Exchange of Messages of Different Sizes

A. Goldman[1], D. Trystram[1], and J. Peters[⋆2]

[1] LMC-IMAG, 100, rue des Mathématiques
38041 Grenoble, France {goldman,trystram}@imag.fr
[2] School of Computing Science, Simon Fraser University
Burnaby, BC V5A 1S6, Canada peters@cs.sfu.ca

Abstract. In this paper, we study the exchange of messages among a set of processors linked through an interconnection network. We focus on general, non-uniform versions of all-to-all (or complete) exchange problems in asynchronous systems with a linear cost model and messages of arbitrary sizes. We extend previous complexity results to show that the general asynchronous problems are NP-complete. We present several approximation algorithms and determine which heuristics are best suited to several parallel systems. We conclude with experimental results that show that our algorithms outperform the native all-to-all exchange algorithm on an IBM SP2 when the number of processors is odd.

1 Introduction

In this paper, we study the exchange of messages among a set of processors linked through an interconnection network. The goal is to minimize the time to exchange all messages. The motivation for this work comes from several concrete problems for which parallel or distributed algorithms need to alternate periods of computing with periods of data exchange among the processing nodes, and for which messages of arbitrary length need to be exchanged.

One example is the solution of partial differential equations in which computing and data exchange phases are repeated until a solution is found. The amount of data exchanged among the processors depends on the chosen initial problem partition, and is not necessarily uniform. Another example is the integration over time of N-body equations in molecular dynamics simulations [14]. The behavior exhibited in computations for problems like these can be modeled by the BSP model [19]. A good BSP implementation requires a good algorithm for message exchange and synchronization.

1.1 Related work

There are many papers devoted to message exchange and similar problems. One of the first was a study about scheduling file transfers in a complete network by Coffman, Garey, Johnson and Lapaugh [8]. They used an undirected weighted

⋆ This work was done when the author was visiting LMC-IMAG in 1997.

multigraph model which did not allow either message forwarding or message splitting. They showed that computation of the optimal solution in the general case is NP-complete, and presented approximation algorithms, including some distributed algorithms, that guarantee near optimality.

Choi and Hakimi [7] investigated directed message transfer in complete networks using a weighted directed graph model in which the messages can be split in time, or among multiple arcs between two vertices. They described a polynomial algorithm to solve the problem when the start-up cost can be ignored. When the start-up cost is not negligible, the problem becomes NP-complete.

Bongiovanni, Coppersmith and Wong [4] proposed an algorithm for a similar problem. They presented an optimum time slot assignment algorithm for a satellite system with a variable number of transponders. Their model can be seen as a message exchange in a complete network in which each processor can send messages to itself, and the messages can be sent in several not necessarily consecutive parts. Other authors have studied several variations [5, 1].

Several papers consider the message exchange problem for specific interconnection structures such as meshes [18, 9, 15], and hypercubes [3]. Three algorithms for message exchange in meshes and hypercubes are proposed in [2] - one for short messages, another for long messages, and a hybrid algorithm. Most of these results concern messages of the same size.

Gonzalez was interested in multi-message multicasting [12, 13]. He worked with a fully connected network and proposed solutions with, and without message forwarding. His solutions assume that all messages have the same size.

A similar problem is TDM/WDM scheduling in passive star fiber optics networks [17]. A passive star with N nodes supports C different wavelengths (generally, $C < N$), and each node has a transmitter and a receiver. Each transmitter can be tuned, within time Δ, to any wavelength, but each receiver is assigned to just one wavelength. Traffic demand is given as an $N \times N$ matrix $D = [d_{i,j}]$ where $d_{i,j}$ is the number of packets to be sent from node i to node j. The goal is to find a valid schedule in which there is at most one transmitter using each wavelength at each instant. The problem of finding a minimum time, non-preemptive schedule was shown to be NP-hard in [11] (for $\Delta = 0$ and $C > 2$) and in [17] (for $\Delta \geq 0$ and $C \geq 2$).

In [16], the authors summarize several techniques for many-to-many personalized communication including a fixed pattern algorithm, a distributed random scheduling algorithm, and a two-stage algorithm.

