Pipelining Broadcasts on Heterogeneous Platforms

O. Beaumont
LaBRI, UMR CNRS 5800, Bordeaux, France
Olivier.Beaumont@labri.fr

A. Legrand, L. Marchal and Y. Robert
LIP, UMR CNRS-INRIA 5668, ENS Lyon, France
{Arnaud.Legrand,Loris.Marchal,Yves.Robert}@ens-lyon.fr

Abstract

In this paper, we consider the communications involved by the execution of a complex application, deployed on a heterogeneous platform. Such applications extensively use macro-communication schemes, for example to broadcast data items. Rather than aiming at minimizing the execution time of a single broadcast, we focus on the steady-state operation. We assume that there is a large number of messages to be broadcast in pipeline fashion, or a large message that can be split into several packets, and we aim at maximizing the throughput, i.e. the (rational) number of messages which can be broadcast every time-step. Achieving the best throughput may well require that the target platform is used in totality: we show that neither spanning trees nor DAGs are as powerful as general graphs. We show how to compute the best throughput using linear programming, and how to exhibit a periodic schedule, first when restricting to a DAG, and then when using a general graph. The polynomial compactness of the description comes from the decomposition of the schedule into several broadcast trees that are used concurrently to reach the best throughput. The concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

1. Introduction

Broadcasting in computer networks is the focus of a vast literature. The one-to-all broadcast, or single-node broadcast [5], is the most primary collective communication pattern: initially, only the source processor has the data that needs to be broadcast; at the end, there is a copy of the original data residing at each processor.

Parallel algorithms often require to send identical data to all other processors, in order to disseminate global information (typically, input data such as the problem size or application parameters). Numerous broadcast algorithms have been designed for parallel machines such as meshes, hypercubes, and variants. The *MPI_Bcast* routine [8] is widely used, and particular care has been given to its efficient implementation on a large variety of platforms. There are three main variants considered in the literature:

Atomic broadcast: the source message is atomic, i.e. cannot be split into packets. A single message is sent by the source processor, and forwarded across the network.

Pipelined broadcast: the source message can be split into an arbitrary number of packets, which may be routed in a pipelined fashion, possibly using different paths.

Series of broadcasts: the same source processor sends a series of atomic one-to-all broadcasts. The processing of these broadcasts can be pipelined (and different paths can be used in the graph).

For the first two problems, the goal is to minimize the total execution time (or makespan). For the third problem, the objective function is rather to optimize the throughput of the steady-state operation, i.e. the average amount of data broadcast per time-unit.

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)
IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

Because there is a single message involved, the *atomic broadcast* is implemented using a spanning tree. In the case of the *pipelined broadcast*, things get more complex: the idea is to use several edge-disjoint spanning trees to route simultaneously several distinct fractions of the total message. Along each spanning tree, the message fraction is divided into packets, which are sent in a pipeline fashion, so as to minimize start-up idle times. See [9] for an illustration with two-dimensional meshes.

The *series of broadcasts* problems has been considered by Moore and Quinn [6], and by Desprez et al. [4], but with a different perspective: they consider that *distinct* processor sources successively broadcast one message, and their goal is to load-balance this series of communications. Here, we assume that the *same* source processor initiates all the broadcasts: this is closer to a master-slave paradigm where the master disseminates the information to the slaves in a pipelined fashion, e.g. the data needed to solve a collection of (independent) problem instances. The *series of broadcasts* resembles the *pipelined broadcast* problem in that we can solve the latter using an algorithm for the former: this amounts to fix the granularity, i.e. the size of the atomic messages (packets) that will be sent in pipeline. However, an efficient solution to the *pipelined broadcast* problem would require to determine the size of the packets as a function of the total message length.

In this paper, we re-visit the *series of broadcasts* and the *pipelined broadcast* problems in the context of heterogeneous computing platforms. Several authors have recently studied broadcasting with processors communicating with their neighbors along links with different capacities, and/or different start-up costs, but they mainly restricted to the *atomic broadcast* problem. Our major result is the design of asymptotically optimal algorithms for the *series of broadcasts* problem, and for the *pipelined broadcast* problem. Both algorithms rely on tools such as linear programming, network flows and graph theory (Edmond's branching theorem).

Due to lack of space, we do not survey related work: instead, we refer to the extended version [1] of this paper.

2. Framework

The target architectural platform is represented by an edge-weighted directed graph $G = (V, E, c)$, as illustrated in Figure 1. Note that this graph may well include cycles and multiple paths. Let $p = |V|$ be the number of nodes. There is a *source* node P_s, which plays a particular role: it initially holds all the data to be broadcast. All the other nodes P_i, $1 \leqslant i \leqslant p, i \neq s$, are destination nodes which must receive all the data sent by P_s.

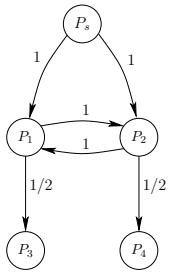


Figure 1. Simple network topology. The value of $c_{j,k}$ is indicated along each edge. The node P_s is the source of the broadcasts.

There are several scenarios for the operation of the processors. In this paper, we concentrate on the *one-port model*, where a processor node can simultaneously receive data from one of its neighbor, and send (independent) data to one of its neighbor. At any given time-step, there are at most two communications involving a given processor, one in emission and the other in reception. Each edge $e_{j,k} : P_j \to P_k$ is labeled by a value $c_{j,k}$ which represents the time needed to communicate one unit-size message from P_j to P_k (start-up costs are dealt with below, for the *pipelined broadcast* problem). The graph is directed, and the time to communicate in the reverse direction, from P_k to P_j, provided that this link exists, is $c_{k,j}$. Note that if there is no communication link between P_j and P_k we let $c_{j,k} = +\infty$, so that $c_{j,k} < +\infty$ means that P_j and P_k are neighbors in the communication graph. We state the communication model more precisely: if P_j sends a unit-size message to P_k at time-step t, then: (i) P_k cannot initiate another receive operation before time-step $t + c_{j,k}$ (but it can perform a send operation); and (ii) P_j cannot initiate another send operation before time-step $t + c_{j,k}$ (but it can perform a receive operation).

Series of broadcasts In the *series of broadcasts* problem, the source processor broadcasts a (potentially infinite) sequence of unit-size messages. Start-

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

up costs are included in the values of the link capacities $c_{j,k}$. The optimization problem, which we denote as SERIES(V, E, c), is to maximize the throughput, i.e. the average number of broadcasts initiated per time-unit. We work out a little example in Section 3, using the platform represented in Figure 1.

Pipelined broadcast In the *pipelined broadcast* problem, the source processor broadcasts a large message of total size L. The message can be split into an arbitrary number of packets. The time to send a packet of size $n_{j,k}$ from P_j to P_k is $\beta_{j,k} + n_{j,k}c_{j,k}$. We include the start-up costs in the definition of the platform graph, which becomes $G = (V, E, c, \beta)$. The optimization problem, which we denote as PIPE(V, E, c, β), is to minimize the makespan, i.e. to find the number and size of the packets, and a routing scheme for each broadcast packet, so that the total execution time is as small as possible.

3. Comparing topologies for series of broadcasts

In this section, we work out a small example, whose objective is to show the difficulty of the problem. We compare the best throughput that can be achieved using a tree, a directed acyclic graph (DAG), or the full topology with cycles.

3.1. Optimal solution

Consider the simple example of the network described on Figure 1. The best throughput that can be achieved on this network is 1, i.e. one message is broadcast every time-step after some initialization phase. On one hand, since the source cannot send more than one message at each time-unit, the best throughput is less than or equal to 1. On the other hand, a feasible schedule for a series of broadcasts realizing this throughput is given in Figure 2, where messages are tagged by their number, and columns represent time-steps. The schedule is periodic, with period length $T = 2$, and steady-state is reached at time-step $t = 5$: a new broadcast is then initiated by the source processor every time-step, so that the throughput of the schedule is equal to 1.

Here are a few comments to read Figure 2. At time-step $t = 1$, the source processor P_s sends the first message m_1 to P_1. At time-step $t = 2$, the source processor P_s sends the second message m_2 to

Period	1		2		3		4	
Time / Link	1	2	3	4	5	6	7	8
$P_s \to P_1$	m_1		m_3		m_5		m_7	
$P_s \to P_2$		m_2		m_4		m_6		m_8
$P_1 \to P_2$		m_1		m_3		m_5		m_7
$P_1 \to P_3$	$\times$		$\times$		m_1, m_2		m_3, m_4	
$P_2 \to P_1$	$\times$		m_2		m_4		m_6	
$P_2 \to P_4$		$\times$		m_1, m_2		m_3, m_4		m_5, m_6

Figure 2. An optimal schedule for the network of Figure 1.

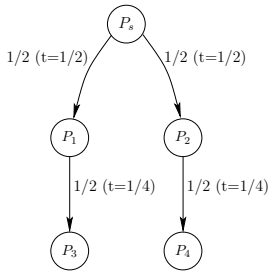


Figure 3. Broadcasting a message from P_s, using the first broadcast tree. Edges are labeled with the (average) number of messages that circulate within one time-unit. The time needed to transfer these messages is indicated between brackets.

P_2. Every odd-numbered step, P_s sends a new message to P_1, and every even-numbered step, P_s sends a new message to P_2. P_1 is idle at time-steps $t = 1$ and $t = 3$: since it has not yet reached its steady-state, we have indicated fictitious messages (represented as crosses "$\times$"), which it would have received from P_s if the computation had started earlier. At time-step $t = 2$, P_1 forwards the first message m_1 to P_2. Every even-numbered time-step, P_1 forwards to P_2 the message that it has received from P_s during the previous step. At step $t = 5$, P_1 forwards two-messages to P_3: message m_1 that it received from P_s at $t = 1$, and message m_2 that it received from P_2 at $t = 3$. Because the link is twice faster ($c_{1,3} = 1/2$), one time-step is enough for sending both messages. From then on, every odd-numbered time-step, P_1 sends two messages to P_3. P_2 operates in a similar fashion, alternately sending one message to P_1 and two messages to P_4.

We further use the example to illustrate the "superiority" of general graphs over DAGs, and of DAGs over spanning trees, for the SERIES problem.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

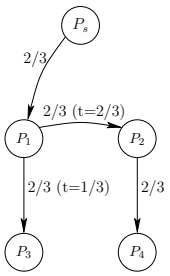


Figure 4. Broadcasting a message from P_s, using the second broadcast tree. Edges are labeled with the (average) number of messages that circulate within one time-unit. The time needed to transfer these messages is indicated between brackets.

3.2. Broadcast trees

As already pointed out, the atomic broadcast is frequently implemented using a spanning tree. This raises a natural question: what is the best throughput that can be achieved for the SERIES problem, using a single spanning tree to broadcast all the messages?

A broadcast tree $T = (V, E_T)$ is a subgraph of G, which is a spanning tree rooted at P_s, source of the broadcast. The broadcast tree can be used to broadcast r messages within a time-unit (in steady state) if the one-port constraints are satisfied:

$$\forall i \in V \qquad \sum_{j \in V, (P_i, P_j) \in E_T} r \times c_{i,j} \leqslant 1 \qquad (1)$$

These are the constraints for outgoing messages: Equation 1 simply states that each node i needs the time to send the message to all of its children in the broadcast tree. As a node receives its messages from only one node (its parent in the tree), the constraint on incoming messages writes $r \times c_{f(i),i} \leqslant 1$, where $f(i)$ is the parent of i in T. This constraint is satisfied for i as soon as Equation 1 is verified for $f(i)$, so we can discard this constraint. In the following, we let $\mathrm{TP}(T)$ denote the throughput of a broadcast tree T, i.e. the maximum number of messages of unit size which can be broadcast using T in one time-unit.

What is the maximal throughput $\mathrm{TP}(T)$ that can be achieved using a sub-tree of the platform described on Figure 1? We can build two kinds of spanning trees: either both P_1 and P_2 are children of the source, or only one of them is a child of the source in the tree. In the first case, where P_1 and P_2 are directly linked to the source, we obtain the broadcast tree of Figure 3. The labels on the figure are not communication capacities. Instead, they represent the (average) number of messages that circulate along each edge within one time-unit. The value 1/2 means that one message is sent every two steps along the edge. Obviously, because of the one-port constraint for the source processor, this is the best throughput that can be achieved using this tree.

In the second case, one of the vertices P_1 and P_2 is not directly linked to the source. Without loss of generality, we assume that the edge (P_s, P_2) does not belong to the tree. This leads to the spanning tree of Figure 4, whose optimal throughput is $\mathrm{TP}(T) = 2/3$. Indeed, on one hand, the one-port constraint for processor P_1 states that P_1 needs 1.5 time-steps to transfer a message to its children P_2 and P_3, so we cannot achieve more than 2 broadcasts every 3 time-steps. We can indeed achieve this throughput $\mathrm{TP}(T) = 2/3$, as illustrated in the figure. Overall, this is the best throughput that can be obtained with a broadcast tree in this network.

3.3. Broadcast DAGs

We choose a less restrictive assumption and try to extract a Directed Acyclic Graph (DAG), instead of a broadcast tree, out of the network. Of course we look for a DAG with a single entry vertex, namely the source processor. Can we can get a better throughput than with a tree? The answer is positive. There are only two candidates DAGs which do not reduce to spanning trees: the DAG shown on Figure 5, and its symmetric counterpart where the edge (P_1, P_2) is replaced by the edge (P_2, P_1). Without loss of generality, we restrict to the DAG of Figure 5. Because the first broadcast tree of Figure 3 is a subgraph of the DAG, we can achieve a throughput at least 1/2. But we can get more. Figure 5 illustrates how to initiate 4 broadcasts every 5 time-steps, hence a throughput 4/5. It turns out that this is the optimal solution with this DAG: we explain in Section 4 how to compute the best throughput for a DAG.

As a conclusion, we point out that the best throughput achieved for the SERIES problem strongly depends upon the graph structure allowed for transferring the messages. As the little example shows, restricting to trees is less powerful than

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

using DAGs (throughput of $\frac{2}{3}$ instead of $\frac{4}{5}$), and restricting to DAGs is less powerful than using the full network graph (throughput of $\frac{4}{5}$ instead of 1).

It turns out that computing the optimal throughput for the SERIES problem is much easier when restricting to DAGs than when dealing with arbitrary graphs (including cycles). Therefore, we give the solution for DAGs in Section 4, to prepare for the difficult algorithm for general graphs (Section 5).

4. Series of broadcasts on a DAG

In this section, we assume the network is organized as a DAG rooted at the source P_s, and that all nodes are reachable from the source. Under this hypothesis, we provide an algorithm to compute the optimal solution to the SERIES(V, E, c) optimization problem.

We let $n_{j,k}$ denote the (fractional) number of unit-size messages sent from processor P_j to processor P_k during one time-unit, and $t_{j,k}$ denote the fraction of time spent by processor P_j to send messages to P_k during one time-unit. As above, $c_{j,k}$ is the time needed to perform the transfer of a unit-size message on edge (P_j, P_k). The activity on edge (P_j, P_k) in one time-unit is bounded:

$$\forall P_j, \forall P_k \qquad 0 \leqslant t_{j,k} = n_{j,k} \times c_{j,k} \leqslant 1 \qquad (2)$$

The one-port model constraints are expressed by the following equations:

$$\forall P_j, \quad \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant 1 \text{ (outgoing msg)} \quad (3)$$

$$\forall P_j, \quad \sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant 1 \text{ (incoming msg)} \quad (4)$$

Moreover, each node should receive the same (fractional) number of messages in one time-unit (that is the throughput TP):

$$\forall P_j \text{ with } j \neq s, \quad \sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP} \quad (5)$$

We summarize these equations in a linear program (with rational coefficients and unknowns):

STEADY-STATE SERIES OF BROADCASTS PROBLEM ON A DAG (SSBDAG(G))
Maximize TP,
subject to

$\forall P_j, \forall P_k$	$0 \leqslant t_{j,k} = n_{j,k} \times c_{j,k} \leqslant 1$
$\forall P_j,$	$\sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant 1$
$\forall P_j,$	$\sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant 1$
$\forall P_j$ with $j \neq s,$	$\sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP}$

Theorem 1. *The solution of the* SSBDAG(G) *linear program provides the optimal solution to the* SERIES *problem on a DAG: the value TP returned by the program is the maximum number of broadcasts that can be initiated per time-unit. Furthermore, it is possible to construct the corresponding optimal periodic schedule in time polynomial in size of the input DAG.*

Proof. We only give the main ideas of the proof here: a detailed proof can be found in [1]. Intuitively, the previous linear program gives a bound on the achievable throughput. To prove that this bound can indeed be achieved, after solving the linear program in rational numbers, we compute the least common multiple T of all denominators that appear in the value of the variables, then we multiply every quantities by T. We get integer results for a steady-state operation with period T. There remains to show that we can derive a schedule which (i) achieves the throughput returned by the linear program; (ii) does not violate any constraint in the model; and (iii) admits a *compact* description, i.e. of size polynomial in the input data.

For (i), we have to ensure that fictitious cycles (think of a parasitic cycle involving two processors exchanging meaningless data) do not account for a fake throughput in the solution of the linear program SSBDAG(G). Because the platform is a DAG, this cannot happen, and the actual schedule will be constructed iteratively, the nodes being topologically sorted according to their shortest distance from the source. However, this forbids the use of SSBDAG(G) for general platforms.

For (ii), the question is the following: given a set of processors operating under the one-port model, can we actually execute any set of communications within a prescribed time-bound T? Of course, a necessary constraint is that Equations 3 and 4 are satisfied by each processor during the time interval: $\forall P_j, \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant T$ (outgoing messages) and $\sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant T$ (incoming messages). However, it is not obvious that these necessary conditions are sufficient to build a schedule, because only independent communications (with disjoint sender and receiver pairs) can be scheduled simultaneously.

For (iii), because T is the least common multiple of values of the linear program solution, $\log(T)$ has a size that is polynomial in the input data size but not necessarily T itself, so a time-step by time-step description of the schedule might be too large.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

We solve both the (ii) and (iii) problems as follows: we transform the platform graph into a weighted bipartite graph by splitting each node P_j into an outgoing node P_j^{send} and an incoming node P_j^{recv}. Each edge from P_j^{send} to P_k^{recv} is weighted by the length of the communication $t_{j,k}$. At any given time-step, we can schedule at most two communications involving a given processor, one in emission and the other in reception. Thus, at a given time step, only communications corresponding to a matching in the bipartite graph can be performed simultaneously. Therefore, we need to decompose the weighted bipartite graph into a sum of matchings. The desired decomposition of the graph is in fact an edge coloring. The weighted edge coloring algorithm of [7, vol.A chapter 20] provides in time $O(|E|^2)$ a number of matchings which is polynomial in the size of the platform graph (in fact there are at most $|E|$ matchings). Moreover, the overall weight of the matchings is equal to the maximum weighted degree of any P_j^{send} or P_j^{recv} node, so that we can use these matchings to perform the different communications. ■

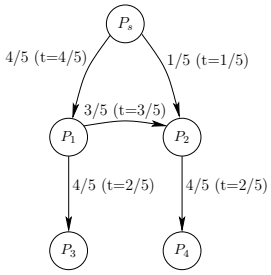


Figure 5. Broadcast on a DAG. 4 broadcasts are initiated every 5 time-steps.

We come back to the example given in Figure 5, for which we claimed to obtain a throughput of 4/5: this is in fact the value returned by the linear program on this example. The values $5 \times n_{j,k}$ ($q = 5$) are given along the edges in Figure 5. The schedule constructed in the proof [1] is represented on Figure 6(a) (the number of the messages sent is mentioned between brackets, $[0, 3]$ stand for messages number 0 to 3). Once pipelined, it gives the communications represented on Figure 6(b). The last step is to use the edge-coloring algorithm to create a schedule where several receptions or emissions never overlap on a node, which leads to the final schedule of Figure 6(c).

Link \ Period	0	1	2
$P_s \to P_1$	[0,3]		
$P_s \to P_2$		[0]	
$P_1 \to P_2$		[1,3]	
$P_1 \to P_3$			[0,3]
$P_2 \to P_4$			[0,3]

(a) Basic schedule

Link \ Period	0	1	2 3	4	5	6
$P_s \to P_1$	[0,3]	[4,7]	[8,11]	[12,15]	[16,19]	[20,23]
$P_s \to P_2$		[0]	[4]	[8]	[12]	[16]
$P_1 \to P_2$		[1,3]	[5,7]	[9,11]	[13,15]	[17,19]
$P_1 \to P_3$			[0,3]	[4,7]	[8,11]	[12,15]
$P_2 \to P_4$			[0,3]	[4,7]	[8,11]	[12,15]

(b) Pipelined communications

Period	1					2					3					4				
Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Link																				
$P_s \to P_1$	0	1	2	3		4	5	6	7		8	9	10	11		12	13	14	15	
$P_s \to P_2$					×					0					4					8
$P_1 \to P_2$	×	×	×			1	2	3			5	6	7			9	10	11		
$P_1 \to P_3$				×	×				0,1	2,3				4,5	6,7				8,9	10,11
$P_2 \to P_4$	×	×				×	×				0,1	2,3				4,5	6,7			

(c) Final schedule

Figure 6. Solution for the example of a broadcast on a DAG

5. Series of broadcasts on a general platform

The SERIES problem turns out to be much more difficult for general graphs than for DAGs. As already mentioned, the linear program SSBDAG(G) would lead to wrong solutions, because fictitious cycles could corrupt the throughput. We still solve the problem by using a linear programming approach, but with a completely different, more involved, formulation.

5.1. Linear program and upper bound for the throughput

We proceed in three main steps:

1. First, we express the linear constraints that have to be satisfied by an optimal solution in steady-state mode. The linear program gives an upper bound of the throughput that can be achieved on the platform.

2. Then, from an optimal solution of the linear program, we build a set of broadcast trees. The weighted sum of these trees leads to the optimal throughput, and satisfy all the communications constraints.

3. Finally, we provide a compact description of the optimal periodic schedule. We ensure that

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

the size of the description is polynomial in the size of the initial data.

We focus on the broadcast of a unit-size message from a processor P_{source} in a platform represented by the graph $G = (V, E, c)$. We want to realize a steady-state broadcast with the best throughput in one time-unit. We introduce $x_i^{j,k}$, the fractional number of messages whose final destination is P_i and which transit through edge (P_j, P_k) in one time-unit. We can already write the linear constraints concerning the number of messages broadcast by the source and received by every other node in one time-unit:

$$\forall P_i \in V, \sum_{P_j,(source,j)\in E} x_i^{source,j} = \text{TP} \quad (6)$$

$$\forall P_i \in V, \sum_{P_j,(j,i)\in E} x_i^{j,i} = \text{TP} \quad (7)$$

We also write a constraint corresponding to the conservation law for messages in P_j targeting P_i (with $P_j \neq P_i$):

$$\forall P_i, P_j, \sum_{P_k,(k,j)\in E} x_i^{k,j} = \sum_{P_k,(j,k)\in E} x_i^{j,k} \quad (8)$$

We denote as $n_{j,k}$ the total (fractional) number of messages going through the edge (P_j, P_k) in one time-unit. The number of such messages targeting each P_i is $x_i^{j,k}$, but these messages can be either different or identical. If they were all different, then we would have $n_{j,k} = \sum_i x_i^{j,k}$, which corresponds to a scatter operation. However, since the source broadcasts the same messages to each node, some of the $x_i^{j,k}$ may be identical. Using $\sum_i x_i^{j,k}$ to define $n_{j,k}$ is too pessimistic for the broadcast operation. To get an upper bound on the achievable throughput, we assume that all messages going through one edge and targeting different nodes are part of a same set of messages:

$$\forall (j,k) \in E, \quad n_{j,k} = \max_i x_i^{j,k} \quad (9)$$

As we know the number of messages going through each edge, we can write the constraints about the one-port model:

$$\forall P_j, \sum_{P_k,(j,k)\in E} c(j,k) \times n_{j,k} \leqslant 1 \quad (10)$$

$$\forall P_j, \sum_{P_k,(k,j)\in E} c(k,j) \times n_{k,j} \leqslant 1 \quad (11)$$

All these constraints are summarized in the following linear program:

Steady-State Broadcast
Problem on a Graph (SSB(G))
Maximize TP,
subject to
$\forall P_i \in V, \sum_{P_j,(source,j)\in E} x_i^{source,j} = \text{TP}$
$\forall P_i \in V, \sum_{P_j,(j,i)\in E} x_i^{j,i} = \text{TP}$
$\forall P_i, P_j, i \neq j, \sum_{P_k,(k,j)\in E} x_i^{k,j} \sum_{P_k,(j,k)\in E} x_i^{j,k}$
$\forall (j,k) \in E, n_{j,k} = \max_i x_i^{j,k}$
$\forall P_j, \sum_{P_k,(j,k)\in E} c(j,k) \times n_{j,k} \leqslant 1$
$\forall P_j, \sum_{P_k,(k,j)\in E} c(k,j) \times n_{k,j} \leqslant 1$

Clearly, the linear program SSB(G) provides an upper bound on the throughput, but there remains to prove that the all set of communications can be orchestrated so that whenever two messages targeted to different processors circulate along a given edge, they are identical. The (somewhat unexpected) result is that the bound can be achieved.

As previously, after solving the linear program in rational numbers, we compute the least common multiple T of all denominators that appear in the value of the variables, then we multiply every quantities by T. We get integer results for a steady-state operation with period T.

5.2. Extracting trees

Once the linear program solved, we have an upper bound on the throughput of a broadcast on the platform. However, we do not know if the bound is achievable, as the assumption we made on the messages going through one edge being part of the same set of messages may be too strong. We use Edmond's Branching theorem to prove that there exists a set of broadcast trees such that the sum of these trees satisfies the linear constraints and reaches the upper bound of the throughput. Edmond's theorem [10] links the number of edges that have to be deleted to make a vertex P_j unreachable from the source (denoted as $\kappa(G, P_{source})$) and the number of edge-disjoint spanning trees rooted in P_0:

Theorem 2. *The number of edge-disjoint spanning trees rooted in P_0 is equal to $\kappa(G, P_{source})$.*

In [1], we prove that $\kappa(G, P_{source}) = T \times \text{TP}$, which is the number of messages broadcast in one period T. This provides a decomposition of the solution in $T \times$ TP broadcast trees, and guarantees that the use of these trees simultaneously in one

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

period T satisfies the constraints of the communication model, as the sum of these trees is a subgraph of the graph solution of the linear program. However, the description of $T \times \mathrm{TP}$ broadcast trees is not compact enough, since the number of trees can be exponential in the size of the initial data. Indeed, TP is obtained from the solution of the linear program built with the initial data, so it can be encoded in polynomial size. Similarly, T is the least common multiple of values of the linear program solution, so it also can be encoded in polynomial size. Nevertheless, to encode all the trees we need at least $|V| \times T \times \mathrm{TP}$, which is not polynomial in the size of the data. Fortunately there exists a weighted version of Edmond's theorem [7, vol.B chapter 53]:

Theorem 3. *Let $G = (V, E, T \times n_{j,k})$ be an oriented graph with weighted edges. There exist k_T trees and k_T integer weights such that: $\forall j,k, \quad \sum_l \lambda_l \chi^T_{j,k}(T_l) \leqslant T n_{j,k}$, where $\chi^T_{j,k}(T_l) = 1$ if $(P_j, P_k) \in T_l$ and 0 otherwise, and $\sum \lambda_l$ maximal. Besides, these trees can be found in polynomial time, and $k_T \leqslant |V|^3 + |E|$*

We can prove that $\sum_l \lambda_l = \kappa(G, P_0) = T$. Then we obtain a decomposition of the solution into a set of weighted trees that can be encoded in polynomial size. As previously, we partition the set of communications into several sets without communication conflicts for the one-port model, using the weighted matching algorithm. We obtain an optimal periodic schedule for the steady-state operation. All these steps are detailed in [1].

5.3. Asymptotic optimality

In this section, we state that the previous periodic schedule is asymptotically optimal: basically, no scheduling algorithm (even non periodic) can execute more broadcast operations in a given timeframe than ours, up to a constant number of operations. This section is devoted to the formal statement of this result, whose proof can be found in [1].

Given a platform graph $G = (V, E, c)$, a source processor P_s holding an infinite number of unit-size messages, and a time bound K, define $opt(G, K)$ as the optimal number of messages that can be received by every target processor in a succession of broadcast operations from P_s, within K time-units. Let $\mathrm{TP}(G)$ be the solution of the linear program $\mathrm{SSB}(G)$ of Section 5.1 applied to this platform graph G. We have the following result:

Lemma 1. $opt(G, K) \leqslant \frac{1}{\mathrm{TP}(G)} \times K$

This lemma states that no schedule can send more messages that the steady-state. There remains to derive an actual schedule based on the periodic solution, where the loss due to the initialization and clean-up phases is bounded. This algorithm is fully described in [1]. Let $steady(G, K)$ denotes the number of messages broadcast by the source in K time-units with this algorithm:

Theorem 4. *The scheduling algorithm based on the steady-state operation is asymptotically optimal:*

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1.$$

6. Pipelined broadcast

In the *pipelined broadcast* problem, the source processor broadcasts a single (large) message of total size L, which can be split into an arbitrary number of packets. To be realistic, the model must include start-up overheads in the communication times: otherwise, with a cost linear in the packet size, the best solution would be to have an infinite number of infinitely small packets. Therefore, in this section we assume that the time to send a packet of size $n_{j,k}$ from P_j to P_k is $\beta_{j,k} + n_{j,k} c_{j,k}$. We include the start-up costs in the definition of the platform graph, which becomes $G = (V, E, c, \beta)$. The $\textsc{Pipe}(V, E, c, \beta)$ problem is to minimize the time needed to broadcast the initial message of size L, i.e. to find the number and size of the packets, and a routing scheme for each packet, so that the total execution time is as small as possible.

Using again the periodic scheme mentioned in Section 5, we can extend Theorem 4 and prove a result of asymptotic optimality for the Pipe optimization problem. Due to lack of space we refer the reader to the extended version [1] for a formal statement. This result is inspired by the work of Bertsimas and Gamarnik [2], who use a fluid relaxation technique to prove the asymptotic optimality of a simpler packet routing problem.

7. A complete example

In this section, we work out a complete example. The platform is generated by Tiers, a random generator of topology [3]. The bandwidth of the links are

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

randomly chosen, and the topology is represented on Figure 8(a). Figure 8(b) shows the results of the linear program SSB(G). The edges of this graph represent communications, and their label is a list of transfers: if edge (i, j) has the item $y(k)$ in its list, it means that $Nx_k^{i,j} = y$, i.e. that in the steady-state integer solution, y messages go through edge (i, j) to reach P_k. Here, the throughput achieved is 2 messages per period of 152 time-units.

From these communications, we extract two broadcast trees, which are represented on Figure 9, where both the logical tree and the communications extracted from Figure 8(b) are mentioned. We point out that not all communications arising from the linear program SSB(G) are actually used in the trees: some are redundant (hence useless). For example, there is a cycle between node P_1 and P_8 for transfers, whose targets are nodes P_3, P_5, P_6 and P_7. These communications do not improve the throughput of the broadcast, but they do not interfere with other communications: indeed, the maximum of all communications on these edges is $Nx^{1,8} = Nx^{8,1} = 1$. Extracting trees from the solution of the linear program enables us to neglect such "parasitic" communications.

8. Conclusion

In this paper, we have studied several broadcasting problems on heterogeneous platforms. Our major objective was to maximize the throughput that can be achieved in steady-state mode, when a large number of same-size broadcasts are done in a pipeline fashion, or when a single large message is split into packets that are broadcast in pipeline fashion too. Achieving the best throughput may well require that the target platform is used in totality: we have shown neither spanning trees nor DAGs are powerful enough. In passing, note that determining in a given graph the broadcast tree that achieves the best throughput among all trees is a NP-complete problem [1]. We have shown how to compute the best throughput using linear programming, and how to exhibit a periodic schedule, first when restricting to a DAG, and then when using a general graph. The polynomial compactness of the description comes from the decomposition of the schedule into several broadcast trees that are used concurrently to reach the best throughput.

An interesting problem would be to extend this work to the case of the multicast operation, where the target processors (the receivers) form a strict subset of the computing resources. In this case, even determining the best throughput in steady-state mode seems to be a challenging problem.

References

[1] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Optimizing the steady-state throughput of broadcasts on heterogeneous platforms heterogeneous platforms. Technical Report RR-2003-34, LIP, ENS Lyon, France, June 2003.

[2] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[3] K.L. Calvert, M.B. Doar, and E.W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[4] F. Desprez, S. Domas, and B. Tourancheau. Performance complexity of lu factorization with efficient pipelining and overlap on a multiprocessor. *Parallel Processing Letters*, 5(2):44–61, 1995.

[5] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 1994.

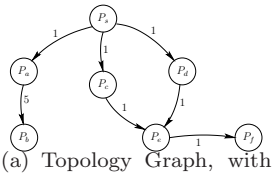
[6] J.A. Moore and M.J. Quinn. Generating an efficient broadcast sequence using reflected gray codes. *IEEE Trans. Parallel and Distributed Systems*, 8(11):1117–1122, 1997.

[7] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.

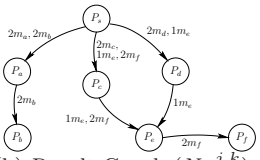
[8] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[9] J. Watts and R. Van De Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2):281–292, 1995.

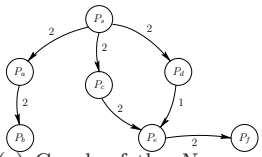
[10] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 1996.



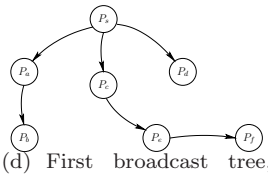
(a) Topology Graph, with the communication cost of each edge.



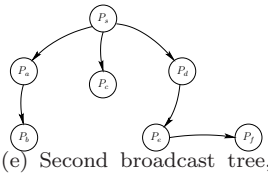
(b) Result Graph ($Nx_i^{j,k}$)



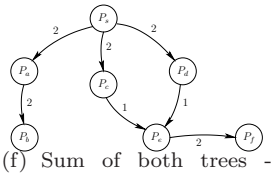
(c) