# Algorithms for the Message Exchange Problem

A. Goldman* D. Trystram
LMC-IMAG
100, rue des Mathématiques
38041 Grenoble France
{goldman,trystram}@imag.fr

Abstract - **In a distributed memory computer system, the tasks of a parallel program spread among the processors have to communicate in order to exchange data. The communication phase can be seen as a *h-relation* whose efficient implementation allow a high performance execution of a large class of algorithms. We apply some well known linear algebra algorithms to find communication patterns, that have a very good performance in practice.**

**Keywords: Personalized communication, routing $h$-relations, BSP model.**

## 1 Introduction

Parallelization of many irregular and loosely synchronous problems leads to all-to-many personalized communication [2, 7, 9]. Within parallel programs with point-to-point communication, one possibility to improve the readability, and almost always the performance, is to group close transmissions, leaving phases of local computation. This is the basic idea of the BSP model, proposed by Valiant [10].

The bulk-synchronous parallel or BSP model, defined by Valiant [10], is a computational model introduced for improving the scalability, the portability and the easiness while developing a code on a distributed memory parallel system. It is used also for performance prediction. The BSP model main idea is to divide in time the computation and the communication in super-steps. Each super-step is composed by a computation, communication and synchronization phase. The next super-step can start only when the communications from the previous super-step have been performed. The communication phase of a super-step can also be viewed as an all-to-many personalized communication.

*Supported by CNPq (Proc. 200590/95-2). On leave from DCC-IME-USP Brazil.

To have an idea of the time necessary to complete an all-to-many personalized communication, the maximum size of the message to be sent, or received is bounded. The $h$-relation personalized communication is a message exchange problem, where each processor has to send or receive at most $h$ data.

Algorithms that implement efficiently the $h$-relation can be used to implement the communication phase of BSP algorithms and for several irregular problems. A complete discussion about this problem with complexity results and approximation algorithms was done by Goldman, Peters and Trystram [5].

### 1.1 Related works

Several algorithms for personalized communications tend to be network or machine dependent [1, 4, 6], only some of them ([12, 3]) are scalable across a number of different platforms.

Ranka and Wank [12, 13] (see also [11, 8]) proposed two methods for solving the $h$-relation. The first one uses a fixed pattern based on the linear permutation with pairwise exchanges. Other approach was a random scheduling that avoids node contention.

Bader, Helman and JáJá [3] proposed a two phases method that outperforms the previous existing algorithms. In this method, two steps of a personalized all-to-all communication, with blocks of same size, are executed.

### 1.2 Our work

Most previous algorithms for the $h$-relation use randomization, trivial permutations or two phase of communications. We are interested in algorithms that determine a permutation order to implement the $h$-relation. As the approaches using random schedulings were used to avoid contention, and as with some permutation algorithms we do not have contention, we restraint our work to deterministic algorithms.

We verify in practice, on a parallel computer, that our algorithms have very good performances compared to the other deterministic ones. This paper consists mostly in implementation and analysis of the obtained results.

### 1.3 Paper Organization

We present the known algorithms and our algorithms. We discuss the advantages and disadvantages of each algorithm. Then we present the performance results for an implementation on a IBM SP2 parallel computer. At the end we conclude with some perspectives for future work.

## 2 Algorithms for the $h$-relation

In this section we will present the deterministic algorithms for the $h$-relation. We use the same notation as Bader, Helman and JáJá [3] to define a simplified $h$-relation, given a set of $n$ elements evenly distributed among $p$ processors. Each element consists of a pair ($data$, $dest$), where $dest$ is the location to where $data$ has to be routed. No processor is the destination of more than $h$ elements, to simplify the problem all elements are supposed to have the same size. Without loss of generality assume that $h$ is an integral multiple of $p$. The processors will be labeled $P_0, \ldots, P_{p-1}$.

### 2.1 Fixed pattern algorithm

The linear permutation (LP) algorithm was proposed by Ranka, Lin and Wang [11]. The algorithm is suitable for a power of two number of processors. In the LP algorithm each processor $P_i$ sends the messages to processor $P_{(i \oplus k)}$[1] and receives the messages from $P_{(i \oplus k)}$, where $0 < k < p$ (see figure 1).

**For all processors, in parallel do**
  **for** $k = 1$ **to** $p - 1$ **do**
    $j = i \oplus k$;
    $P_i$ **sends the elements to** $P_j$;
    $P_i$ **receives the elements from** $P_j$;

Figure 1: The LP algorithm.

[1] Here $\oplus$ denotes the exclusive or operation.

As the startups time when sending a point-to-point message on a parallel computer are not negligible (see figure 2). To improve the efficiency of the previous algorithm, all the messages in processor $P_i$, to processor $P_j$, will be send within the same step.

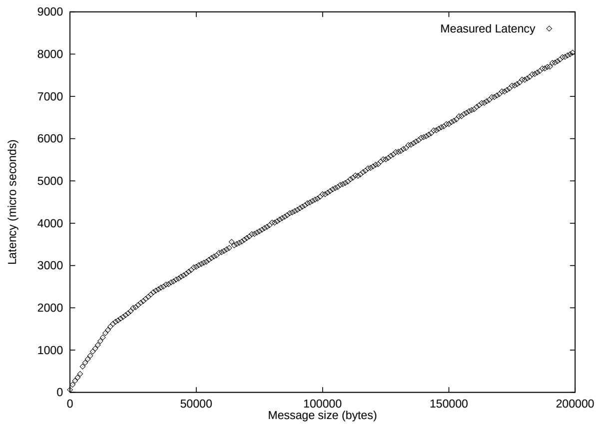


Figure 2: Latency time to send a message between two processors on the IBM SP2 without congestion.

The LP algorithm performs very well when the number of elements $h$ is near $n/p$, on the computers where the full duplex communications have good performance (e.g CM-5). A variant that works with any number of processors, and does not use pairwise communication can be found on figure 3.

**For all processors, in parallel do**
  **for** $k = 1$ **to** $p - 1$ **do**
    $j = i + k \bmod p$; $j' = i - k \bmod p$;
    $P_i$ **sends the elements to** $P_j$;
    $P_i$ **receives the elements from** $P_{j'}$;

Figure 3: Variation of the LP algorithm without pairwise exchanges.

### 2.2 Two phases algorithm

The previous solutions for the $h$-relation are a kind of sorting among the elements by destination and then a routing to the correct processor. But there exists cases that give rise to large variations of message sizes, and hence result in an inefficient use of the communication bandwidth.

To improve the bandwidth use Bader, Helman and JáJá [3] proposed a two phases algorithm. The main idea of the two phases algorithm is to use two rounds of a transpose collective communication primitive, where at each round each processor sends and receives approximatively the same number of elements.

The pseudo-code of the algorithm is as follows:

1. Each processor $P_i$, assigns its $n/p$ elements to one of $p$ bins according to the following rule: if element $k$ is the first occurrence of an element with destination $j$, then it is placed into bin $(i+j) \bmod p$. Otherwise, if the last element with destination $j$ was placed in bin $b$, the element $k$ is placed into bin $(b+1) \bmod p$.

2. Each processor $P_i$ routes the contents of bin $j$ to processor $P_j$.

3. Each processor $P_i$ rearranges the elements received on the previous step into bins according to each element final destination.

4. Each processor $P_i$ routes the contents of bin $j$ to processor $P_j$.

On the second step a transpose communication primitive with block size $\frac{n}{p^2} + \frac{p}{2}$ is performed, and on the fourth step a transpose communication primitive with block size $\frac{h}{p} + \frac{p}{2}$ is performed. So the previous algorithm depends upon a good implementation of the transpose communication primitive (like `mpi_alltoall` on the MPI programming environment).

## 2.3 Global knowledge algorithms

The most important difference between our and the previous approaches is the global knowledge of all the communication matrix in each node. In the fixed pattern algorithms there was only the need to know the size of messages that each processor should receive. In the two phases algorithm there is no need to know the messages size before the actual transmissions, as the blocks of messages have sizes that depends only on $n, p$ and $h$.

Our algorithms have a preprocessing phase where the messages are sorted in bins, according to the destination processor. We denote by $m_{i,j}$ the number of elements to be sent from processor $P_i$ to processor $P_j$. Our algorithms have two main phases, the first one with the exchange of the message sizes among the processors, and the second one with the transmission of the elements.

After the first phase each processor will have the whole communication matrix $M$ ($M = (m_{i,j})$). The main idea is that each processor will apply an algorithm to find a permutation on matrix $M$, we implemented two different strategies, the *min-max* where the minimum message size is maximized, and the *max-sum*, where the sum of messages is maximized. Both strategies use polynomial algorithms.

So our algorithms can be viewed as:

```
sort elements according to the destination
transpose with the message sizes
best strategy decision
strategy execution
```

Figure 4: Draft of the algorithms with global knowledge.

To improve the efficiency of the algorithm we overlap the calculation in the third step with the communication on the fourth step.

