Proceedings of the 17th IASTED International Conference
Parallel and Distributed Computing and Systems
November 14-16, 2005, Phoenix, AZ, USA

A-77

IMPROVING THE COMPUTATION AND COMMUNICATION TIME WITH BUFFERS

Teofilo F. Gonzalez
Department of Computer Science
University of California
Santa Barbara, CA, 93106, USA
email: teo@cs.ucsb.edu

ABSTRACT
We consider the multimessage multicasting over the n processor complete (or fully connected) static network when there are l incoming buffers to each processor. We present an efficient distributed algorithm to route the messages for every degree d problem instance with total communication time $d^2/l + l - 1$, where d is the maximum number of messages that each processor may send (or receive). Our algorithm takes linear time with respect to the input length. When $l = d$ our algorithm generates communication schedules with smaller total communication time than previous algorithms, even when forwarding is allowed. Furthermore, such schedules can be constructed in linear time. For $l = 1, d/2$, and d we present lower bounds for the total communication time. The lower bounds match the upper bounds for the schedules generated by our algorithm when $l = 1$ and $l = d$, and are within 25% when $l = d/2$.

KEY WORDS
Approximation Algorithms, Multimessage Multicasting, Forwarding, Buffers, Fully Connected Networks.

1 Introduction

Parallel and distributed systems were introduced to execute programs at unprecedented speeds. To accomplish this goal a program must be partitioned into tasks and the communications that must take place between these tasks must be identified to ensure a correct execution of the program. To achieve high performance one must assign each task to a processing unit (statically or dynamically) and develop communication programs to perform all the intertask communications efficiently. Efficiency depends on the algorithms used to route messages to their destinations, which is a function of the underlying communication network, its primitive operations and the communication model. Given a network with a communication model, a set of communication primitives and a set of messages that need to be exchanged, our problem is to find a schedule to transmit all the messages in the least total number of communication rounds. Generating an optimal communication schedule, i.e., one with the least total communication rounds, for our message routing problems over a wide range of communication networks is an NP-hard problem. To cope with intractability efficient message routing approximation algorithms for classes of networks under different communication assumptions have been developed. In this paper we consider the message communication problem where l buffers have been placed at the receiving end of each processor. We show that speedups with factor of about l, over the case without buffers, can always be achieved.

The Multimessage Multicasting, MM_C, problem was introduced by Gonzalez [1, 2] and Shen [3]. The problem consists of constructing a communication schedule, for an n processor static network (or simply a network), with least total communication time for multicasting (transmitting) any given set of messages. Specifically, there are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via a network N. Each processor is executing processes, and these processes are exchanging messages that must be routed through the links of N. Our objective is to determine when each of these messages is to be transmitted so that all the communications can be carried in the least total amount of time.

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i and *needs* to receive the set of messages n_i. We assume that $\bigcup h_i = \bigcup n_i$, and that each message is initially in exactly one set h_i. We define the *degree* of a problem instance as $d = \mathbf{max}\{| h_i |, | n_i |\}$, i.e., the maximum number of messages that any processor sends or receives. Consider the following example.

Example 1.1 *There are nine processors ($n = 9$). Processors P_1, P_2, and P_3 send messages only, and the remaining six processors receive messages only* [1]*. The messages each processor holds and needs are given in Table 1. For this example the degree d is* 3.

One may visualize problem instances by directed multigraphs. Each processor P_i is represented by the vertex labeled i, and there is a directed edge (or branch) from vertex i to vertex j for each message that processor P_i needs to transmit to processor P_j. The set of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1.1

[1] Note that in general processors may send and receive messages.

Table 1. Hold and Need vectors for Example 1.1.

h_1	h_2	h_3	h_4	h_5
$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\emptyset$	$\emptyset$

n_1	n_2	n_3	n_4	n_5
$\emptyset$	$\emptyset$	$\emptyset$	$\{a,c,e\}$	$\{a,d,f\}$

h_6	h_7	h_8	h_9
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

n_6	n_7	n_8	n_9
$\{b,c,e\}$	$\{b,d,f\}$	$\{c,d,e\}$	$\{c,d,f\}$

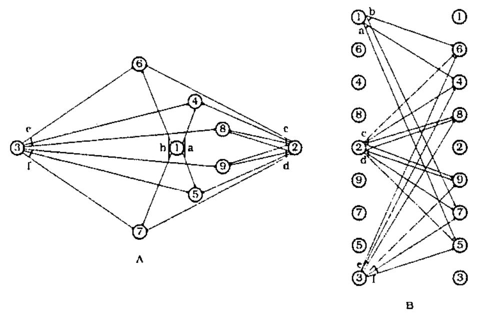

Figure 1. (A) Directed Multigraph Representation, and (B) Directed Bipartite Multigraph Representation for Example 1.1. The thick line joins all the edges (branches) in the same bundle.

is depicted in Figure 1(A) as a directed multigraph with additional thick lines that identify all edges or branches in each bundle. It is also interesting to visualize problem instances as directed bipartite multigraphs. Each processor P_i is represented by two vertices, one is the *source* and the other the *sink* for all the messages associated with processor P_i. There is a directed edge (or branch) from source vertex i to sink vertex j for each message that processor P_i needs to transmit to processor P_j. The set of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1.1 is depicted in Figure 1(B) as a directed bipartite multigraph with additional thick lines that identify all edges or branches in each bundle.

In the *single port communication mode* every processor sends at most one message and receives at most one message during each communication round. A processor may send at each communication round one of the messages it holds (i.e., a message in its hold set h_i at the beginning of the time unit), but such message can be multicasted to a set of processors. The message also remains in the hold set h_i. During each time unit each processor may receive at most one message. The message that processor P_i receives (if any) is added to its hold set h_i at the end of the time unit.

The communication process ends when each processor has $n_i \subseteq h_i$, i.e., each processor holds all the messages it needs. The *total communication time* is the total number of communication rounds. Our communication model allows us to transmit any of the messages in one or more stages. I.e., any given message may be transmitted at different times. This added routing flexibility reduces the total communication time. In many cases it is a considerably reduction.

Algorithms for the completely connected architecture have wide applicability in the sense that the schedules can be easily translated to communication schedules for every pr-network, a large family of communication networks. There is some penalty one has to pay for the translation process which is doubling the communication rounds. However, this penalty is not always incurred. When the processors are connected via a pr-dynamic network a communication mode can be performed in two stages by using standard techniques: the data replication step followed by the data distribution step. This two stage process can be used in the MEIKO CS-2 machine [2] as well as in most architectures connected through Benes type networks.

Our introduction is a condensed version of our previous papers which includes complete justification for the multimessage multicasting problem as well as motivations, applications, and examples.

There are relatively inexpensive ways to speed-up communication. One such technique consists of adding l buffers at the receiving end of each processor and developing controlling hardware so the buffering behaves as follows: (1) if at the beginning of a communication round one buffer has a message, then one such message (perhaps in a FIFO fashion) is passed to the processor and the buffer will be labeled empty for the current communication round; and (2) if there are j empty buffers during the current communication round, then up to j messages may be received and stored in these free buffers. In this paper we present an efficient algorithm to construct for every degree d problem instance a communication schedule with total communication time at most $d^2/l + l - 1$, where d is the maximum number of messages that each processor may send (or receive) and l is the number of input buffers on each processor. For $l = 1, d/2$, and d we present lower bounds for the total communication time. The lower bounds match the upper bounds for the schedules generated by our algorithm when $l = 1$ and $l = d$, and are within 25% when $l = d/2$.

2 Applications and Previous Results

The multimessage multicasting communication problem arises naturally when solving large scientific problems via iterative methods in a parallel or distributed computing environment, for example, solving large sparse system of linear equations using stationary iterative methods. Another application of multimessage multicasting arises when executing most dynamic programming procedures in a parallel or distributed computing environment. In information systems, multimessage multicasting arises naturally when multicasting information over a b-channel ad-hoc wireless communication network. Other applications include sorting, matrix multiplication, discrete Fourier transform, etc. Message routing problems under the multicasting communication primitives arise in sensor networks which are simply static or slow changing ad-hoc wireless networks. These type of networks have received considerable attention because of applications in battlefields, emergency disaster relief, etc. Ad-hoc wireless networks are suited for many different scenarios including situations where it is not economically practical or physically possible to provide Internet or Intranet wired communication. Other applications in high performance communication systems include voice and video conferencing, operations on massive distributed data, scientific applications and visualization, high performance supercomputing, medical imaging, etc. The need to deliver multidestination (multicasting) messages is expected to increase rapidly in the near future.