We study the message exchange problem on an asynchronous model which corresponds closely to actual distributed memory machines. In the next section, we describe the problem and our model precisely and establish some lower bounds. In Section 3, we recall previous complexity results and show that the general asynchronous problem is NP-complete. In Section 4, we adapt some approximation algorithms that were developed for other problems to our all-to-all exchange problems. We analyze the algorithms, give suggestions about which algorithms are best suited to different parallel systems, and present experimental results.

2 Problem

The message exchange problem (MEP for short) can be described by a weighted digraph, $dG(V, E)$ in which the vertices represent the processors, and the arcs represent the messages to be transmitted. Each arc $e = (u, v)$ is labeled with an integer $w(e)$ that represents the size of the message to be sent from vertex u to vertex v. If several messages are to be sent from u to v, then $w(e)$ is the sum of the message sizes. The goal is to minimize the time to exchange all messages.

The port capacity of each processor is represented by two integers. For each $v \in V$, v can transmit $c_t(v)$ messages and receive $c_r(v)$ messages simultaneously. We will concentrate on the minimization problem with port capacity restricted to one for both sending and receiving, i.e. $c_t(v) = c_r(v) = 1$, for each $v \in V$. We refer to the graph $dG(V, E)$ of a message exchange problem as a *message exchange graph* (MEG for short). It is worth observing that there is not necessarily any relationship between the MEG and the topology of the interconnection network.

In addition to the labelled digraph representation described above, we will use two other represenations of MEGs - a bipartite graph representaion, and a matrix of integers. Given a digraph $dG(V, E)$ we construct a bipartite graph $bG(V' = V_1 \bigcup V_2, E')$. For each vertex $v \in V$ we have two vertices, $v_1 \in V_1$ and $v_2 \in V_2$, in V'. For each arc $e = (u, v) \in E$ there is a corresponding edge $e' = \{u_1, v_2\} \in E'$ with weight $w(e') = w(e)$.

A problem instance $dG(V, E)$ can also be represented as a square matrix A of size $|V|$, with $V = \{v_1, \ldots, v_n\}$. The elements of A are the weights on the arcs of $dG(V, E)$. If there is a message to send from v_i to v_j, then $a_{i,j} = w(v_i, v_j)$, and $a_{i,j} = 0$ otherwise.

2.1 Model

In this section we present the communication, transmission, and synchronization models. We assume that the network is fully connected. The models that we have chosen can be used to approximately model most current parallel machines with distributed memory.

Communication Model: The links between the processors are bidirectional, and each processor can use at most one link to transmit and one link to receive data at any given time.

Transmission Model: The time to send a message of size L from a processor u to another processor v, is $\beta + L\gamma$, where β is the start-up time and $\frac{1}{\gamma}$ is the bandwidth.

We will study two different message transmission properties: message splitting, and message forwarding. If message splitting is permitted, then a message can be partitioned and sent as several (not necessarily consecutive) transmissions. If message splitting is not permitted, then once the message transmission begins, it continues without interruption until completion. There are also two possibilities for message forwarding: either files must be transmitted directly between the source and the destination, or files can be forwarding along a path of intermediate processors.

Synchronization Model: The given algorithms assume an asynchronous model in which transmissions from different processors do not have to start at the same time. However, all processors start the algorithm at same time, that is, there is a synchronization before the start of the algorithm. The total time for an algorithm is the difference between the initial time and the completion time of the last processor.

2.2 Notation and lower bounds

In this section, we present theoretical bounds on the minimum number of communications and on the bandwidth for the general message exchange problem. The number of communications is determined by the number of start-ups, and the bandwidth depends on the message weights. We consider two cases - either message forwarding is permitted or it is not permitted.

Given a MEG $dG(V,E)$, we denote the in-degree of each vertex $v \in V$ by $\Delta_r(v)$, and the out-degree by $\Delta_s(v)$. Let $\Delta_r = \max_{v \in V}\{\Delta_r(v)\}$ and $\Delta_s = \max_{v \in V}\{\Delta_s(v)\}$.

Since our model does not assume any additional overhead to provide synchronization, we can not focus the study on steps as on a BSP-like model [19]. Instead, we consider the number of start-ups to be the number of start-ups that must be disjoint in time. If message forwarding is not allowed, we can compute the following straightforward bound on the number of start-ups.

Claim. To solve a message exchange problem on a digraph $dG(V,E)$ without message forwarding, the number of start-ups is at least $\max\{\Delta_s, \Delta_r\}$.