On the algorithms we look at the matrix $M$, as a bipartite graph $dG(V, E)$ where $V = \{P_0, \ldots, P_{p-1}, P'_O, \ldots, P'_{p-1}\}$, and there is an arc $e \in E$ from $P_i$ to $P'_j$ with weight $m_{i,j}$, if $m_{i,j} \neq 0$. So, in the following algorithms we look for a sequence of matchings, where a matching is a graph in which each node has in-degree at most one and out-degree at most one. We look for the first matching, $M_1$, in the graph $dG(V, E)$. We look for the second matching in the graph $dG(V, E) - M_1$[2], and so on. We stop when all the communications are in the sequence of matchings.

**Max-Min strategy** In this strategy, we look for a sequence of maximal matchings in which the minimum weight is maximized. The algorithm for finding such matchings is polynomial. Given a problem instance, this strategy finds a solution that solves the problem in at most $n-1$ phases.

**Max-Weight strategy** In this strategy, we find matchings in which the sum of the weights is maximal. The algorithm for finding such matchings is polynomial.

For all these global knowledge algorithms, when all the message sizes are small the first three phases can be as time-consuming as the strategy execution. When the messages to be send are about the same size, an algorithm based on the linear permutation would give better results, as there is no need to calculate the best strategy.

[2] This operation also removes arcs with weight zero.

# 3 Experimental results

In [3] the authors compared the two phases algorithm with the linear permutation one. They examined the performance of the $h$-relation algorithms on several configurations of the CM-5, IBM SP-2, Meiko CS-2, and Cray T3D. The two phases algorithm outperformed the linear permutation on almost all configurations.

We implemented the same tests as in [3], we verify the performance of the two-phases and global knowledge $h$-relation algorithms on a IBM SP-2, using four values of $h$: $h = \frac{n}{p}, 2\frac{n}{p}, 4\frac{n}{p}$, and $8\frac{n}{p}$. The data sets used in the experiments are defined as follows. Our input of size $n$ is initially distributed cyclically across the $p$ processors such that each processor $P_i$ initially holds $\frac{n}{p}$ keys, for $(0 \leq i \leq p-1)$.

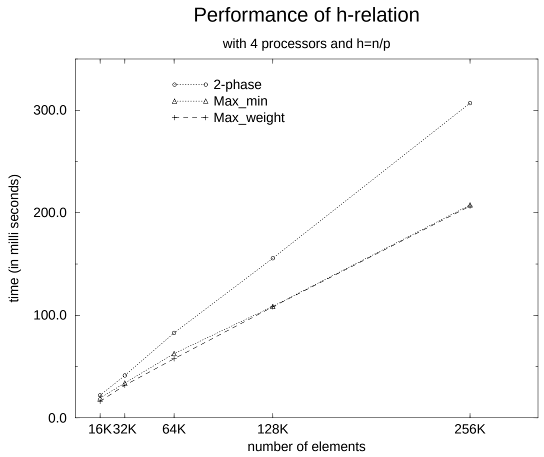


Figure 5: Performance tests on the IBM-SP2

For $h = \frac{n}{p}$ the input consists of $v_0 = \frac{n}{p}$ keys labeled for $P_0$, followed by $v_1 = \frac{n}{p}$ keys labeled for $P_1$, and so on. For $h > \frac{n}{p}$, instead of an equal number of elements destined to each processor, the function $v_i$, for $0 \leq i \leq p-1$ is done by:

$$\begin{cases} \left\lfloor h\left(1 - \frac{h}{2n-h} i\right)\right\rfloor, & \text{if } i < \frac{2n}{h}, i \neq p-1, \\ 0, & \text{if } \frac{2n}{h} \leq i < p-1, \\ n - \sum_{j=0}^{\frac{2n}{h}-1} \left\lfloor h\left(1 - \frac{h}{2n-h} j\right)\right\rfloor, & \text{if } i = p-1. \end{cases}$$

Note that in these experiments, each element consists of two integer fields, *data* and *dest*, although only the destination field *dest* is used to route each element.

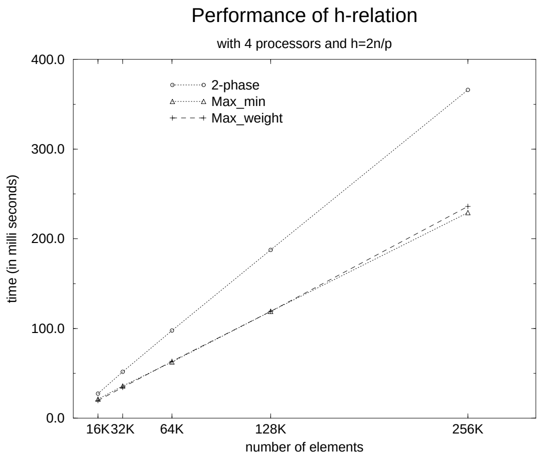


Figure 6: Performance tests on the IBM-SP2

On the performance results of figures 5, 6 we can see that for a small number of elements ($\leq 32K$) the algorithms have a similar performance. The performance difference is proportional to the number of elements, this is due to the overhead on message re-sending on the two phases algorithm and to the weight of the strategy decision phase, that is the same, independing on the number of elements. Both global knowledge algorithms have similar performances.

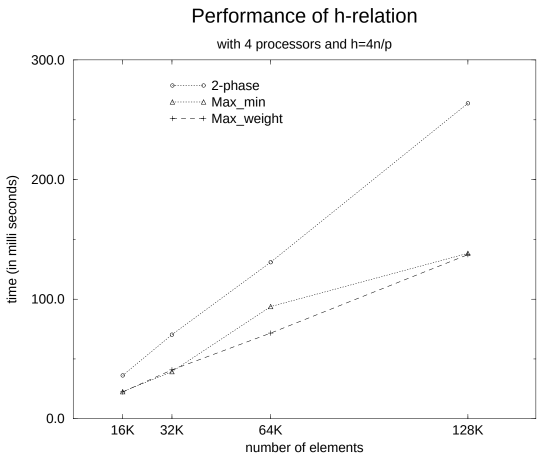


Figure 7: Performance tests on the IBM-SP2

In figure 7 we see the performance results for the $h$-relation from $16K$ to $128K$ elements, we did not perform tests with $256K$ elements because we had problems with the messages size ($> 2M$bytes). The problems were due to the asynchronous nature of the target computer and the MPI library, as to send a message of more than $2M$bytes the correspondent receive should be previously posted.

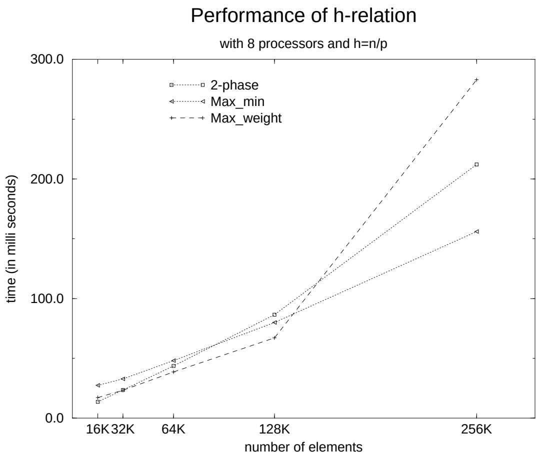


Figure 8: Performance tests on the IBM-SP2

With 8 processors, the overhead of the strategy decision on the global knowledge algorithms is more costly, so when all the processors have to send, and receive messages of the same size, we can not affirm which one is the best algorithm (see figure 8).

But as the messages to be send became more unbalanced the global knowledge algorithms starts to perform better (see figures 9, 10).

Observe that we could improve our algorithms performance, if we send only the *data* part of the element. This way we could reduce the communications by half in these experiments, where *data* and *dest* were integers coded with four bytes. Observe that we can not do this on the two phases algorithm.

It is worth to observe also that when the elements have *data* parts of different sizes, it will be necessary to do some preprocessing in order to apply the two phases algorithm. This is not the case for the global knowledge algorithms.

So, we can conclude from the experiments that the best algorithms depends on the differences on the messages to be sent/received. It depends also on the efficiency of the available transpose primitive on the target computer. On the IBM SP-2, when the messages are about the same size we can use the two phases algorithm, or the linear permutation one. When the messages size have large variation an algorithm with global knowledge will perform better.

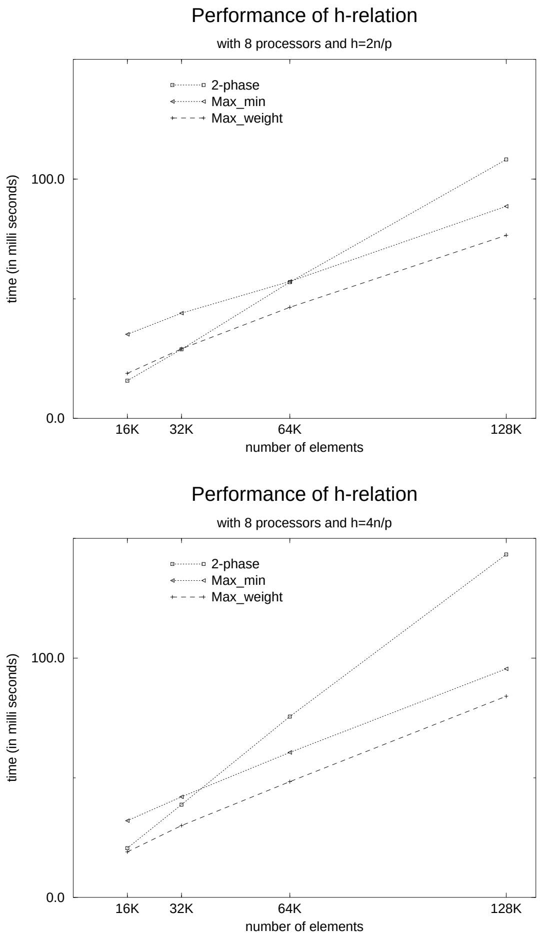


Figure 9: Performance tests on the IBM-SP2

# 4 Conclusion

In this work we proposed new algorithms for the $h$-relation problem that use some well known matrix algorithms for maximizing bipartite matchings, namely maximize the minimum weight and maximize the sum on a bipartite matching.

We verify in the experimental section that these algorithms performed better than the previous proposed deterministic ones when the messages size is not uniform.

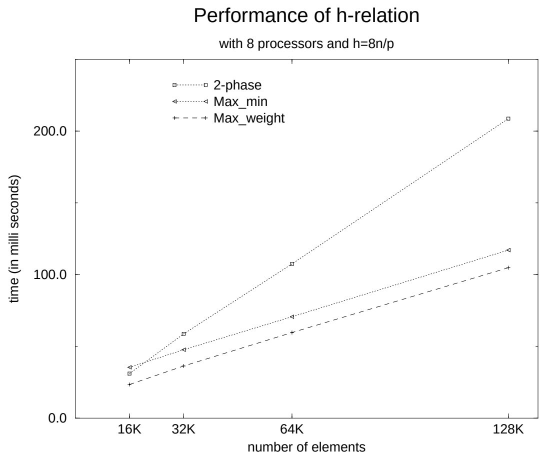


Figure 10: Performance tests on the IBM-SP2

It would be interesting to test the global knowledge algorithms on the same configurations as the two phases one, that is on the CM-5, the Cray T3D, and on the Meiko Scientific CS-2. It would be also interesting to compare the deterministic algorithms with the random ones.

**Acknowledgements:** We would like to thank Gregory Mounie for his suggestions and help.

# References

[1] B. Abali, F. Özgüner, and A. Bataineh. Balanced parallel sort on hypercube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 5(4):572–581, 1993.

[2] E. Anderson and Y. Saad. Solving sparse triangular linear systems on parallel computers. *International Journal of High Speed Computing*, 1(1):73–95, 1989.

[3] D.A. Bader, D.R. Helman, and J.JáJá. Practical parallel algorithms for personalized communication and integer sorting. Technical report, Institute for Advanced Computer Studies, and Department of Electrical Engineering, University of Maryland, November 1995.

[4] D.P. Bertsekas, C. Özveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[5] A. Goldman, J. Peters, and D. Trystram. Exchange of messages of different sizes. In *IRREGULAR'98*, volume August, to appear in LNCS, 1998.

[6] S.L. Johnsson and C-T. Ho. Optimal broadcasting and personalized communication in hypercube. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.

[7] P.C. Liewer and V.K. Decyk. A general concurrent algorithm for plasma particle-in-cell simulation codes. *Journal of Computational Physics*, 2:302–322, 1985.

[8] S. Ranka, R.V Shankar, and K.A. Alsabti. Many-to-many personalized communication with bounded traffic. In *The Fifth Symposium on the Frontiers of Massively Parallel Computation*, pages 20–27, February 1995.

[9] Y. Saad. Communication complexity of the gaussian elimination algorithm on multi-processors. *Linear Algebra Application*, 77:315–340, 1986.

[10] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[11] J-C Wang, T-H Lin, and S. Ranka. Distributed scheduling of unstructured collective communication on the cm-5. Technical report, School of Computer and Information Sience, Syracuse University, July 1993. Appeared in Parallel Processing Letters, December, 1995, special issue on Partitioning and Scheduling.

[12] J-C Wang and S. Ranka. Irregular personalized communoication on distributed memory machines. Technical report, School of Computer and Information Sience, Syracuse University, April 1993. Appeared in the Proceedings of the International Conference on Parallel Processing, 1993, August, St. Charles, IL.

[13] J-C Wang and S. Ranka. Static and runtime scheduling of unstructured communication. Technical report, School of Computer and Information Sience, Syracuse University, February 1993. Appeared in International Journal of Computing Systems in Engineering, 1993.