyuki Fujimoto and Kenichi Hagihara
Graduate School of Engineering Science
Osaka University
Toyonaka, Osaka 560-8531, Japan
{ino, fujimoto, hagihara}@ics.es.osaka-u.ac.jp

ABSTRACT

We present a new parallel computational model, named LogGPS, which captures synchronization.

The LogGPS model is an extension of the LogGP model, which abstracts communication on parallel platforms. Although the LogGP model captures long messages with one bandwidth parameter (G), it does not capture synchronization that is needed before sending a long message by high-level communication libraries. Our model has one additional parameter, S, defined as the threshold for message length, above which synchronous messages are sent.

We also present some experimental results using both models. The results include (1) a verification of the LogGPS model, (2) an example of synchronization analysis using an MPI program and (3) a comparison of the models. The results indicate that the LogGPS model is more accurate than the LogGP model, and analyzing synchronization costs is important when improving parallel program performance.

1. INTRODUCTION

When developing parallel programs, the programs are expected to get much better performance on many parallel platforms. To develop such programs, parallel computational models are useful for developers. The models abstract the performance of the platforms so that, by changing the parameters of the model, developers can know the predicted performance of programs on every platform.

Both the LogP model [5] and the LogGP model [2] are such realistic models. In [2, 5, 6] the models have shown good accuracy for low-level communication libraries such as Active Messages [19] and Elan library [8]. The LogP model abstracts the communication performance of a platform by four parameters: L, o, g and P (see §3). The LogGP model has one additional parameter (G) compared with the LogP model and captures special support (that many platforms have) for long messages, which provides much higher communication bandwidth for long messages compared to short messages. That is, the LogGP model uses two bandwidth: $1/G$ for long messages and $1/g$ for short messages.

Thus, the LogGP model reflects the advantage of the special support, however, it does not reflect the disadvantage, i.e., the need of synchronization when sending a long message. This lack might make the model inaccurate, especially for programs written using high-level communication libraries such as MPI [13] and MPL [9]. For example, many MPI implementations [4, 7, 10, 15] switch communication protocols according on message length, and the protocols can be classified into two groups: synchronous and asynchronous. This protocol switch is out of consideration under the LogGP model, which reflects only the advantage of the support. Therefore, for accuracy, models for parallel programs such as MPI programs should reflect not only the advantage but also the disadvantage of the support.

In previous works [1, 12, 14, 17], the LogGP model has been used to analyze the performance of MPI programs. In [1, 14] they indicated that synchronization occurs before sending a long message. They assumed that the sender processor synchronizes to the receiver processor at the shortest time and included the synchronization cost into the overhead, o, defined as a constant value for every message length. However, since the sender does not always synchronize at the shortest time, their method has insufficient accuracy for programs that vary its synchronization costs at long range. The same discussion can be held for [12], which misses considering synchronization.

In [17] they built a synchronization model for a wavefront application that uses MPI routines. They showed its synchronization cost as a formula, which consists of the communication latency and the processor ID, and added the synchronization cost to the communication cost derived by the LogGP model. Their method has good accuracy but cannot apply to programs that are difficult to derive the equation without run. In addition, the derived model is dependent on the target application, so that we have to repeatedly build a synchronization model for every application.

Through the previous works, we think that synchronization costs must be accurately analyzed for the whole analysis of MPI programs and synchronization models must be independent of the target programs. To analyze the synchronization costs in accurate, the LogGP model (and also the LogP model) is insufficient because it misses considering the protocol switch.

Furthermore, both the methods for deriving the parameters of the models and the definitions of MPI routine costs

PPOPP'01, June 18-20, 2001, Snowbird, Utah, USA.

Table 1: MPICH communication protocols

Protocol	Packet	Synchronization
Short	single	no
Eager	multiple	no
Rendezvous	multiple	yes

are different among the works, and it is not clear which method provides better accuracy.

Now, our goals are as follows.

(G1) Develop a realistic model for synchronization analysis.

(G2) Develop a method for deriving the parameters of the model and apply the model to high-level communication routines.

To achieve goal (G1), we extended the LogGP model by adding one parameter, S, or the threshold for message length, above which synchronous protocols are employed. Furthermore, to make the model more accurate, we represented the communication overhead as a linear function of message length. Then we named the new model LogGPS. To achieve goal (G2), we applied the LogGPS model to some MPI routines [13] and compared with the LogGP model by analyzing an MPI program.

Finally, we believe that the LogGPS model is useful to analyze the synchronization costs of parallel programs, especially when improving the performance of the programs.

The rest of the paper is organized as follows. §2 describes the communication protocols used in MPI implementations [4, 7, 10, 15]. §3 presents the LogGPS model and the definitions of MPI routine costs. §4 presents the experimental results. Finally, §5 concludes the paper.

2. MPI COMMUNICATION PROTOCOLS

In this section, we describe the communication protocols used in MPICH [7], the base of many MPI implementations. Some other implementations are also shown in Table 2.

MPICH consists of two layers, machine-dependent and machine-independent, separated by Abstract Device Interface (ADI). The default ADI (P4) uses three protocols: *Short*, *Eager* and *Rendezvous* (*Rndv.*), as shown in Table 1. In the following, let *Ps* and *Pr* be the sender processor and the receiver processor, respectively.

In the Short and Eager protocol, *Ps* should not wait for *Pr* to call the matching receive routine because *Pr* copies messages to own buffer (Fig. 1(a)). In contrast, in the Rndv. protocol, *Ps* has to send a request (REQ) prior to the original message and wait for an acknowledgement (ACK) from *Pr* (Fig. 1(b)). Therefore, the use of blocking send routine (MPI_Send) makes *Ps* idle inside the send routine during the wait. On the other hand, nonblocking send routine (MPI_Isend) allows *Ps* to return immediately after sending the REQ. However, if the ACK has not been arrived when *Ps* calls the matching wait routine (MPI_Wait), *Ps* becomes idle inside the wait routine (Fig. 1(c)). Thus, the Rndv. protocol needs synchronization, but messages are transmitted in bulk after the synchronization, so that practical communication bandwidth grows up. Furthermore, some platforms have special support for long messages to raise the bandwidth more. For example, the NEC Cenju-4 [11] and Myrinet [3] provide remote DMA transfer.

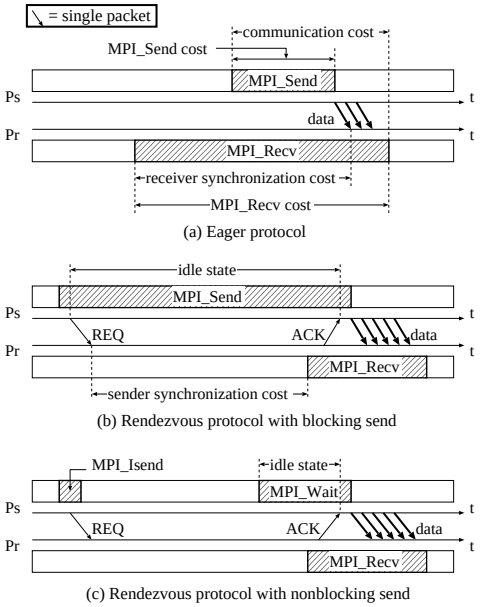


Figure 1: Behavior of MPICH communication protocols

Table 2: Relationship between message length and communication protocols of MPI implementations

Implementation	Message length (bytes)		
	Short	Eager	Rndv.
MPICH P4	∼1023	∼127999	128000∼
MPICH Cenju-4	∼1028	–	1029∼
MPICH-SCore	∼8191*	∼16383**	16384** ∼
LAM C2C	∼65536	–	65537∼
IBM PE	∼4095	–	4096** ∼

∗: on Myrinet ∗∗: changeable by runtime option

Table 2 shows the relationship between message length and the communication protocols of MPI implementations [4, 7, 10, 15]. Each employs asynchronous protocols (Short and Eager) for short messages and synchronous (Rndv.) for long. That is, MPI implementations switch the protocol to provide better bandwidth for any given message length.

To change the threshold length between the asynchronous and synchronous protocols, we must modify the source code of MPI implementation and re-compile it. But some implementations such as MPICH-SCore [15] and IBM PE [10] can change the threshold statically by runtime option. As far as we know, the threshold is unchangeable during a run.

In the following we use the term *sender synchronization cost*, defined as the length of the wait time at *Ps*, time after the arrival of a REQ until the call of the receive routine (Fig. 1(b)). We also use the term *receiver synchronization cost*, the length of the wait time at *Pr*, time after the call of a receive routine until the arrival of the message (Fig. 1(a)). *Synchronization costs* are defined as the sum of sender synchronization costs and receiver synchronization costs. Besides, we use the term *communication cost*, defined as the length of time after the call of a send routine at *Ps* until the call of the receive routine at *Pr*, and the term *MPI routine cost*, defined as the length of time after the call of the routine until the return of the routine (Fig. 1(a)).

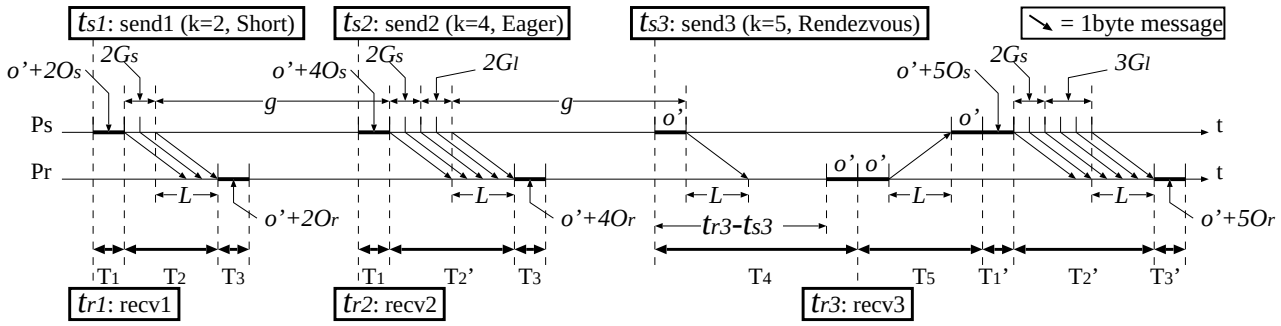


Figure 2: An example of message sends under the LogGPS model ($s = 2, S = 4$)

Table 3: Communication costs under the LogGPS model

Condition	Communication cost		T_1,T_2,...,T_5	
$k \leq s$	$T_1 + T_2 + T_3$	(1)	$T_1 = o' + kO_s$	$T_2 = kG_s + L$
$s < k \leq S$	$T_1 + T_2' + T_3$	(2)	$T_2' = sG_s + (k-s)G_l + L$	$T_3 = o' + kO_r$
$k > S$	$T_4 + T_5 + T_1 + T_2' + T_3$	(3)	$T_4 = max\{o' + L, t_r - t_s\} + o'$	$T_5 = o' + L + o'$

3. THE LOGGPS MODEL

The LogGP model [2], the base of the LogGPS model, has five parameters as follows.

- L: an upper bound on the *Latency*, incurred in sending a message from its source processor to its target processor.
- o: the *overhead*, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time the processor cannot perform other operations.
- g: the *gap* between messages, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor. The reciprocal of g corresponds to the available per processor communication bandwidth for short messages.
- G: the *Gap per byte* for long messages, defined as the time per byte for a long message. The reciprocal of G characterizes the available per processor communication bandwidth for long messages.
- P: the number of processor/memory modules.

Compared with the LogGP model, the LogGPS model has three differences as follows: differences (D1), (D2) and (D3). Let k be the length of a message in the following ($k \geq 0$).

(D1) Add one parameter, S, defined as the threshold for message length, above which synchronous messages are sent. When $k > S$, Ps waits for an ACK from Pr. Otherwise, Ps sends an asynchronous message.

(D2) Divide the overhead, o, between Ps and Pr, and let $o' + kO_s$ and $o' + kO_r$ be the send overhead and the receive overhead, respectively; o' represents the overhead for the first byte of a message, and O_s and O_r represent the send overhead and the receive overhead per byte for the subsequent byte of the message, respectively.

(D3) Add one parameter, s, defined as the threshold for message length, above which messages are sent in multiple packets. Furthermore, divide the Gap, G, between single and multiple packets, i.e., Gap G_l for $k \leq s$ and G_s for $k > s$.

Difference (D1) allows the LogGPS model to decide whether a message should be sent in asynchronous or synchronous so that the model captures synchronization. The rest of differences, (D2) and (D3), are ideas to make the model more accurate. The benefits of these ideas are shown in §4.3.

3.1 Communication Costs under the Model

In this section, we define the communication cost of a k-bytes message under the LogGPS model (Table 3).

Figure 2 shows an example of three messages (send1, send2 and send3) under the LogGPS model, where $s = 2$ and $S = 4$.