The case when each message has fixed *fan-out* k (maximum number of processors that may receive any given message) has been studied [2]. For $k = 1$ (the problem is called multimessage unicasting MU_C), Gonzalez [2] showed that the problem corresponds to the makespan openshop preemptive scheduling problem which can be solved in polynomial time, and each degree d problem instance has a communication schedule with total communication time equal to d [4]. The makespan openshop preemptive scheduling problem is a generalization of the edge coloring of bipartite multigraphs. Algorithms for this problem may also be used to solve the MU_C problem; however, Gonzalez and Sahni's [4] algorithm is currently the fastest method to solve the MU_C problem, whereas Cole, Ost and Schirra's [5] edge coloring algorithm is the fastest one when the edge multiplicities are small [6].

It is not surprising that several authors have studied the MU_C problem as well as several interesting variations for which NP-completeness has been established, subproblems have been shown to be polynomially solvable, and approximation algorithms and heuristics have been developed [7, 8, 9, 10, 11, 12].

With the exception of the work reported in [1, 2, 3, 13, 14, 15, 16, 17], research has been limited to unicasting and most multicasting results are limited to single messages. Shen [3] has studied multimessage multicasting for hypercube connected processors. The heuristic try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. Recently, Thaker and Rouskas [17] survey strategies for multimessage multicasting problems defined for all-optical networks.

Gonzalez [2] shows that even when $k = 2$ the decision version of the MM_C problem is NP-complete. Gonzalez [2] also shows that every degree d instance of the MM_C problem with n processors has a communication schedule with total communication time at most d^2. This bound is best possible in the sense that for all $d \geq 1$ there are problem instances that require d^2 communication time [2].

Gonzalez [13] presents a series of approximation algorithms for all $k \geq 3$ that generate solutions with $(\sqrt{k} + \sqrt{2} - 1)d$ communication time. Schedules with total communication of about $qd + k^{\frac{1}{q}}(d-1)$ for $2 \leq q \leq k$ can be constructed in $O(q \cdot d \cdot e)$ time, where $e \leq nd$ by sending each message to all its destinations during q time periods rather than just two [2].

When *forwarding* is allowed the problem is referred to as the MMF_C problem. In this case messages may be sent through indirect paths even though single-edge paths exist. At first glance it appears that forwarding will not really help deliver messages faster for the MM_C problem because forwarding consumes more resources, and the networks is complete (all the bidirectional links are present).

The reduction used to establish that the MM_C is NP-hard [2] can be easily modified to establish that the MMF_C problem is an NP-hard problem [15]. Gonzalez [15] developed algorithms to construct communication schedules with total communication time at most $2d$. in $O(r(min\{r, n^2\} + n \log n))$ time, where r is the total number of messages ($r \leq dn$). Clearly, these approximation algorithms are slower than the ones discussed above; however, they generate communication schedules with significantly smaller total communication time.

We have also studied the *distributed* version of the MMF_C, which we refer to as the $DMMF_C$ problem. Each processor initially knows only the value of n and d, and at a given time it learns the messages it will be sending and their destinations. The strategy to solve the $DMMF_C$ problem is to use the standard parallel prefix algorithm, followed by a distributed version of the forwarding algorithm in [15] and then the distributed multimessage unicasting distributed algorithm in [18]. The algorithm performs all the multicasting operations in $O(d + \log n)$ expected communication steps [14].

It is simple to see that the $DMMF_C$ problem is more general than the MMF_C and the MM_C problems, but the best communication schedule for the $DMMF_C$ problem has total communication time $\Omega(d + \log n)$ where as for the MM_C problem is d^2, and just $2d$ for the MMF_C problem. Therefore, knowing all the communication information ahead of time allows one to construct significantly better communication schedules, and forwarding plays a very important role in reducing the total communication time in our message routing problems.