When message forwarding is allowed, a message $e = (u,v)$ can be routed along a directed path from u to v. In this case, a vertex can send one or more messages on the same link, using the same startup time, so we can derive a smaller lower bound as follows:

Claim. The number of time-disjoint start-ups needed to complete a personalized all-to-all exchange among n processors with message forwarding is at least $\lceil \log_2 n \rceil$.

Proof: This is easy to verify by observing the lower bound for broadcasting. As we assume a 1-port model, the number of informed nodes can at most double during each start-up time. ❑

Given a MEG $dG(V,E)$, the bandwidth bounds are determined by two obvious bottlenecks for each vertex - the time to send the messages and the time to receive the messages. Each vertex v has to send the messages $S_v = \{e \in E | e = (v,u), u \in V\}$ and this takes time at least $t_s = \max_{v \in V} \sum_{e \in S_v} w(e)\gamma$. The time for v to receive the messages $R_v = \{e \in E | e = (u,v), u \in V\}$ is at least $t_r = \max_{v \in V} \sum_{e \in R_v} w(e)\gamma$. The next two claims are straightforward.

Claim. If message forwarding is permitted, then the time to complete a personalized all-to-all exchange is at least $\max\{t_s, t_r\}$.

Claim. If message forwarding is not allowed, the time to complete a personalized all-to-all exchange is at least $\max\{\Delta_r, \Delta_s\}\beta + \max\{t_s, t_r\}\gamma$.

3 Complexity

In this section, we will present some complexity results about the message exchange problem. To give a clear mathematical meaning to our goal of minimizing the time, we will formally define the scheduling function.

Two different possibilities corresponding to our classification of problems will be studied - the case with no preemptive message transmissions, and the case where the messages can be split and sent as sub-messages. Both analyses assume that message forwarding is not allowed.

3.1 Indivisible messages

Given a MEG $dG(V, E)$, a schedule is defined to be a function $s : E \to [0, \infty)$ that assigns to each arc e, a starting time $s(e)$, in such a way that at each instant, for every vertex v, at most one of the arcs of v is used for transmissions and at most one arc is used for receptions. An arc e is used from the starting time $s(e)$ to the finishing time $s(e) + \beta + \gamma w(e)$.

The length of a schedule s (also called *makespan*) is the latest finishing time, that is, $\text{mak}(s) = \max_{e \in E} s(e) + \beta + \gamma w(e)$. The goal is to find the minimum schedule length $t_{\text{opt}} = min_{s \in S} \text{mak}(s)$, where S is the set of all feasible schedules.

Property 1. Given a schedule s, and an instant t, the active arcs e such that $s(e) \leq t < s(e) + (\beta + \gamma w(e))$ form a subgraph of dG, where for each vertex v, the in-degree and out-degree are each at most one.

It is worth observing that if all the arc weights are the same, the scheduling problem can be viewed as finding a set of subgraphs $G_1, \ldots, G_k$ of dG, such that each arc $e \in E$ belongs to only one subgraph G_i.

In [8], the authors studied the file transfer problem. Their model uses a graph $G(V, E)$ where the edge weights represent the amount of time to transfer a file between the endpoints of the edges, and the start-up time is not considered. They do not use the concept of send or receive, and each vertex $v \in V$, has an integer port constraint $p(v)$. Their goal was to solve the problem in minimum time using a scheduling function similar to ours, but without any distinction between transmission and reception.

Finding the minimum makespan for file transfer problems is NP-complete [8]. The problem is NP-complete even when all the edge weights are equal to one ($w(e) = 1, e \in E$) and all port capacities are equal to one ($p(v) = 1, v \in V$) (by a reduction from the chromatic index problem). The problem is also NP-complete when there are only two vertices in a multi-graph and $p(v) \geq 2$ for every $v \in V$ (by a reduction from a multiprocessor scheduling problem).

Claim. Given an instance of the message exchange problem, there is an equivalent instance of the file transfer problem.

Proof: The main idea is that when preemption is not allowed, the startup time is not meaningful. Using the bipartite representation $bG(V, E)$ of the MEG, an edge with weight $w(e)$ in the MEG will correspond to an edge with weight $\beta + w(e)\gamma$ in the file transfer graph $G(V, E')$. ❏

The following two theorems from [8] concern the difficulty of finding a minimum makespan schedule for the file transfer problem. Given a file transfer problem instance, $G(V, E)$:

Theorem 1. *An optimal schedule for the message exchange problem can be found in polynomial time when all arc weights are the same.*

Theorem 2. *If $G(V, E)$ is a tree of single port vertices, finding the minimum makespan for the file transfer problem is an NP-hard problem.*

Using the previous claim we can state the following corollary:

Corollary 1. *Optimal scheduling for the message exchange problem is an NP-hard problem.*

3.2 Divisible messages

When the messages can be partitioned and sent in several transmissions, the previous scheduling formulation cannot be used. Instead, a scheduling formulation similar to the one presented in [7] will be used. First, we present some notation.

- Given a graph g and a positive number t we define $t.g$ to be the weighted graph obtained from g by assigning weight t to each edge of g.
- Given two graphs $g_1(V, E_1)$ and $g_2(V, E_2)$ with the same set of vertices, and two positive integers t_1, t_2, we define $t_1.g_1 + t_2.g_2$ to be the graph $g(V, E)$ with edge set $E = \{(u, v)|(u, v) \in E_1 \bigcup E_2\}$. The edge $e = (u, v) \in E$ has weight equal to the sum of the weights of all edges in $t_1.g_1$ and $t_2.g_2$ that join u and v.
- Given two weighted graphs g_1, g_2 with the same vertex set, we use $g_1 \leq g_2$ to mean that g_1 is a subgraph of g_2, and the weight of each edge in g_1 is no larger than the weight of the corresponding edge in g_2.

Given an instance of the message exchange problem, $dG(V, E)$ we will consider the bipartite graph representation, $bG(V', E')$. A schedule of $bG(V', E')$ is a sequence of subgraphs $g_1, \ldots g_n$ and a sequence of positive integers $t_1, \ldots, t_n$ such that $\sum_{i=1}^{n} t_i.g_i \geq bG$. The scheduling time is $n\beta + \sum_{i=1}^{n} t_i$. The first two of the following three theorems are from [7].

Theorem 3. *When $\beta = 0$, an optimal schedule for $bG(V', E')$ can be found in polynomial time.*

For a synchronous model, we have the following theorem [7].

Theorem 4. *When $\beta > 0$, the problem of deciding if there is a schedule for $bG(V', E')$ with completion time at most M is NP-complete.*

The problem is also NP-complete in an asynchronous model.

Theorem 5. *When $\beta > 0$, the problem of deciding if there is a schedule for $bG(V', E')$ with completion time at most M is NP-complete for the asynchronous model.*

Proof: The proof uses a reduction from the 3-partition problem [10].

4 Algorithms

All of the algorithms presented here have been studied in different settings. We analyze them here for the message exchange problem. We describe an algorithm as a graph $dG(V, E)$, in which we label the $n = |V|$ vertices (processors) as $p_0, \ldots, p_{n-1}$, and let $m_{i,j}$ denote the weight of the arc from p_i to p_j. Let M be the largest message size ($M = \max_{e \in E} w(e)$), and S be the smallest message size ($S = \min_{e \in E} w(e)$). We use k to denote the ratio between the sizes of the largest and the smallest messages $\left(k = \frac{M}{S}\right)$.

In our model, each processor knows the messages to be sent and their sizes, but does not know the sizes of messages to be received. In our algorithm descriptions, we will ignore the extra communications needed to determine the sizes of messages to be received.

4.1 Hypercube-like all-to-all algorithm

We present this algorithm for n a power of 2. Let $n = 2^p$ and let $b_i = (b_{i_1}, \ldots, b_{i_p})$ be the binary representation of p_i. We use $b_i^{\bar{j}}$ to denote $(b_{i_1}, \ldots, \bar{b}_{i_j}, \ldots, b_{i_p})$.

```
for t = 1 to p do
   do in parallel for all j (0 ≤ j < n)
       b_i exchange messages with b_i^t̄
```

Analysis: In this algorithm, each processor executes $\log n$ phases. In each phase, each processor exchanges its message, and all the previously received messages with one neighbor. Notice that this algorithm does message forwarding. We do not consider the times for copying and rearrangement of messages in each processor.

For the first exchange, the time is bounded by $\beta + (n-1)M\gamma$ $(\geq \beta + t_s\gamma)$. For the second exchange, the time is bounded by $\beta + 2(n-1)M\gamma$. In general, at step i the time for the exchange is bounded by $\beta + 2^{i-1}(n-1)M\gamma$. The total time of this algorithm is at most $\log_2 n\beta + (n-1)^2 M\gamma$.

