Exploring the Predictability of MPI Messages

Felix Freitag, Jordi Caubet, Montse Farrera, Toni Cortes, Jesus Labarta
Computer Architecture Department (DAC)
European Center for Parallelism of Barcelona (CEPBA)
Technical University of Catalonia (UPC)
{felix ,jordics, mfarrera, toni, jesus}@ac.upc.es

Abstract

Scalability to large number of processes is one of the weaknesses of current MPI implementations. Standard implementations are able to scale to hundreds of nodes, but no beyond that. The main problem of current implementations is that performance is more important than scalability and thus some assumptions about resources are taken that will not scale well.

The objective of this paper is twofold. On the one hand, we will show that characteristics such as the size and the sender of MPI messages are very predictable (accuracy above 90%). On the other hand, we will also present some examples where current MPI implementations would not work well when run on a large configuration and how this predictability could be used to solve the scalability problem.

1. Introduction

MPI (Message Passing Interface) is a specification for a standard library to address the message-passing model of parallel computation [11]. In this model, applications are divided into different tasks (or processes) that communicate by sending and receiving messages among them. This model is very widely used and it is used by applications running on distributed-memory parallel supercomputers (with high-bandwidth intercommunication systems) as well as by applications running on commodity clusters of workstations (with slower networks).

Although many implementations of MPI are already available like MPICH from Argonne National Laboratory [13], CHIMP from Edinburgh Parallel Computing Center (EPCC) [5], LAM from Ohio Supercomputer Center [10], and commercial implementations, they all suffer from, at least, one of the following problems.

First, in order to speed up communication, each process keeps a buffer for each of the other processes in the application in order to receive messages avoiding a time-consuming control flow [8]. Although this solution seems reasonable for a limited number of processes, it makes no sense if we plan to scale the number of processors to thousands. Just imagine the amount of memory it could take.

Second, it is also a typical MPI optimization to avoid any kind of control flow for short messages [13]. This kind of message is usually sent without checking whether the receiver has enough memory for it or not. Once again, this cannot scale well if thousands of nodes send a short message to the same process (collective operation). This could make the receiver to run out of memory.

Finally, many implementations use different algorithms for short and long messages. The problem is that long messages are sent using a slower protocol. It would be interesting to be able to send all messages using the fast mechanism as long as there are enough resources to do so.

Summarizing, we can see that current design and implementations of MPI are not ready for the challenge of running on machines with thousands of nodes, which is the trend to follow [12].

The goal of this paper is twofold. On one hand, we want to prove that the communication patterns of MPI applications are easy to predict. For instance, we will prove that knowing the sender of the next message and its size can easily be predicted. On the other hand, we will present possible ways to use this predictability to solve the above-mentioned problems and thus make MPI scalable to thousands of nodes.

The remainder of this paper is structured as follows. In section 2, we explain the problems, which motivate our approach. In section 3, we identify the iterative temporal characteristics in MPI message behavior, which allow the communication patterns to be predicted. In section 4 we propose the prediction scheme. Section 5 evaluates the predicability of the MPI message behavior. Section 6 discusses our results with related work. In section 7 we conclude the paper.

2. Problems to be solved and possible solutions

2.1. Memory reduction

When a message is sent to a node, it is important to guarantee that the receiver will have enough memory to keep this message. For this reason, some kind of control flow is needed. For instance, the sender should first ask permission to send the message, then wait for a confirmation to finally send a message. As this mechanism increases the latency quite a lot (3 messages are sent, but only one has useful data), many MPI implementations have tried to solve this problem by pre-allocation buffers. Each process allocates a buffer for each other process. With this mechanism, the sender knows that a message can be sent without having to ask permission (reducing the latency significantly).

This mechanism has the drawback that it does not scale well, which it is our objective. Just imagine that each process allocates a 16KB buffer for each other process (as done by the IBM MPI implementation). If we have 10000 nodes (like in the IBM Blue Gene), this process will need to allocate 160MB of memory per process.

A possible solution would be very simple if we could predict which nodes are the ones that usually send the messages in each part of the application execution. Each process could only allocate the buffers that are really needed and inform the other nodes about it. Then the mechanism would continue like always but only a small number of buffers would be needed. In case of a miss-prediction (i.e. a message is sent form an unexpected node) the slow mechanism of asking permission could be used.

2.2. Control flow

Also in the trend of avoiding "unnecessary" control flow messages, many MPI implementations assume that the receiver has enough memory to receive a short message (normally less than 16KB) and just sends it (as it is done by MPICH). In this case, the receiver has to allocate the memory for this message and keep it.

Once again, the assumption that the receiver will have enough memory is reasonable if the number of nodes is not too big. If many nodes send to the same receiver a large number of small messages, the receiver may run out of memory and thus the sent messages will be lost or, even worse, the application may crash.

If we could predict which processes are going to send messages and which sizes they will be, we could pre-allocate the buffers and inform the senders before the message is sent. Then, the sender can send the message knowing that the receiver has already allocated the buffer. In case the receiver is not able to allocate the buffer (or this message was not predicted), this credit will not be sent and thus the sender will not send the message directly, but it will ask permission first. It will be a slower send, but the memory problem in the receiver side will be avoided.

2.3. Different protocols depending on the size of the message

Finally, we have always made an implicit difference between short and long messages. A short message can be sent directly to the receiver without having to ask permission (in most cases), but a large message always needs a *rendezvous* mechanism to check that the receiver is ready to get the message. This extra communication increases the time (latency) needed to send a long message. This mechanism is used because the sender cannot assume that the receiver will have enough memory to keep the message.

We would see a very different scenario if the sender could be sure that the receiver has enough memory to keep the large message. The solution to this problem is for the receiver to predict that a large message will come from a given sender, then allocate the necessary memory and then inform the sender (before it even knows such a message is to be sent) that the necessary memory is already available. Then, when the sender decides to send the message, it knows that the receiver has already allocated the memory for this message and sends it without asking for permission. Thus, the message is sent as if it were a short one and takes advantage of a faster send mechanism.

3. Experimental framework

3.1. Logical and physical communication

In order to characterize the message stream behavior we instrument the MPICH implementation [13] at two levels: the logical and the physical communication level.

To obtain the logical communication data we trace the MPI calls from the application code to the top level of the MPI library. These calls directly reflect the application structure. The execution of loops in the application code will be seen as iterative patterns in the calls to the MPI library. We can describe the logical communication data as a function of the application code.

To obtain the physical communication data we trace at the low level of the MPI library implementation. Our tracing points show at what time messages are actually received, from which sender process and which is the message size. These calls reflect the application structure, but also random effects in the physical data transfer between processes, load balance, network congestion, and so on. We can describe the physical communication data as a function of the application code and random effects in different parameters.

3.2. Benchmarks used

In our experiments we use applications from the NAS and the ASCI benchmark suites. We use the NAS BT, CG, LU, IS [4], and the ASCI Sweep3D application [14]. Other applications from the NAS benchmarks were evaluated, but are not included in this study for two reasons: Either their communication pattern was very simple or the number of messages was very small.

In Table 1 several characteristics of the benchmarks used are given. Column two shows the number of processes we executed the applications with. In the third, fourth and fifth column the number of point-to-point, collective, and total number of messages received by a process is shown. The remaining columns indicate the number of different message sizes and different sender processes appearing in the MPI message stream received by a process[1].

Table 1. MPI applications used for this study.

Appli-cation	# of proc.	# of P2P msgs	# of coll. msgs	# of msg. sizes	# of sender
NAS BT	4	2416	9	3	3
	9	3651	9	3	7
	16	4826	9	3	7
	25	6030	9	3	7
NAS CG	4	1679	0	2	2
	8	2942	0	2	2
	16	2942	0	2	2
	32	4204	0	2	2
NAS LU	4	31472	18	2	2
	8	31474	18	4	2
	16	31474	18	2	2
	32	47211	18	4	2
NAS IS	4	11	89	3	4
	8	11	177	3	8
	16	11	353	3	16
	32	11	705	3	32
Sweep 3D	6	1438	36	2	3
	16	949	36	2	2
	32	949	36	2	2

In Table 1 it can be observed that the message stream of these benchmarks typically is of a few thousand messages, with exception of the LU benchmark, which has tens of thousand of messages. Four of the applications (BT, CG, LU, and Sweep3D), use mainly point-to-point messages. In fact, the CG benchmark has only point-to-point messages. The NAS IS benchmark, on the other hand, is an example, which uses almost only collective messages.

The applications are executed on an IBM RS/6000 in a dedicated environment. We use the class A problem size of the NAS benchmarks.

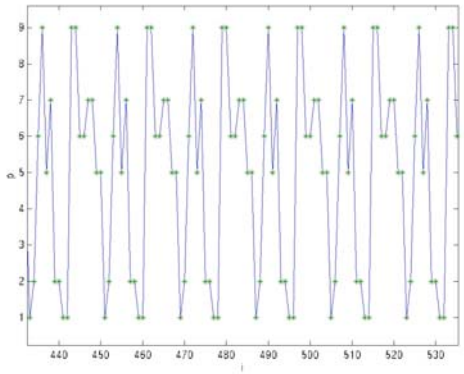
4. Design of the predictor

4.1. MPI message stream characteristics

The first issue we have to examine is the existence of some kind of iterative pattern in the MPI message stream, which would be a strong argument in favor of predictability. In Figure 1a-c, we show a portion of the data streams of MPI receives to see if there are any iterative patterns. For this experiment, we executed the Bt application with 9 processes and show the data streams of process 3. The streams are depicted with continuous lines in order to visualize the iterative pattern. The data value is indicated with a "*". It can be observed that the period of the senders, message size, and arrival time streams is 18.

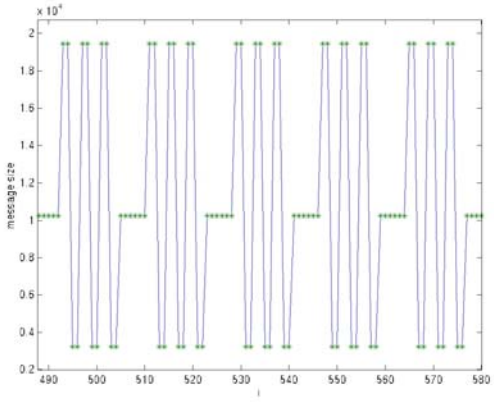
In Figure 1a, the data stream of the processes sending messages to process 3 is given. It can be seen that the sender pattern is repetitive. The senders are the processes 1, 2, 5, 7, and 9. The periodicity in the data stream is 18. Every 18 messages the pattern repeats.

In Figure 1b, the message size of the messages send to process 3 is shown. The message sizes are: 3240 bytes, 10240 bytes, and 19440 bytes. It can be seen that the order in which these message sizes occur is iterative. The message size pattern repeats every 18 messages.



a) sender process stream of process 3 of the Bt with 9 proc.

[1] For the sake of clarity we show the frequently appearing sender and message sizes.



b) message size stream of process 3 of the Bt with 9 proc.

Figure 1. Study of process 3 from the NAS Bt application executed with 9 processes. a) senders, b) message size, c) time between messages.

4.2. The prediction scheme

The objective of the predictor is to dynamically detect the iterative pattern in the MPI message streams and to predict future values. First, the predictor has to determine the periodicity of the data stream. Second, knowing the periodicity or the size of the iterative pattern allows knowing several future values. The predictor should indicate whether periodicity exists in the data stream, give the length of the iterative pattern, and predict the next future values.

Our analysis of the message stream behavior showed that the data streams we are interested in predicting contain repetitive patterns and thus this kind of prediction seems feasible. In order to perform the prediction task, we modify the dynamic periodicity detector (DPD) from [6] to enable the prediction of the data streams. The DPD allows segmenting the data stream into repetitive patterns, thus is able to capture the periodicity of the data stream. The knowledge of the periodic pattern allows predicting several future values. The predictor returns an indication whether periodicity exists in the data stream, about the length of the iterative pattern, and the prediction of the next future values.

Prediction by means of periodicity detection is a powerful technique, since the mechanism learns fast and makes use of the knowledge on the short-term temporal structure of a data stream. In contrast, predictions made by statistical models such as Markov models require more training time and although the temporal structure of the data stream can be considered to be embedded in the probability matrices, these models usually do not detect periodicities and are not prepared to predict several future values.

The algorithm used by the periodicity detector is based on the distance metric given in equation (1).

$$d(m) = sign\sum_{i=0}^{N-1} | x(i) - x(i-m) | \quad (1)$$

In equation (1) N is the size of the data window, m is the delay (0<m<M), M<=N, x[i] is the current value of the data stream, and d(m) is the value computed to detect the periodicity. It can be seen that equation (1) compares the data sequence with the data sequence shifted m samples. Equation (1) computes the distance between two vectors of size N by summing the magnitudes of the L1-metric distance of N vector elements. The sign function is used to set the values d(m) to 1 if the distance is not zero. The value d(m) becomes zero if the data window contains an identical periodic pattern with periodicity m.