3 New Results

In this paper we consider the communication architecture where there are l buffers at the receiving end of each processor and developing controlling hardware so the buffering behaves as follows: (1) if at the beginning of a communication round one buffer has a message, then one such message (perhaps in a FIFO fashion) is passed to the processor and the buffer will be labeled empty for the current communication round; and (2) if there are j empty buffers during the current communication round, then up to j messages may be received and stored in these free buffers. In this paper we present an efficient algorithm to construct for every degree d problem instance a communication schedule with total communication time at most $d^2/l + l - 1$, where d is the maximum number of messages that each processor may send (or receive) and l is the number of input buffers on each processor. Furthermore, such schedules can be constructed in linear time. For $l = 1, d/2$, and d we present lower bounds for the total communication time. The lower bounds match the upper bounds for the schedules generated by our algorithm when $l = 1$ and $l = d$, and are within 25% when $l = d/2$.

When $l = 1$ our algorithm generates a schedule with the same communication time as the one for the MM_C problem. On the other hand when $l = d$ our algorithm generates a schedule with slightly smaller communication time than the one for the MMF_C problem. Our new algorithm has the added property that it does not forward any of the messages and the time complexity to generate the solution is considerably smaller. Forwarding requires heavier link traffic since messages will be sent through more than one link. In other words, our new algorithm utilize the minimum amount of network capacity at the expense of the introduction of the buffers.

3.1 Algorithms and Lower Bounds

We show that for every degree d instance of the MM_C problem one can construct in linear time, with respect to input length, a schedule with total communication time $d^2/l + l - 1$ when there are l input buffers. Before we present this result it is convenient to consider Procedure Coloring that guarantees that all the messages will be delivered by time $d^2/l - d/l + d$. Then we give Procedure Ordered-Coloring, which is a slight modification of Procedure Coloring, but generates a solution with total communication time $d^2/l + l - 1$. This is better than the previous one since by definition $l \leq d$. But note that both procedures generate a solution with identical total communication time in the extreme cases, i.e., when $l = 1$ and $l = d$.

Let P be any n processor instance of the MM_C problem of degree d. We assume that d is a multiple of the number of buffers, l. First we define the set of d^2/l colors as follows: $\{(i,j) | 1 \leq i \leq d \text{ and } 1 \leq j \leq d/l\}$. Now assign an order $(1 \leq i \leq d)$ to all the bundles emanating from each vertex. Now assign the value of 1 to l incoming edges to each processor, the value of 2 to l incoming edges to each processor, ..., and the value of d/l to l incoming edges to each processor. Assign color (i,j) to edge $e = \{p, q\}$ if e belongs to the i^{th} bundle emanating form vertex p, and e is assigned the value of j as an incoming edge to vertex q.

Now we construct a schedule with total communication time $d^2/l - d/l + d$ as follows. All the messages colored $(i, 1)$ are transmitted at time i, for $1 \leq i \leq d$. Since for each processor there are at most l messages incoming with the color $(i, 1)$ it then follows that none of the buffers for the processors will overflow. But it may be that for one or more processors there are l messages that arrive at time d. So one needs $l - 1$ time units to empty all the buffers. So the transmission and reception of all the $(i, 1)$ messages takes $d + l - 1$ time units. The same arguments can be applied to each of the $d/l - 1$ groups of messages $(i, 2)$, $(i, 3)$, ..., $(i, d/l)$, and conclude that each group transmission can be achieved during the same communication time period. Therefore the total transmission time is $d^2/l - d/l + d$.

Theorem 3.1 *The informal algorithm described above generates a communication schedule with total communication time at most $d^2/l - d/l + d$ for every degree d instance of the MM_C when there are l buffers. Furthermore, the algorithm takes linear time with respect to the number of nodes and edges in the multigraph.*

Proof: The proof follows the same arguments discussed above. □

We now show that the bound $d^2/l - d/l + d$ is best possible for the algorithm given above by providing a problem instance for which it is tight.

Example 3.1 *For any integer $d > 1$, we define problem instance I_d as follows. The number of processors is $n = d^d d! + d$. Processors $P_1, P_2, \ldots, P_d$ just send messages and the remaining processors $P_{i,J}$ for $J = (j_1, j_2, \cdots j_d)$ and $1 \leq i \leq d$, where each $j_l \in [1 : d]$, i.e., each j_l is an integer whose value is between 1 and d. Each processor $P_{i,J}$ for $J = (j_1, j_2, \cdots j_d)$ and $1 \leq i \leq d$, receives a message from the j_lth bundle of processor P_l. Note that the for every J there are $d!$ different processors that receive their messages for the same set of d bundles.*

Theorem 3.2 *When we apply the above algorithm to the problem instance I_d results in a solution where every processor receives at least one message at each time unit from time 1 to time $d^2/l - d/l + d$.*

Proof: Lets consider first Example 3.1. Assume that the algorithm assigns the values 1 to the first l edges incoming to a vertex, the value of 2 to the second l edges incoming to a vertex, and so on. Assume that for every J all the $d!$ processors $P_{i,J}$ have a different ordering for the bundles where the edges incoming to the processor originate. Since there are d incoming edges to all of these processors and all these processors have their edges originating on the same set of d bundles, it then follows that this is always possible.

It is easy to see that the set of messages colored (i, j), for some $1 \leq i \leq d$ and $1 \leq j \leq l$, is not empty simply because every bundle coming out of a processor P_k is not empty and it has a edge that ends in the d processors $P_{i,J}$ for some i and J. The edge will be in position $1, 2, \ldots, d$ in these processors so there will be an edge colored (i, j) for all $1 \leq i \leq d$ and $1 \leq j \leq l$. Therefore, when the messages colored (i, j) are being sent, at least one processor receives a message.

We need to show that for $1 \leq i < d$ one cannot commence with the delivery of the messages colored $(i + 1, 1)$

until $l-1$ time units have elapsed after the messages colored (i,d) are delivered. Consider the $d!$ processors $P_{i,J}$, for $J=(1,1,\ldots,1,d,d,\ldots,d)$, where there are il entries with ds and $d-il$ entries with 1s. For one of the processors $P_{i,J}$ all the messages that originate at the bundles labeled d will appear before those originating at the the bundles labeled 1. Therefore the processor has l messages colored (i,d) and l messages colored $(i+1,1)$ One of the messages colored (i,d) will be received by the processors at the time it is sent, but the $(l-1)$ remaining messages will received during the next $l-1$ steps. The same arguments can be used for $1 \leq i < d$. For the case when $i=d$ one can use the same argument, i.e., that there is a processor with $l-1$ messages in its buffer at the time the last message is sent.

This completes the proof of the theorem.

□

The above theorem establishes that for the algorithm given above the total communication of the schedules generated is best possible. That does not prove that there is no algorithm that generates solutions with smaller total communication time. Further improvement to the total communication time may be achieved by overlapping the time used to empty the buffers and the transmission of the messages with higher labels. Luckily it is possible to do this by a slight modification of the algorithm given above. In what follows we present the new algorithm. We show that for every degree d instance of the MM_C problem one can construct in linear time, with respect to input length, a schedule with total communication time $d^2/l+l+1$ when there are l input buffers.

Again, we assume that d is a multiple of the number of buffers, l. First we define the set of d^2/l colors as follows: $\{(i,j) | 1 \leq i \leq d \text{ and } 1 \leq j \leq d/l\}$. Now assign an order $(1 \leq i \leq d)$ to all the bundles emanating from each vertex. Now we depart from the previous algorithm. Each incoming edge to a processor will be assigned a label which is just the index assigned to the bundle where it emanates, i.e., an integer value between 1 and d. We order all the incoming edges to each processor in ascending order of its label. This can be easily done via Radix sort in linear time. Now with respect to this order assign the value of 1 to the first l incoming edges to each processor, the value of 2 to next l incoming edges to each processor, ..., and the value of d/l to last l incoming edges to each processor. Assign color (i,j) to edge $e=\{p,q\}$ if e belongs to the i^{th} bundle emanating form vertex p, and e is assigned the value of j as an incoming edge to vertex q.

Now we construct a schedule with total communication time $d^2/l+l+1$ as follows. All the messages colored $(i,1)$ are transmitted at time i, for $1 \leq i \leq d$. Since for each processor there are at most l messages incoming with the color $(i,1)$ it then follows that none of the buffers for the processors will overflow. But it may be that for one or more processors there are l messages that arrive at time d. The beauty of the new algorithm is that one may overlap the process of emptying the buffers after all the messages colored (i,j) and the messages $(i,j+1)$ are sent for all $1 \leq j < d$. This means that the transmission and reception of all the $(i,1)$ messages takes $d+l-1$ time units, but the last $l-1$ units of time may be overlapped with the transmission of the $i,1)$ messages. The same arguments can be applied to each of the $d/l-1$ groups of messages $(i,2),(i,3),\ldots,(i,d/l)$, and conclude that each group transmission can be achieved during the same communication time period. Therefore the total transmission time is $d*(d/l-1)+d+l-1=d^2/l+l+1$.

Theorem 3.3 *Procedure Ordered-Coloring described above generates a communication schedule with total communication time at most $d^2/l+l+1$ for every degree d instance of the MM_C when there are l buffers. Furthermore, the algorithm takes linear time with respect to the number of nodes and edges in the multigraph.*

Proof: The proof follows the same arguments discussed above, except that we need to show that there is no overflow of buffers when we send the messages (i,j) for $1 \leq i \leq d$ and $1 \leq j \leq d$. The same arguments as before apply for the case of the messages colored $(i,1)$ for $1 \leq i \leq d$. Since the proof for the case for all values of j is very similar, we only prove the case for $j=2$.

Consider the case when one is sending the messages colored $(1,2)$ and let us consider any processor Q that receives any subset of those messages. Clearly all of these messages belong to bundle 1 of some processors. All of the incoming messages to processor Q were ordered according to their labels which are just the index of their bundles. Since the messages colored $(1,2)$ have label 1, it then follows that all the messages received processor Q during the $(i,1)$ process were colored $(1,1)$. But that was d time units ago, so the buffer is empty now since $l \leq d$. The same argument can be applied to all the messages colored $(i,2)$. This concludes the proof of the theorem, since the proof of the cases when $j>2$ is similar.

□

We now establish a lower bound for the total communication time required by any algorithm. The proof of the lower bound for the case when $l=1$ is given in [2].

Let us now establish a lower bound for the case when $l=d$. Consider Example 3.2 given below which is a restricted version of Example 3.1.

Example 3.2 *For any integer $d>1$, we define problem instance I_d as follows. The number of processors is $n=d^d+d$. Processors $P_1,P_2,\ldots,P_d$ just send messages and the remaining processors P_J for $J=(j_1,j_2,\ldots j_d)$, where each $j_l \in [1:d]$, i.e., each j_l is an integer whose value is between 1 and d. Each processor P_J for $J=(j_1,j_2,\ldots j_d)$, receives a message from the j_lth bundle of processor P_l.*

For the problem instance given in Example 3.2 we know that processor P_1 must send the messages emanating out of one of its bundles for the first time at time d or later simply because there are d bundles emanating out of P_1 and no two messages belonging to different bundles may be sent concurrently. The same holds $P_2, P_3, \ldots, P_d$. Lets say that these were bundles $(j_1, j_2, \ldots, j_d)$. Let $J = (j_1, j_2, \ldots, j_d)$. Now processor P_J receives a message from all of these bundles at time d or later. To empty its buffer it requires at least $d-1$ time units. Therefore, the total communication time of every solution is at least $2d-1$.

Let us now consider the case when $l = d/2$. Given any two bundles emanating from different processors it is not possible to send all their messages during the same time unit. Therefore in a schedule with total communication time at most $2d-1$ can have at most $2d-1$ bundles sending their messages just during 1 time period. So we know that there is a processor for which all the messages emanating from at most one bundle are sent during one period of time and at least $d-1$ bundles need to be sent each during two or more time periods. Therefore the total communication time is at least $1 + 2(d-1) = 2d-1$. Our schedules have total communication time equal to $2d + d/2 - 1$, which is about 25% from the lower bound.