When $k \leq S$, Ps sends asynchronous messages (send1 and send2). The communication costs are defined by summing up time T_1, T_2 (or T_2') and T_3, each shown in Figure 2. First, the time to push the first byte of a message into the network, T_1, is defined as the send overhead, or $o' + kO_s$. Second, the time after the departure of the first byte at Ps until the arrival of the last byte at Pr, T_2 (or T_2'), is defined as $kG_s + L$ (or $sG_s + (k-s)G_l + L$). This is explained as follows. (a) Subsequent bytes take G_s cycles each to go out. (b) After the bytes amount to length s, the subsequent bytes take G_l cycles each. (c) Each byte transmitted through the network for L cycles. Third, the time to get the last byte from the network, T_3, is defined as the receive overhead, or $o' + kO_r$. Finally, when $k \leq S$, the communication costs are defined as equations (1) and (2) shown in Table 3.

When $k > S$, Ps synchronizes to Pr (send3). In addition to times T_1, T_2 and T_3, the communication cost is defined by summing up the time to establish synchronization, times T_4 and T_5 as shown in Figure 2. First, Ps sends a REQ to Pr. We consider the REQ as a zero-byte message, so that the REQ arrives at time $o' + L$. Second, Ps waits for an ACK from Pr. On the other hand, Pr confirms if the REQ have been arrived when it calls the receive routine. This

Table 4: MPI routine costs under the LogGPS model

Routine	Condition	Cost	
`MPI_Send`	$k \leq S$	T_1	(4)
	$k > S$	$T_4 + T_5 + T_1$	(5)
`MPI_Isend`		o'	(6)
`MPI_Recv`	$k \leq s$	$max\{T_1 + T_2 - (t_r - t_s), 0\} + T_3$	(7)
	$s < k \leq S$	$max\{T_1 + T_2' - (t_r - t_s), 0\} + T_3$	(8)
	$k > S$	$max\{o' + L - (t_r - t_s), 0\} + o' + T_5 + T_1 + T_2' + T_3$	(9)
`MPI_Irecv`		o'	(10)
`MPI_Wait`		$max\{T_{blk} - (t_i - t_w), o'\}$	(11)

confirmation takes the receive overhead, or o'. Thus, the time after the call of the send routine until the confirmation of the REQ, T_4, is defined as $max\{o' + L, t_r - t_s\} + o'$, where t_s is the call time of the send routine and t_r is the call time of the matching receive routine. Third, Pr sends an ACK to Ps. As we did for REQ, we consider the ACK as a zero-byte message, so that the time to communicate the ACK, T_5, is defined as $o' + L + o'$. Finally, after receiving the ACK from Pr, Ps sends the original message. After this synchronization, the rest of the time required for communication is equal to equation (2). Therefore, when $k > S$, we define the communication cost as equation (3). Since equation (3) includes the biggest term from terms $t_r - t_s$ and $o' + L$, the LogGPS model captures synchronization costs.

3.2 MPI routine costs under the Model

In §3.1, we defined communication costs, but MPI routine costs are not equal to the communication costs, as shown in Figure 1(a). Furthermore, the performance of MPI programs varies according to the mode of employed MPI routines. That is, the performance depends on whether the programs employ blocking or nonblocking routines (although the communication costs are the same under both modes). Therefore, to get better accuracy, we should define the MPI routine costs.

MPI libraries are usually implemented on low-level communication libraries. For example, MPICH for the NEC Cenju-4 uses Paralib/CJ4 [11] and MPICH-SCore does PM [15]. Accordingly, we can apply the LogGPS model according to the following two policies.

(P1) Apply the model to each low-level communication routine, which constructs MPI routines.

(P2) Apply the model to each MPI routine so that disregard the inside implementation of the MPI routine.

According to policy (P1), we can get an accurate MPI model but the model is dependent on each MPI implementation. In contrast, according to policy (P2), we can get a model that is applicable to any implementations. However, because the model disregards the inside behavior of MPI routines, an idea to solve paradoxes is needed. The paradox is, for example, given a model that defines the MPI_Send cost as a constant value (o) and an MPI implementation that returns from an MPI_Send call after the entire message has been pushed into the network, the model cannot define the MPI_Send cost in accurate. This paradox is caused by the conflict between the constant value (o) and the linear function of message length (k), or the cost to push the entire message into the network. Therefore, in the LogGPS model, all of times T_1, T_2 and T_3 are represented as linear functions of message length. This representation allows messages to be pushed in any of times T_1, T_2 and T_3 and the model to disregard the inside implementation of MPI routines. Although the meanings of each parameter could change, we choose policy (P2) to achieve goal (G1), that is, the goal to develop a realistic model for many platforms.

Now, in the following, we define three MPI routine costs: MPI_Send, MPI_Recv and MPI_Wait. Table 4 shows some major MPI routine costs, including the above three.

First, we define the MPI_Send cost. When $k \leq S$, that is, when sending an asynchronous message, Ps spends the send overhead, T_1, and returns from the MPI_Send (equation (4)). When $k > S$, Ps synchronizes to Pr. Therefore, to return from the MPI_Send call, the time to establish synchronization, $T_4 + T_5$, is added to T_1 (equation (5)).

Second, we define the MPI_Isend cost. When $k \leq S$, Ps returns from the MPI_Isend after the start of send operation (equation (6)). When $k > S$, Ps has to synchronize but can return after the send of an ACK (equation (6)).

Besides, if send routines or receive routines are called consecutively, each consecutive call must be called at g intervals at the shortest. Therefore, all equations shown in Table 4 should include term g in a strict way. However, we disregard g and use the equations that exclude g. The reason of this exclusion will be presented in §4.2.

Third, we define the MPI_Wait cost, T_{wait}. Usually, a blocking routine consists of a nonblocking routine and a wait routine. Therefore, we compute back T_{wait} from the cost of the blocking routine, T_{blk}, as follows. First, we temporarily replace the nonblocking routine, which matches to the target MPI_Wait, to a blocking routine. Then, we can calculate T_{blk} using equations (4), (5) and (7)-(9). Second, if let t_w be the call time of the MPI_Wait and t_i be the call time of the matching nonblocking routine, then at time t_w, a processor have already processed the partial operation corresponds to time $t_i - t_w$ while the whole operation corresponds to time T_{blk}. Therefore, T_{wait} is defined as equation (11). Equation (11) includes term o' to guarantee itself to take a positive value. Strictly, the most suitable term is the time to confirm the completion of the communication, but for the simplicity of the model, we substitute the overhead, o'.

4. EXPERIMENTAL RESULTS

In this section, we present some experimental results using both the LogGPS model and the LogGP model. The experiments are structured as follows.

1. The verification of the LogGPS model: We verified that both differences (D2) and (D3) (see §3) make the model more accurate on several platforms (§4.3).

Table 5: LogGPS parameters for each platform

Platform	L (ns)	o' (ns)	O_s (ns)	O_r (ns)	G_s (ns)	G_l (ns)	s (bytes)	S (bytes)
Cenju-4	0.64×10^3	7.82×10^3	7.63	12.77	0.57	-12.74	1028	1028
Myrinet	1.16×10^3	6.55×10^3	6.86	2.57	15.48	- 0.74	8191	16383
Fast Ethernet	35.22×10^3	20.59×10^3	10.67	5.87	191.89	74.95	1023	16383

Table 6: LogGPS Round Trip Time

Condition	Round Trip Time: RTT_1	
$k \leq s$	$max\{T_1 + 2T_2 + T_3 - w, 0\} + w + T_1 + T_3$	(12)
$s < k \leq S$	$max\{T_1 + 2T_2' + T_3 - w, 0\} + w + T_1 + T_3$	(13)
$k > S$	$max\{T_2' + T_3 + o' + L - w, 0\} + w + 2T_1 + T_2' + T_3 + 2T_4 + 2T_5 - (o' + L)$	(14)

2. The usefulness of the LogGPS model: We used the model to analyze a Gaussian elimination program, and detected the sender synchronization costs as its performance bottleneck (§4.4). We also tried to eliminate the bottleneck using the model (§4.5).

3. The comparison of the models: We compared the LogGPS model with the LogGP model and confirmed that the LogGPS model yields better accuracy (§4.6).

4.1 Experimental environments

The parallel platforms we used are an NEC Cenju-4 [11] and a cluster. The cluster consists of sixteen 450MHz Pentium II processors and each is connected by Myrinet [3] and Fast Ethernet. The MPI implementations are MPICH [7] for the Cenju-4 and MPICH-SCore [15] for the cluster.

The MPI programs we used are a Round Trip Time (RTT) measurement program and a Gaussian elimination program [18] (with partial pivoting), the first prize winner in NEC Cenju-3 sections of Parallel Software Contest '95 (PSC95) [16]. The latter program solves the set of equations, $Ax = b$, where A is a dense matrix with matrix size n.

4.2 Deriving the LogGPS Parameters

The method for deriving the LogGPS parameters from each parallel platform is as follows. First, both parameters S and s are dependent on each MPI implementation, so that they are derived from its document or its source code (see Table 2). Second, the rest of the parameters except g are derived by comparing the measured RTT using the RTT measurement program and the modeled RTT using the LogGPS model (see §4.3 for detail). Finally, the gap (g) has little effect on the communication costs of high-level communication routines, therefore, as did in [14], we disregard g. Through the experiments, we used equations (4)-(11) that exclude term g. Table 5 shows the values of the LogGPS parameters for each platform.

Note that $G_l < 0$ is true. As we presented in §3.2, we apply the LogGPS model to each MPI routine, and disregard its inside implementation. Thus, the meanings of each parameter could change, and this causes $G_l < 0$. In §4.3 we will show that all MPI routine costs, defined as equations (4)-(11), take always positive values, even if $G_l < 0$.

4.3 Verifying the LogGPS Model

In this section, we verify the LogGPS model. To do this, we developed a simple RTT measurement program, which makes a k-bytes message go and come back between two processors, as illustrated in Figure 3. To imbalance processor loads, the program has one dummy iteration processed only one side of the processors. Let w be the length of the time to process the iteration.

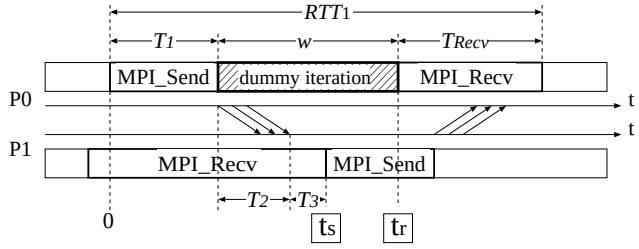
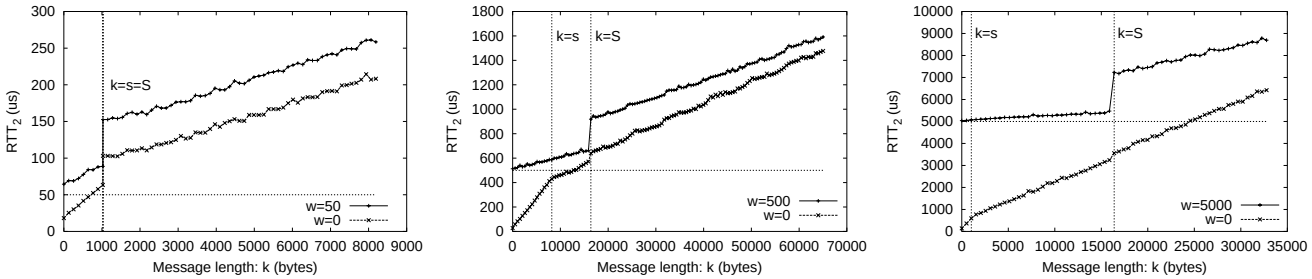


Figure 3: Behavior of the RTT measurement program ($k \leq s$)

In the following, we build a RTT model using the LogGPS model (Table 6). For want of space, only when $k \leq s$ is described. Let RTT_1 be the LogGPS RTT and let RTT_2 be the measured RTT. From Figure 3, RTT_1 can be represented as $T_1 + w + T_{Recv}$, where T_{Recv} is the MPI_Recv cost at processor *P0*. Then, as the value of w grows up, *P0* becomes busier than *P1*, so that we assume that *P0* does not wait for *P1* to call MPI routines. Therefore, if *P0* calls MPI_Send at time 0, then *P1* calls MPI_Send at time t_s, or $T_1 + T_2 + T_3$. On the other hand, *P0* calls MPI_Recv at time t_r, or $T_1 + w$. So, using equation (7), $T_{Recv} = max\{T_1 + 2T_2 + T_3 - w, 0\} + T_3$. Finally, when $k \leq s$, RTT_1 can be represented as equation (12) shown in Table 6. From equation (12), when $w \geq T_1 + 2T_2 + T_3$, RTT_1 is independent of term T_2. That is, communication costs T_2 can overlap with calculation costs w. Besides, when $s < k \leq S$ and when $k > S$, RTT_1 can also be derived in a similar way, and Table 6 shows the results.

Figure 4 shows the relationship between RTT_1 and message length, k, and Figure 5 shows the relationship between RTT_2 and k (when w takes a large value).

Now, we verify the results shown in Figure 4. First, when $w = W$ and $k \leq S$, the gradient of the graph is $O_s + O_r$. If we assume that the communication overhead is represented as a constant, o', that is $O_s = O_r = 0$, this gradient takes zero. Thus, the gradient of the graph, where $w = W$ and $k \leq S$, implies that the overhead should be represented as a linear function of message length. In Figure 5, the gradient takes non-zero on all platforms. Therefore, it is much accurate to represent the overhead as a linear function, which we described in (D2).



(a) Cenju-4 (s=S=1028) (b) Myrinet (s=8191,S=16383) (c) Fast Ethernet (s=1023,S=16383)

Figure 5: Measured Round Trip Time for different message lengths

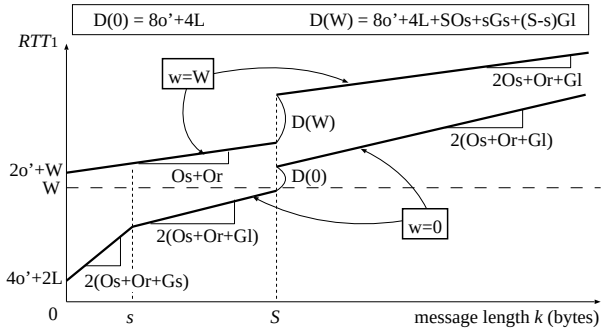


Figure 4: LogGPS Round Trip Time for different message lengths

Second, when $w = W$, the graph is continuous at $k = s$ (Fig. 4). This implies that time T_2' should be represented as $sG_s+(k-s)G_l+L$. If we assume that time T_2' is represented as kG_l+L, then the overhead, o', must take two values, one for $k \leq s$ and another for $s < k \leq S$, to keep the graph continuous at $k = s$ when $w = 0$. If o' takes two values, the graph becomes discontinuous at $k = s$ when $w = W$. This result conflicts with the measured result, as shown in Figure 5. Therefore, when $w = W$, if the graph is continuous at $k = s$, time T_2' should be represented as $sG_s+(k-s)G_l+L$. This equation represents the performance character of the wormhole routing, which transmits contiguous packets that follow the first packet. Besides, from Figure 5, the Gap G takes two values, G_s for $k \leq s$ and G_l for $k > s$, as we described in (D3).

Third, when $w = W$ and $k \leq S$, the gradient of the graph is independent of both terms G_s and G_l (Fig. 4). That is, when $k \leq S$, if processors are loaded in imbalance, communication libraries have more effect on the performance of MPI programs than networks. In fact, in Figure 5(c), when $w = 5000$ and $k \leq 16383$, the communication bandwidth is 292 Mbit/s and exceeds the theory bandwidth of the Fast Ethernet, 100 Mbit/s. This benefit is produced by asynchronous protocols, that is, when *P0* calls MPI_Recv, the message from *P1* has already arrived at the buffer of *P0*, so that *P0* copies the message from own buffer.

Fourth, when $w = 0$ and $s < k \leq S$, the gradient of the graph is $2(O_s + O_r + G_l)$. If we assume that $2(O_s + O_r + G_l) > 0$, then all MPI routine costs, defined as equations (4)-(11), take positive values. This can be explained as follows. When $G_l < 0$, equation (9) is the only equation that is possible to take a negative value among equations (4)-(11), but $O_s + O_r + G_l > 0$ guarantees that equation (9) takes always a positive value. Therefore, if the assumption $2(O_s + O_r + G_l) > 0$ (that seems to be always true from Figure 4) is true, then all MPI routine costs take positive values.

Finally, when $w = W$, $D(w)$ in Figure 4 is dependent on term S. Therefore, if processors are loaded in imbalance, the performance of MPI programs become worse when the message length exceeds the value of S. Furthermore, the value of S, defined generally by MPI implementations, determines the degree of performance loss.

Besides, to derive the LogGPS parameters from a parallel platform, we compare the modeled result in Figure 4 and the measured result in Figure 5. First, both parameters L and o' can be derived by solving two equations, which are yielded from two vertical axis segments. Similarly, solving four equations, yielded from four gradients, can derive the rest of parameters. The equations on Myrinet are as follows ($W = 5 \times 10^5 ns$).

$$
\begin{array}{lll}
4o' + 2L = & 2.850902 \times 10^4 & (ns) \\
2o' + W = & 5.130990 \times 10^5 & (ns) \\
O_s + O_r = & 9.430108 & (ns) \\
2(O_s + O_r + G_s) = & 4.981455 \times 10^1 & (ns) \\
2(O_s + O_r + G_l) = & 1.737150 \times 10^1 & (ns) \\
2O_s + O_r + G_l = & 1.554669 \times 10^1 & (ns)
\end{array}
$$

4.4 An Example of Synchronization Analysis

In this section, we present an example of synchronization analysis using the LogGPS model. The MPI program we used is the Gaussian elimination program as mentioned in §4.1. See [18] for the detail of the program.

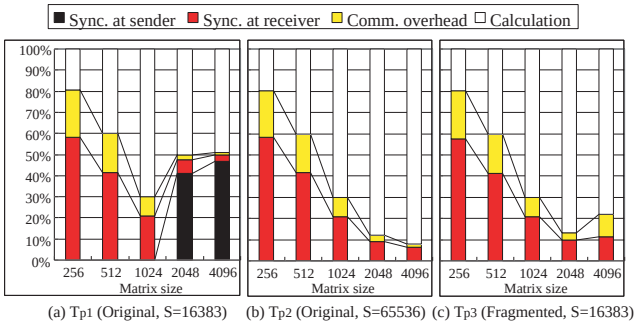
To analyze the program, we developed a tool. The tool requires two inputs, the LogGPS parameters and trace data, and outputs the synchronization cost and predicted times under the LogGPS model. The trace data is generated by running the target MPI program and consists of a sequence of time-stamped MPI events. An event has the following data: the call and return time of the MPI routine that recorded the event and the ID of the processor that called the routine and the arguments of the routine. For want of space, the detail of the tool is omitted.

First, to generate trace data, we executed the Gaussian elimination program on our cluster (Myrinet) with different matrix sizes: $n = 128$ to 4096. Second, we gave the tool two inputs: the generated trace data and the LogGPS parameters on Myrinet (see Table 5). Finally, the tool supplied us the synchronization costs and the predicted times under the LogGPS model, as shown in the left column of Table 7. The column shows (1) the measured times of the program, T_{m1},

Table 7: Measured and predicted times for the Gaussian elimination program (LogGPS)

Matrix size	Original code, S=16383			Original code, S=65536			Fragmented code, S=16383		
	Meas.	Pred.	Err.	Meas.	Pred.	Err.	Meas.	Pred.	Err.
n	T_{m1}	T_{p1}	σ_1	T_{m2}	T_{p2}	σ_2	T_{m3}	T_{p3}	σ_3
256	0.047	0.050	6.4	0.047	0.050	6.4	0.047	0.050	6.4
512	0.121	0.128	5.8	0.121	0.128	5.8	0.121	0.128	5.8
1024	0.612	0.592	- 3.3	0.612	0.592	- 3.3	0.612	0.592	- 3.3
2048	7.668	7.600	- 0.9	4.403	4.328	- 1.7	4.512	4.401	- 2.5
4096	62.038	62.188	0.2	35.902	33.200	- 7.5	43.795	39.571	- 9.6

Times in seconds and errors in percentage



(a) T_{p1} (Original, S=16383) (b) T_{p2} (Original, S=65536) (c) T_{p3} (Fragmented, S=16383)

Figure 6: Breakdowns of predicted times (LogGPS)

(2) the predicted times under the LogGPS model, T_{p1}, and (3) the errors, σ_1, defined as $\sigma_1 = 100 \times (T_{p1} - T_{m1})/T_{m1}$. The breakdowns of the predicted times, averaged over all processors, are also shown in Figure 6(a)

From errors σ_1, the LogGPS model predicts the execution time within 7% errors. In addition, under all values of n except 1024, the synchronization costs account for roughly 50% of the predicted times (Fig. 6(a)). Therefore, the synchronization cost seems to be a performance bottleneck of this program. Furthermore, when $n \leq 1024$, the receiver synchronization costs are the major, in contrast, when $n > 1024$, the sender synchronization costs are. In this program, the length of messages became long as matrix size, n, grew up, and when $n = 1024$, the longest message was 16320 bytes near to the value of S, 16383 bytes. Therefore, when $n > 1024$, as n grew up, messages were sent by Rndv. protocol, and the sender synchronization cost increased.

Note that the value of S is 128000 bytes in the MPI implementation used through the contest. Thus, we think that the sender synchronization cost was not a performance bottleneck of the program when the contest held. In this time, the LogGPS model allows us to detect the hidden performance bottleneck by analyzing its synchronization cost with a small value of S.

Thus, the LogGPS model allows us to divide MPI routine costs into three inner costs, the sender synchronization cost, receiver synchronization cost and communication overhead, so that we can investigate the performance of MPI programs in detail.

4.5 An Example of eliminating the sender synchronization cost

In this section, we try to eliminate the bottleneck detected in §4.4. Three methods can be employed to do this. From an easy method, the methods are as follows.

(M1) Change the value of S to a longer length by runtime option.

(M2) Fragment all messages into small messages, where the length of each small messages is at most S.

(M3) Change the algorithm.

First, method (M1) can be employed on the MPI implementation that can change the value of S by runtime option and avoids modifying the program. Second, method (M2) can be employed on any implementations but needs to modify the program to communicate all messages within length S. Third, method (M3) can also be employed on any implementations but needs to modify the algorithm to reduce synchronization costs and re-implement the program.

To eliminate the bottleneck, we can use the analysis as follows. (1) Predict the performance with the actual value of S, and this let us know what cost is paid for synchronization (our prediction method currently needs a run of the target program). If we find significant synchronization cost from the predicted result, then we have a chance to improve the performance using this analysis. (2) Re-predict the performance with the bigger value of S, and this let us know what performance will be archived when eliminating the bottleneck. (3) If the predicted performance becomes better, we can apply method (M1) or (M2) to improve the real performance. Otherwise, we should try method (M3). In the following, we describe methods (M1) and (M2).

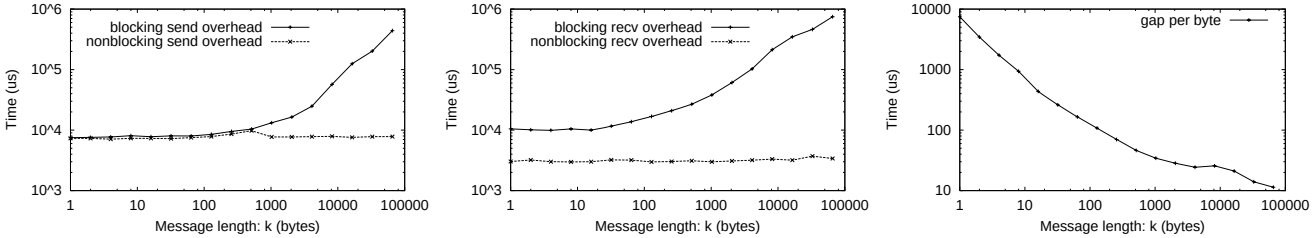
First, we investigated the longest message from the trace data, and the length proved to be 65472 bytes when $n = 4096$. We then re-analyzed its synchronization costs with $S = 65536$ using the same trace data generated in §4.4. We also re-executed the program with $S = 65536$ by specifying the runtime option of MPICH-SCore (without any modification of the program). The center column of Table 7 and Figure 6(b) show the results.

From T_{m1} and T_{m2} in Table 7, the change of S affects the measured time, T_{m2}, when $n > 1024$. The same results can be seen between the predicted times under the LogGPS model, times T_{p1} and T_{p2}. From Figure 6(a) and (b), the elimination of the sender synchronization cost shortened the predicted times, T_{m1}, and the ratio of the synchronization cost dropped from 50% to 7% when $n = 4096$.

Second, we describe method (M2). From Table 3 and Table 5, the communication cost of an S-byte message is calculated as 288 μs on Myrinet ($S = 16383$). On the other hand, when $n = 4096$, the total amount length of messages that exceed the value of S (16383 bytes) was 22.5 MBytes. Then, to send all fragmented messages, processors have to repeatedly send the fragmented messages for 1438 times. The total amount communication cost for these messages is

Table 8: Comparisons between the LogGPS model and the LogGP model (k: message length)

Item	LogGP-1 [1, 14]	LogGP-2 [17]	LogGPS
(D1) Synchronization	Constants for every k (included in o)	Model separately	Calculated by the call time of MPI routines
(D2) Overhead o	Constants for every k, divided between Ps and Pr	Two constants, divided between $k \leq s$ and $k > s$	Two linear functions on k, divided between Ps and Pr
(D3) Bandwidth $1/G$	Constants for every k	Two constants, divided between $k \leq s$ and $k > s$	Two constants, divided between $k \leq s$ and $k > s$



(a) Send overhead (b) Recv overhead (c) Gap per byte

Figure 7: LogGP-1 parameters for Myrinet

calculated as 0.42 seconds, and this is much smaller than the synchronization cost, 29.128 seconds when $n = 4096$. Therefore, by fragmenting messages, we have a chance to improve the program performance.

Then, we modified the program to repeatedly send and receive messages within length S. Then, we re-executed the modified program and re-analyzed. The right column of Table 7 and Figure 6(c) show the results.

As shown in the measured times, T_{m3}, we also improved the performance by method (M2). The predicted times, T_{p3}, show a trend similar to T_{m3}. Note that errors σ_3 are larger than errors σ_1 and σ_2 when $n = 4096$. This is explained as follows. If we employ method (M2), then the number of asynchronous messages grows up, and this loads the buffer of the receiver. To avoid the buffer to overflow, MPI implementations control messages inside MPI routines. However, as the LogGP model does, the LogGPS model also assumes that no delay occurs when copying messages into a buffer. Therefore, the predicted times under the LogGPS model, T_{p3}, are smaller than the measured times, T_{m3}.

4.6 A comparison between the LogGPS Model and the LogGP Model

In the following, we present a comparison between the LogGPS model and the LogGP model. The program used for the comparison is the Gaussian elimination program used in the pervious section.

The definitions of MPI routine costs under the LogGP model follow the definitions shown in [1, 14] and [17]. We call the former LogGP-1 and the latter LogGP-2 in the following. Table 8 lists the differences among the models, from the viewpoint of differences (D1), (D2) and (D3) described in §3. Note that the LogGP-1 model requires deriving both parameters o and G for every message length. Furthermore, parameter o must be derived for every MPI routine.

First, according to each literature, we derived LogGP parameters. Table 9 and Figure 7 show the derived parameters. Second, we analyzed the synchronization cost of the program, as we did for the LogGPS model. The trace data we used were the same as for the LogGPS model. In precise

Table 9: LogGP parameters for Myrinet

Model	Parameter	Message length (bytes) ≤ 8191	> 8192
LogGP-1	L (ns)	7.46×10^3	
	o (ns)	see Fig. 7(a) and (b)	
	G (ns)	see Fig. 7(c)	
LogGP-2	L (ns)	7.00×10^3	
	o (ns)	3.83×10^3	72.88×10^3
	G (ns)	24.98	8.85

g is unused in both models

way, we must build a synchronization model for the LogGP-2 model, but for the accuracy of the analysis, we analyzed the synchronization cost in the same way of the LogGPS model. Table 10 and Figure 8 show the results.

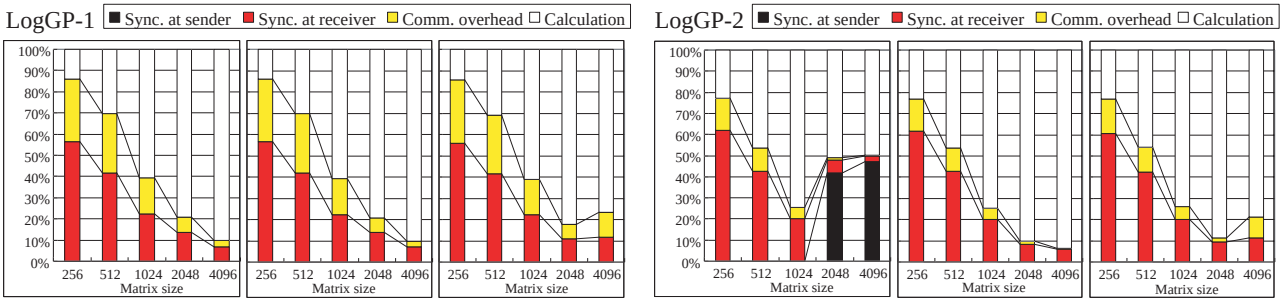
Now we discuss about the results shown in Table 10 and Figure 8. First, when $n = 4096$, the LogGP-1 model is inaccurate (σ_4=-45.6%). This is due to the lack of consideration for the sender synchronization cost. The LogGP-1 model includes the sender synchronization cost into the send overhead, o_s, and defines o_s as a constant value for every message length. Therefore, given a synchronization-bottlenecked program, the model cannot predict its bottleneck (Fig. 8(a)), and the results become inaccurate. In contrast, the LogGP-2 model builds a synchronization model apart from the communication cost, so that the LogGP-2 model is nearly as accurate as the LogGPS model (σ_7=-0.7%). Summarizing the above discussions, both the rise of bandwidth and need of synchronization are the dominant factor in long messages. Therefore, to analyze the communication costs of long messages in accurate, models should consider not only the rise of bandwidth but also the need of synchronization.

Second, when $n \leq 1024$, the LogGP-1 model is also inaccurate (σ_4=-46.8%). This is due to the method for deriving the parameters, as described in [1, 14]. Their method calls a measurement routine at each MPI call. Therefore, the measurement cost could be high compared to the target, or the cost of MPI routine, especially when the routine sends

Table 10: Predicted times for the Gaussian elimination program (LogGP)

Matrix size n	Original code, S=16383				Original code, S=65536				Fragmented code, S=16383			
	LogGP-1		LogGP-2		LogGP-1		LogGP-2		LogGP-1		LogGP-2	
	Pred. T_{p4}	Err. σ_4	Pred. T_{p7}	Err. σ_7	Pred. T_{p5}	Err. σ_5	Pred. T_{p8}	Err. σ_8	Pred. T_{p6}	Err. σ_6	Pred. T_{p9}	Err. σ_9
256	0.069	46.8	0.043	- 8.5	0.069	46.8	0.043	- 8.5	0.069	46.8	0.043	- 8.5
512	0.165	36.4	0.110	- 9.1	0.165	36.4	0.110	- 9.1	0.165	36.4	0.110	- 9.1
1024	0.668	9.2	0.551	- 10.0	0.668	9.2	0.551	- 10.0	0.668	9.2	0.551	- 10.0
2048	4.528	- 37.4	7.486	- 2.4	4.528	8.9	4.225	- 4.0	4.609	2.1	4.284	- 5.0
4096	33.805	- 45.6	61.591	- 0.7	33.805	- 5.8	32.764	- 8.7	40.170	- 8.3	39.129	- 10.7

Times in seconds and errors in percentage



(a) T_{p4} (Original, S=16383) (b) T_{p5} (Original, S=65536) (c) T_{p6} (Fragmented, S=16383) (d) T_{p7} (Original, S=16383) (e) T_{p8} (Original, S=65536) (f) T_{p9} (Fragmented, S=16383)

Figure 8: Breakdowns of predicted times (LogGP)

a short message. In fact, the measurement cost, measured by consecutively calling MPI_Wtime, is 4 μs on our cluster, and this value is near to the MPI_Recv cost, 6.55 μs (length $k = 1$). Then, we re-measured the parameters using CPU counter (the Pentium RDTSC instruction), which requires lower overhead, but the predicted results were within 27.7% errors. Therefore, the target for measurement should be bulked, and its cost should be derived by calculating the cost per call. Furthermore, bulking the MPI routines must be performed without loading the buffer of the receiver. To do this, we measured the MPI routine costs indirectly, that is, we measured RTT and not each MPI routine. In the RTT measurement program, processors synchronize at every round trip, so only one message is stored in both buffers at every round trip. Therefore, even if processors repeatedly process the round trip operation, the buffer of the receiver is kept to be unloaded, so that we can measure the bulked costs in accurate.

In contrast, when $n \leq 1024$, from errors σ_7, the LogGP-2 model is accurate (maximum of -10.0%) but inferior to the errors of the LogGPS model, σ_1 (maximum of 6.4%). This is caused by the representation of the overhead, which the LogGP-2 defines as a constant. That is, given a communication-bottlenecked program that communicates many short messages, the LogGP-2 model predicts its communication cost lower than measured cost. The above discussions are summarized as follows. The overhead is the dominant factor in short messages, so that to analyze the cost of short messages in accurate, models should represent the overhead cost as precise as possible, and a linear function of message length is one good choice.

Besides, the only difference among the models is the method for deriving parameters and the definitions of MPI routine costs, so that we confirmed that these differences greatly affect the accuracy of the model.

5. CONCLUSIONS

In this paper, we presented a new parallel computational model, the LogGPS model, which is useful to analyze synchronization costs of parallel programs. To get better accuracy for programs written using high-level communication libraries, the LogGPS model has two additional parameters (G and S) compared with the original LogP model. Parameter S represents the disadvantage of the special support for long messages, the need of synchronization, while parameter G does the advantage, the raise of bandwidth, as the LogGP model has done.

We also presented an example of synchronization analysis. Through the analysis, we confirmed that (1) the sender synchronization cost is a performance bottleneck in MPI programs and (2) the LogGPS model is more accurate than the LogGP model. Therefore, parameter S is an important parameter to predict the performance of programs written using high-level communication libraries, which automatically switch communication protocols. As a result, synchronization analysis is important when improving the performance of parallel programs.

Finally, the work, to capture the buffer mechanism that controls asynchronous messages, is remained.

6. ACKNOWLEDGMENTS

This work was supported by Grant-in-Aid for Scientific Research, JSPS Research for the Future Program JSPS-RFTF99I00903 and Kayamori Foundation of Informational Science Advancement. We acknowledge the use of the NEC Cenju-4 at NEC Parallel Processing Center. We would like to thank the anonymous reviewers for their valuable comments.

7. REFERENCES

[1] K. Al-Tawil and C. A. Moritz. LogGP Quantified: The Case for MPI. In *Proc. 7th IEEE International Symp. on High Performance Distributed Computing (HPDC-7)*, Chicago, IL, August 1998.

[2] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model — One Step Closer Towards a Realistic Model for Parallel Computation. In *Proc. 7th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'95)*, pages 95–105, Santa Barbara, CA, July 1995.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet–A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[4] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *In Proc. of Supercomputing Symposium '94 (SS'94)*, pages 379–386, Toronto, June 1994. available from http://www.mpi.nd.edu/lam/.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP'93)*, pages 1–12, San Diego, CA, May 1993.

[6] A. C. Dusseau, D. E. Culler, K. E. Schauser, and R. P. Martin. Fast Parallel Sorting Under LogP: Experience with the CM-5. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):791–805, August 1996.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996. available from http://www.mcs.anl.gov/mpi/mpich/.

[8] M. Homewood and M. McLaren. Meiko CS-2 interconnect elan – elite design. In *Proc. IEEE Hot Interconnects '93 Symp.*, pages 95–105, August 1993.

[9] IBM Corp. IBM Parallel Enviroment for AIX, MPL Programming and Subroutine Reference, Document Number GC23-3893-00, 1995.

[10] IBM Corp. IBM Parallel Enviroment (PE), http://www.rs6000.ibm.com/software/sp_products/pe.html, 2000.

[11] Y. Kanoh, M. Nakamura, T. Hirose, T. Hosomi, and T. Nakata. User-level Network Interface for a Parallel Computer Cenju-4. *Trans. of Information Processing Society of Japan*, 41(5):1379–1389, 1999.

[12] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Proc. 4th Workshop on Runtime Systems for Parallel Programming*, pages 1176–1183, May 2000.

[13] Message Passing Interface Forum. MPI: A Message–Passing Interface Standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[14] C. A. Moritz. *Cost Modeling and Analysis: Towards Optimal Resource Utilization in Parallel Compuer Systems*. PhD thesis, Royal Institute of Technology, Stockholm, 1998.

[15] F. O'Carroll, H. Tezuka, A. Hori, and Y. Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *International Conference on Supercomputing '98 (SC98)*, pages 243–250, July 1998. available from http://pdswww.rwcp.or.jp/dist/score/.

[16] Parallel Software Contest (PSC95). http://www.info.waseda.ac.jp/muraoka/project/psc95/, 1995.

[17] D. Sundaram-Stukel and M. K. Vernon. Predictive Analysis of a Wavefront Application Using LogGP. In *Proc. 7th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP'99)*, pages 141–150, Atlanta, GA, May 1999.

[18] O. Tatebe. Software, http://phase.etl.go.jp/%7etatebe/software/index-j.html, 2000.

[19] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A mechanism for integrated communication and computation. In *Proc. 19th Ann. International Symp. on Computer Architecture (ISCA)*, pages 256–266, Gold Coast, May 1992.