The implementation of the predictor is done with circular lists, which reduces the overhead of the predictor. To have a small overhead is important since prediction has to be done at runtime. It was shown in [6] that the overhead of such an implementation is small.

5. Evaluation of the MPI message predictability

We explore the predictability of MPI message behavior with our prediction scheme. We are interested in concluding how well the logical and physical communication of MPI can be predicted. The task is to predict the next five processes, which send to a particular process and the message size of the next five messages received by a particular process. In the graphics we denote these five future values of the senders and message size +1 ... +5.

5.1. Prediction of the logical MPI communication

In these experiments, the logical communication stream (see section 3.1), as seen by the top level of the MPI library, is the input to the predictor. Figure 3 shows the prediction accuracy obtained when predicting the next five future values of the sender and message size stream.

In Figure 3 we can see that the logical communication of MPI is predicted with very high accuracy in all used benchmarks. The reason for this high predictability is that the message streams exhibit regular iterative patterns, which are captured by our prediction scheme. Since our prediction approach is based on periodicity detection, it is easy to predict several future values once the periodicity is detected. The obtained prediction rates are higher than 90 %, mostly close to 100%. Only in the NAS IS.4 we have around 80%. The reason is that the data stream with 100 samples is very short. Some data samples cannot be

predicted since a sample of the pattern has to be seen by the predictor for learning. When the NAS IS is executed with more processes and more messages are sent, high prediction rates are obtained. We can conclude from this experiment that the message stream behavior of both MPI applications with collective and point-to-point communications can be predicted very accurately at the logical level.

5.2. Prediction of the physical MPI communication

In these experiments, the physical communication stream (see section 3.1), as seen by the low level of the MPI library, is the input to the predictor. Figure 4 shows the prediction accuracy obtained when predicting the next five future values of the sender and message size stream.

In Figure 4 we can see that the physical communication of MPI is predicted with less accuracy. The reason that we do not achieve the same high prediction rates as with the prediction of the logical communication is that the message streams suffers from random effects, which make the iterative pattern less visible. Each random change of the message pattern leads to a failure in the prediction. In some applications like the NAS LU and Sweep3D we still obtain high predictability. One of the reasons is that the message and sender streams have only a few different elements, which hide the random effects. Once we have more different elements in the message streams, like in the BT benchmark, the prediction rate decreases. In the IS benchmark, prediction is very hard. This benchmark uses many collective communications, such that a process can receive from any other process. Random changes in the temporal order of messages received are frequent, for which the prediction rates are less than in the other benchmarks. We can conclude from this experiment that the prediction at the physical level of MPI communication is less accurate.

5.3. Discussion of results

The prediction of the logical communications proved to be very successful, achieving very high prediction rates. The prediction of the physical communication suffers from randomness, which reduces the prediction rates.

In Figure 2 we show by means of the BT application how the stream of message behavior can differ in the logical and physical communication. For this illustration we use the BT executed with 4 processes. We can observe in the logical communications the regular pattern of sender processes given by the sequence 00221, corresponding to the processes 0, 1, and 2, respectivly. In the physical communications this pattern can appear in the same order, or in a different order, such as indicated by the circles in Figure 5. Our predictor has no rule for these (random) changes in the pattern and cannot predict correctly, since it expects the pattern 002211.

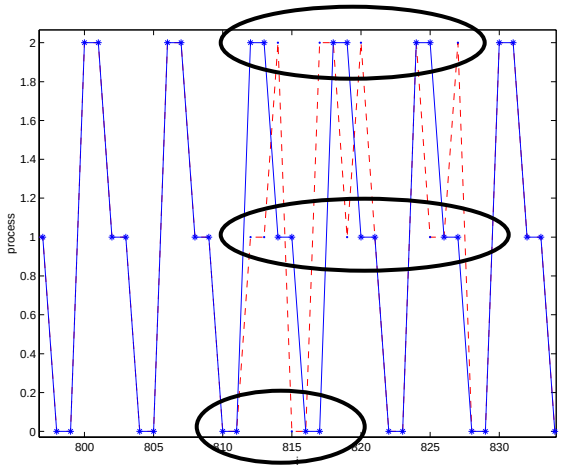


Figure 2. Sender processes to process 3 of the Bt application with 4 processes. Logical and physical communication streams.

Similarly, the message size stream on the physical level suffers from random changes, which decrease the prediction rates.

Depending on the application of the prediction, predicting the exact order of the future sender and message size may not be required. For instance, if the prediction will be used to allocate memory for message buffers of predicted senders, knowing the exact temporal order may not be necessary. Rather, knowing the next senders and their message size may be useful. This information is available with high accuracy also on the physical level, if we make use of the particular feature of this predictor, which allows several future values to be known.

6. Related work

The contribution of this work is firstly the characterization of the temporal communication patterns of MPI applications. Secondly, we proposed a prediction scheme based on periodicity detection, which achieves high prediction rates.

In [9] the communication patterns of MPI applications in the spatial domain is described (communication locality). It is found that the processes communicate only with a small number of the partners (message-destination locality). Also, it is observed that the MPI applications usually have only 2-3 distinct message sizes (message size locality). Another result is that while the problem size and the number of processes varies, the size and

number of messages varies, but the observed communication localities are quite unaffected. In our work we considered the temporal domain of the message stream. We identified the size of the iterative patterns appearing in the temporal domain. Although the spatial locality shown in [9] suggests the possibility of an iterative behavior in the temporal domain, it is not guaranteed by the spatial locality. In fact, we observed in the data streams of the physical level that communication latencies can be affected by randomness, which as a consequence affects the temporal pattern. When the communication latencies become significant compared to the computation time, randomness will not influence in the spatial locality of the messages, but the temporal message behavior may become unknown and unpredictable.

In [1, 2] the prediction of MPI messages are proposed. In this work a number of heuristics for the prediction of MPI messages are presented. The prediction heuristics are used to predict only the next value of a given data stream. The proposed application is to cache the incoming message to be nearby when the consuming thread accesses it. It was shown that the prediction heuristics provide very high hit rates for the studied streams. Compared to the work in [1, 2], the predictor we propose is based on the computation of distances between patterns in the stream. This has the benefit that the iterative pattern in the data stream is identified. In contrast to these prediction heuristics, knowing the iterative pattern allows not only the next value to be predicted, but several future values of the data stream.

7. Conclusions

We have first detected and described some scalability problems found in current MPI implementations. These problems do not appear in configurations with hundreds of nodes, but will become a huge problem for larger configurations.

We have investigated the predictability of MPI communication streams at both the physical and logical levels. We have concluded that communication streams are very predictable at the logical level, with an accuracy of over 90% (even in application with many collective operations). We have also seen that prediction at the physical level is not as accurate in some cases, due to the randomness of the environment. However, predicting the future senders and message sizes at the physical level may also be useful, if the exact temporal order of the predicted values does not have to be known.

Finally, we have proposed ways to solve the scalability problems detected by taking advantage of the high-predictability observed. Nevertheless, we are also confident that the research community will be able to use the high predictability demonstrated in this paper to solve many other problems.

References

[1] A. Afsahi, N. J. Dimopoulos. "Efficient Communication Using Message Prediction for Cluster of Multiprocessors". In *Proceedings of the Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing CANPC'00*, held in conjunction with the 6th International Symposium on High-Performance Computer Architecture, HPCA-6, January 2000.

[2] A. Afsahi, N.J. Dimopoulos. Hiding Communication Latency in Reconfigurable Message Passing Environments. Technical Report ECE-99-3. University of Victoria, January 1999.

[3] K. Al-Tawil, C.A. Moritz. Performance Modeling and Evaluation of MPI. In *Journal of Parallel and Distributed Computing,* 61(2), pp. 202-223, 2001.

[4] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, NAS Parallel Benchmark Result 3-94, *Proceedings of the Scalable High-Performance Computing Conference*, 1994, pp. 111- 120.

[5] CHIMP/MOI Project: http://www.epcc.ed.ac.uk/epcc-projects/CHIMP

[6] F. Freitag, J. Corbalan, J. Labarta. "A dynamic periodicity detector: Application to Speedup Computation". In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS 2001),* April 2001.

[7] W. Gropp, E. Lusk, N. Doss and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. In *Journal of Parallel Computing*, 22(6), pp. 789-828, September 1996.

[8] IBM Parallel Environment for AIX: *MPI Programming and Subroutine Reference,* Version 2, Release 3, IBM, 1997.

[9] J. Kim D. J. Lilja. "Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs". In *Proceedings of the Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing,* held in conjunction with the International Symposium on High Performance Computer Architecture, pp. 202-216, February 1998.

[10] LAM/MPI home page: http://www.lam-mpi.org/mpi

[11] MPI Forum. MPI: A message-passing interface standard. http://www.mpi-forum.org

[12] MPI Forum. MPI 2.0. http://www.mpi-forum.org/docs/docs.html

[13] MPICH home page. http://www-unix.mcs.anl.gov/mpi/mpich

[14] Sweep3D. http://www.llnl.gov /asci_benchmarks /asci /limited /sweep3d /asci_sweep3d.html

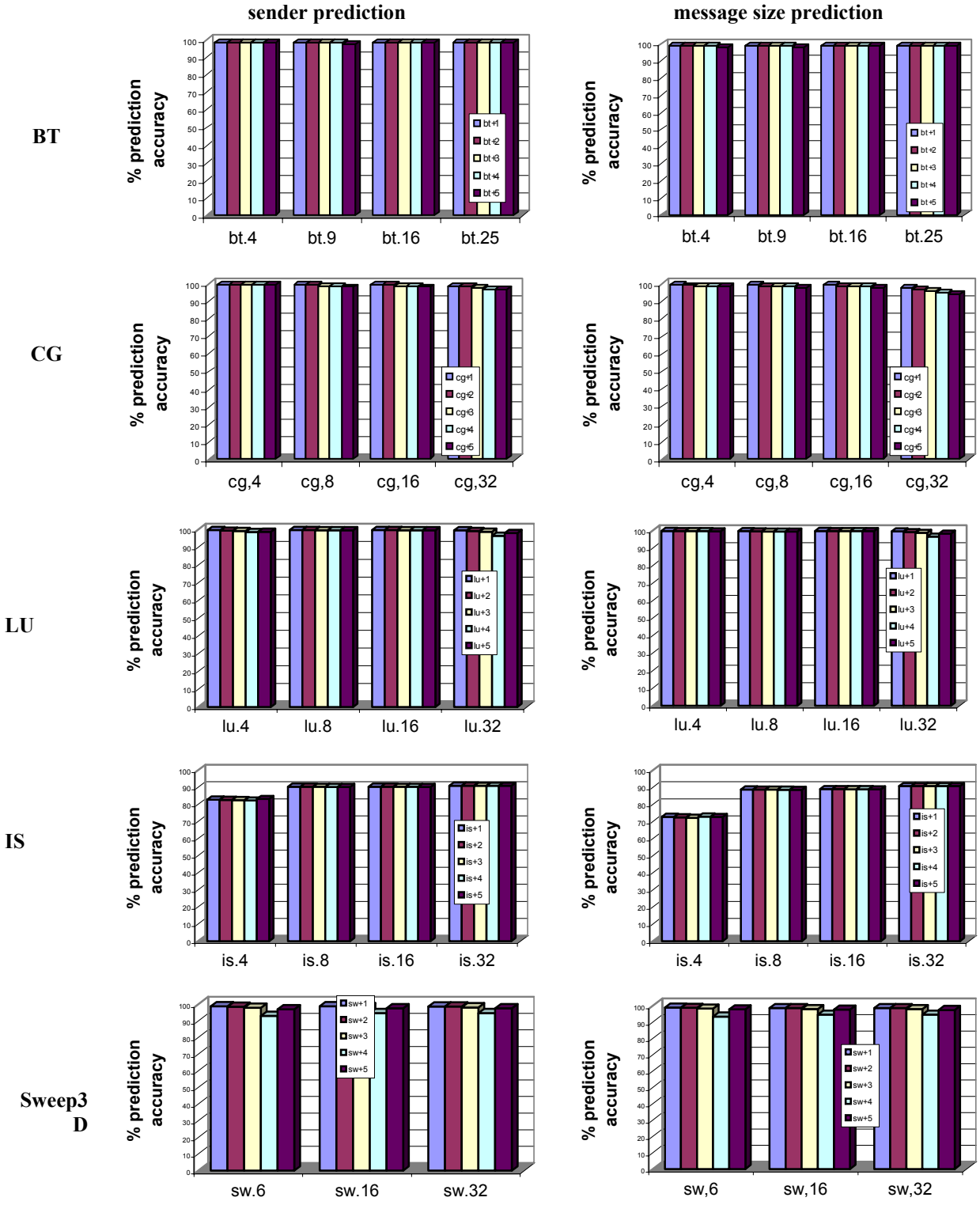


Figure 3. Prediction of the logical MPI communication.

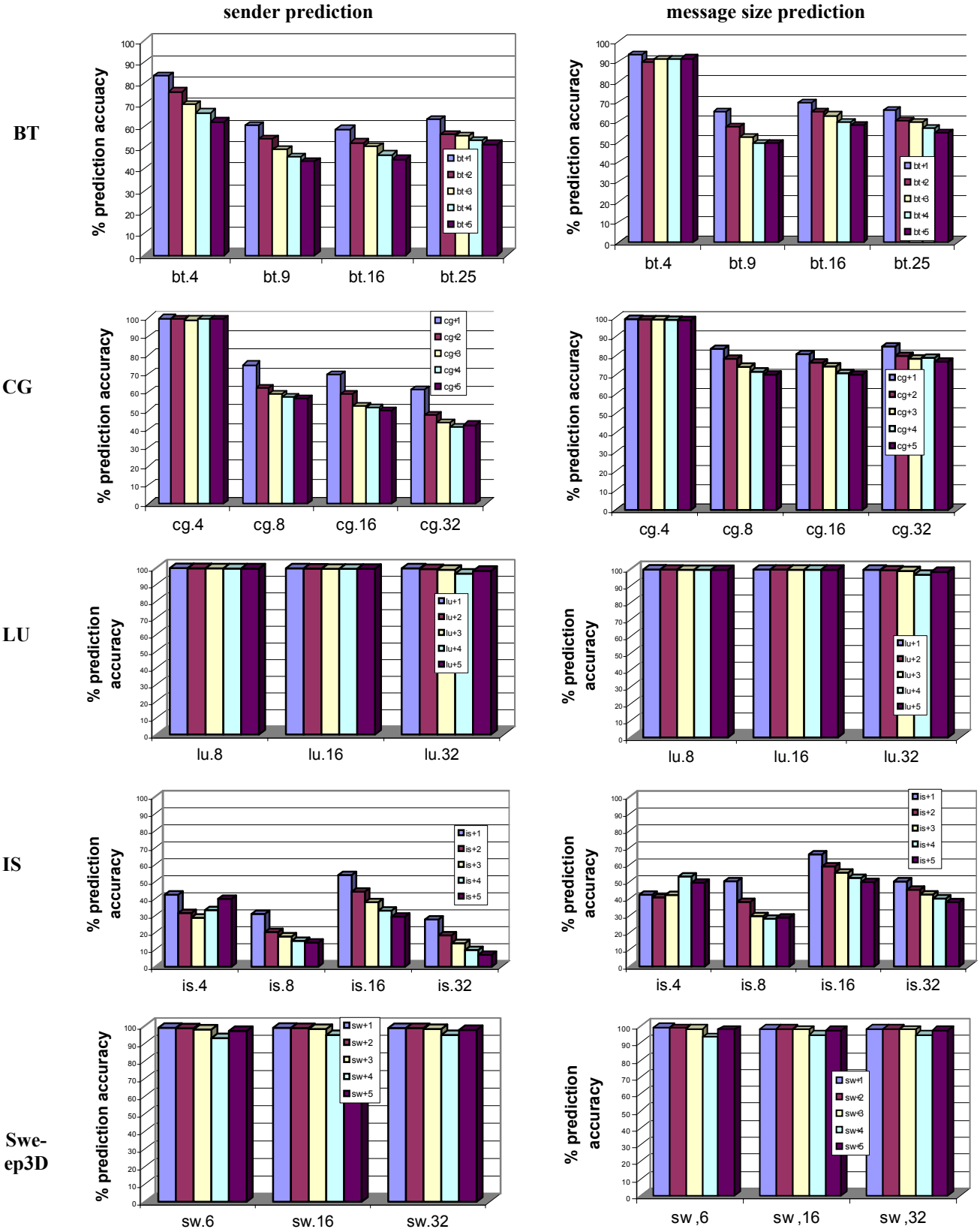


Figure 4. Prediction of the physical MPI communication.