We conjecture that similar lower bounds can be establish for all values of l. For brevity we cannot include our preliminary results here.

4 Discussion

We have shown the buffers, a relatively inexpensive ways to speed-up communication, can be used to generate solutions that require considerable smaller total communication time than that required for the MM_C problems. The solutions are similar to the ones obtained for the MMF_C problems. However they can be generated much faster than for the MMF_C problem. Furthermore, the solutions presented in this paper use the fewest number of communication links since messages are sent directly, rather than indirectly though several links. The most important open problem is to determine whether or not buffers can be used to reduce the total communication time for the MMF_C problem to obtain communication schedules with at most $3d/2$ communication rounds.

We have established tight lower bounds when $l = 1$ and $l = d$, as well as almost tight bounds when $l = d/2$. We conjecture that our solutions are almost tight for all values of l.

References

[1] Gonzalez, T. F., "MultiMessage Multicasting," Proceedings The Irregular'96 Workshop, LNCS (1117), Springer, (1996), pp. 217 – 228.

[2] Gonzalez, T. F., "Complexity and Approximations for Multimessage Multicasting," *J. of Parallel and Distributed Computing*, 55(2), (1998), 215 – 235.

[3] Shen, H, "Efficient Multiple Multicasting in Hypercubes," *J. of Systems Architecture*, 43(9), (1997).

[4] Gonzalez, T. F., and S. Sahni, "Open Shop Scheduling to Minimize Finish Time," *Journal of the ACM*, 23(4), (1976), pp. 665 – 679.

[5] Cole, R., K. Ost and S. Schirra, "Edge-Coloring Bipartite Multigraphs in $O(E \log D)$," *Combinatorica*, 21 (2001), pp. 5 – 12.

[6] Gonzalez, T. F., "Open Shop Scheduling," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Ed. Y. J.-T. Leung, Chapter 6, Chapman & Hall / CRC, 2004.

[7] Coffman, Jr. E. J., M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling File Transfers in Distributed Networks," *SIAM J. on Computing*, 14(3) (1985), pp. 744 – 780.

[8] Whitehead, J. "The Complexity of File Transfer Scheduling with Forwarding," *SIAM J. on Computing*, 19(2), (1990), pp. 222 – 245.

[9] Choi, H. A., and S. L. Hakimi, "Data Transfers in Networks," *Algorithmica*, 3, (1988), pp. 223 – 245.

[10] Hajek, B., and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Transactions on Information Theory*, 34(5), (1988), pp. 910 – 917.

[11] Gopal, I. S., G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong, "An Optimal Switching Algorithm for Multibeam Satellite Systems with Variable Band Width Beams," *IEEE Transactions on Communications*, 30(11), (1982) pp. 2475 – 2481.

[12] Rivera-Vega, P. I., R. Varadarajan, and S. B. Navathe, "Scheduling File Transfers in Fully Connected Networks," *Networks*, 22, (1992), pp. 563 – 588.

[13] Gonzalez, T. F., "Improved Approximation Algorithms for Multimessage Multicasting," *Nordic Journal on Computing*, 5, (1998), 196 – 213.

[14] Gonzalez, T. F., "Distributed Multimessage Multicasting," *Journal of Interconnection Networks*, 1(4), (2000), pp 303 – 315.

[15] Gonzalez, T. F., "Simple Multimessage Multicasting Approximation Algorithms With Forwarding," *Algorithmica*, 29, (2001), pp. 511 – 533.

[16] Gonzalez, T. F., "On Solving Multimessage Multicasting Problems," *International Journal of Foundations of Computer Science*, 12(6), (2001), pp. 791 – 808.

[17] Thaker, D. and Rouskas, G., "Multi-Destination Communication in Broadcast WDM Networks: A Survey," *Optical Networks*, 3(1), (2002), 34-44.

[18] Valiant, L. G., "General Purpose Parallel Architectures," *in Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., Elsevier, New York, 1990.