Worst case: Each message m is transmitted by one processor in the first phase, by two processors in the second phase, by four processors in the third phase, and so on. After phase i the message m is known by 2^i processors, and all processors know m at the end. As message m has only one destination, there is an overhead of $n-2$ transmissions for each message. This overhead can be ignored in the case of a large start-up time with small messages. On the other hand, with large messages the overhead will be the predominant factor.

This algorithm can be improved with tighter control on which messages should be forwarded. Even with this improvement the algorithm will be $O(n)$ times slower than the bandwidth lower bound when message forwarding is not allowed. There is a clear trade-off between the β term and the γ term. To obtain an algorithm with only $O(\log n)$ phases, we have to use message forwarding; it is impossible using only direct transmissions.

4.2 Fixed pattern algorithm

```
for t = 1 to n − 1 do
   do in parallel for all j (0 ≤ j < n)
         p_j sends the message addressed to p_{j+t mod n}
         p_j receives the message from p_{j−t mod n}
```

Analysis: Each processor executes $n-1$ steps in this algorithm. In each step t, processor p_i sends (non-blocking send) its message to the processor $p_{i+t \bmod n}$. There are no conflicts because each processor sends to only one other processor and receives from one other processor at any given time. The time to complete the algorithm is $(n-1)\beta + \sum_{t=1}^{n-1} \max_i\{m_{i,i+t \bmod n}\}\gamma \leq (n-1)\beta + (n-1)M\gamma$.

Worst case: Suppose that there are only two message sizes M and S, and the algorithm behaves synchronously. If there are $n-1$ large messages such that each one will be sent in a different step, then the time will be $(n-1)\beta + (n-1)M\gamma$. In contrast, the lower bound on the bandwidth is $((n-2)S+M)\gamma$ (when each processor has to receive at most one message). Since $M/S = k$, the algorithm will be $\frac{k(n-1)}{(n-2)+k}$ times slower than the bandwidth lower bound. On the other hand, when all the messages have the same size, this algorithm produces an optimal schedule (see Theorem 1).

4.3 Centralized pattern algorithms

The general structure of these algorithms is the following.

```
all-to-all exchange of the message sizes
best strategy decision
strategy execution
```

Analysis: These algorithms have three different phases. The purpose of the first phase is to provide each processor with global knowledge of the message sizes. In the second phase, a sequential algorithm is used to find a communication pattern. The third phase is the execution of the communication pattern found. To improve the efficiency of the algorithm we overlap the calculation in the second step with the communication in the third step.

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 $dG(V,E) - M_1$[1], 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.

Uniform strategy. In this strategy, we look for a sequence of matchings in which the weights of all edges are the same. To do this, we have to do a preprocessing step to obtain a special matrix.

Given the graph $dG(V,E)$, let A be the square matrix $(m_{i,j})$. We generate a quasi-doubly stochastic matrix (a matrix with all line sums being equal) Q. Let r_i be the sum of row i and c_j be the sum of column j. Let $S = \max_{i=1}^{n}\{r_i, c_i\}$, that is $S = \max\{t_s, t_r\}$. We use the following algorithm to obtain Q.

$Q \leftarrow A$
`for` $i = 1$ `to` n `do`
  `for` $j = 1$ `to` n `do`
    `if` $i \neq j$ `then` $q_{ij} \leftarrow q_{ij} + \min\{S - r_i, S - c_j\}$

At the end, all lines (rows and columns) of Q will have sum S. Let $dQ(V,E)$ be the corresponding graph. In the graph $dQ(V,E)$, all processors have to transmit (and also receive) messages with weights that sum to S.

Find a sequence of maximal matchings, and then normalize each matching so that all edges in a matching have the same (minimum) weight. Since we are working with an auxiliary graph, there will be some *dummy* traffic, but there is no need to do the corresponding dummy transmissions. In this algorithm we suppose that the messages can be partitioned. The number of phases will almost always be greater than $n-1$ (in general $O(n^2)$). The generated schedule is optimal when there is no start-up time.

Worst case: For all these centralized pattern algorithms, when all the message sizes are small the first two phases can be as time-consuming as the strategy execution. For the uniform strategy, the start-up times are important $(O(n^2))$.

Recommendations The choice of the right algorithm depends on the parameters of the linear model (β and γ) and the average message length L.

$\beta >> L\gamma$. The hypercube-like algorithm minimizes the number of start-ups.

$L\gamma >> \beta$. The bandwidth should be used in the best way, so the centralized algorithm with the uniform strategy is the right choice.

$L\gamma \sim \beta$. If the messages have approximately the same size, the fixed pattern is a good choice. With messages of different sizes the Max-Min or Max-Weight strategies for the centralized pattern algorithm perform better.

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

4.4 Experimental results

In this section we analyze the behavior of the algorithms on a real machine. We implemented three of the algorithms on an IBM SP2 with 32 processing nodes, interconnected by a high performance switch (HiPS). The algorithms were coded in MPICH, the IBM version of MPI. The implemented algorithms were:

Step Fixed pattern algorithm

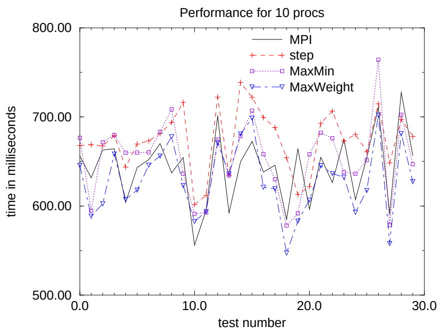
MaxMin Centralized pattern algorithm with Max-Min strategy

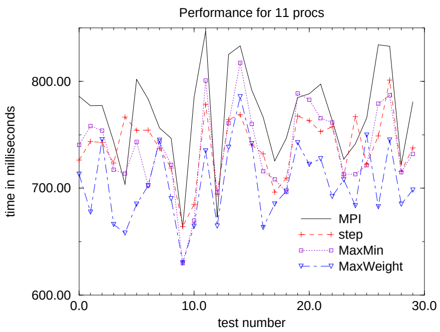
MaxWeight Centralized pattern algorithm with Max-Weight strategy

These algorithms were implemented using the send and receive primitives from MPI, without any special optimization. As a basis of comparison, we utilized the native `MPI_Alltoall` procedure. Since we did not have access to the source code of MPICH, where we expect that some low-level primitives are used, we consider performances close to the performance of `MPI_Alltoall` to be good results.

In the algorithms, there is always an initial phase in which the message sizes are exchanged. This phase is needed to allocate the receive buffers. In the first algorithm, each processor receives only the size of the messages that it will receive. In contrast, in the last two algorithms (MaxMin and MaxWeight) there is a total exchange of all message sizes. For the initial phases of all algorithms we used the fixed pattern algorithm (small messages of equal size).

We measured the times of the message exchange algorithms with the number of processors ranging from 3 to 14. For each experiment we generated a matrix composed of random message sizes following a Gaussian distribution with average 800k and deviation 400k. To avoid experimental errors, we calculated the average of 30 runs for each algorithm on each message exchange matrix. For the whole experiment, 30 different matrices were generated for each number of processors. We present below the results for 10 and 11 processors.

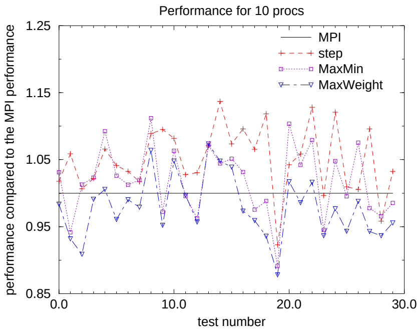


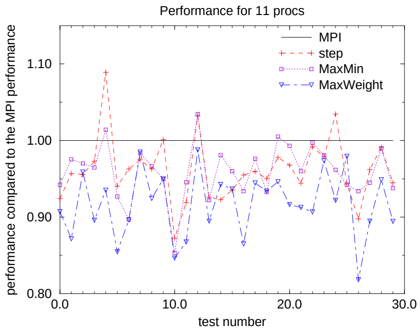


Analysis Table 1 compares the average performance of our algorithms to the performance of the MPICH algorithm. The numbers in the table are the aver-

# of procs	3	4	5	6	7	8	9	10	11	12	13	14
Step (%)	20	-6	14	-3	6	-4	3	-5	4	-5	0	-6
MaxMin (%)	19	-6	15	-3	8	-2	5	-2	4	-4	1	-6
MaxWeight (%)	19	-5	14	-2	10	1	9	2	8	0	6	-1

Table 1. Average gain compared to `MPI_Alltoall`





age percentage gains of our algorithms compared to the MPICH performances. We can see that the best performances were obtained when the number of processors was odd, and that the MaxWeight algorithm has the best performance of our three algorithms. The MaxWeight algorithm clearly outperforms the `MPI_Alltoall` primitive when the number of processors is odd.

5 Conclusions

In this paper, we investigated the problem of exchanging messages among processing nodes. We have extended previously known complexity results for the message exchange problem to show that the problem is NP-complete even in the more realistic asynchronous model.

We presented five polynomial approximation algorithms to solve the message exchange problem. The heuristics used in these algorithms are not new, but this is the first time that they have been used for the message exchange problem. Our analyses showed that the most suitable algorithm depends on machine characteristics (startup time and bandwidth), and on the uniformity of the message sizes. We implemented three of our algorithms, without message forwarding or message preemption, on an actual parallel machine. Our experiments showed that the centralized knowledge algorithms are the best ones when the message sizes are not uniform. Our algorithms outperform the native MPI routine of the machine when the number of processors is odd. We are currently implementing an all-to-all primitive for Athapascan-0 [6], a library for parallel processing with threads, that was developed in our research group.

References

1. P. Barcaccia, M. Bonuccelli, and M. Ianni. Minimun length scheduling of precedence constrained messages in distributed systems. In *EUROPAR'96*, volume LNCS 1123, pages 594–601, 1996.
2. M. Barnett, L. Shuler, R. van de Geijn, S. Gupta, D.G. Payne, and J. Watts. Interprocessor collective communication library. In *Proceedings of the Scalable High Performance Computing Conference*, pages 357–364, 1994.
3. S.H. Bokhari. Multiphase complete exchange on a circuit switched hypercube. In *Proceedings of the 1991 International Conference on Parallel Procesing*, volume I, pages 525–529, 1991.
4. G. Bongiovanni, D. Coppersmith, and C. Wong. An optimum time slot assignment algorithm for an ss/tdma system with variable number of transponders. *IEEE Transactions on Communications*, 29:721–726, 1981.
5. M.A. Bonuccelli. A polynomial time optimal algorithm for satellite-switched time-division multiple access satellite communications with general switching modes. *SIAM J. Disc. Math.*, 4:28–35, February 1991.
6. J. Briat, I. Ginzburg, and M. Pasin. Athapascan-0b reference manual. Technical Report, Apache, LMC - IMAG, Grenoble, France, 1997.
7. H. Choi and S.L. Hakimi. Data transfer in networks. *Algorithmica*, 3:223–245, 1988.
8. E.G. Coffman, M.R. Garey, D.S. Johnson, and A.S. Lapaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, August 1985.
9. P. Fraigniaud and J.G. Peters. Structured communication in cut-through routed torus networks. Technical Report TR 97-05, School of Computing Science, Simon Fraser Univ, 1997.
10. A. Goldman, J. Peters, and D. Trystram. Exchange of messages of different sizes. manuscript, www-apache.imag.fr/apache/.
11. T. Gonzales and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 4(23):665–679, October 1976.
12. T.F. Gonzales. Multimessage multicasting with forwarding. Technical Report, UCSD Department of Computer Science, TRCS-96-16.
13. T.F. Gonzales. Multi-message multicasting. In *IRREGULAR'96*, volume LNCS 1117, pages 217–228, 1996.
14. Y.S. Hwang, R. Das, J. Saltz, M. Hodoscek, and B. Brooks. Parallelizing molecular dynamics programs for distributed memory machines. *IEEE Computational Science & Engineering*, 2(2):18–29, 1995.
15. J.G. Peters and C.C. Spencer. Global communication on circuit-switched toroidal meshes. Technical Report TR 97-02, School of Computing Science, Simon Fraser Univ., 1997. to appear in PPL.
16. 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.
17. G.N Rouskas and V. Sivaraman. On the design of optimal tdm schedules for broadcast wdm networks with arbitrary transceiver tuning latencies. In *INFOCOM'96*, pages 1217–1224, 1996.
18. R. Tahkur and A. Choudhary. All-to-all communication on meshes with wormhole routing. In *IPPS'94*, pages 561–565, 1994.
19. L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.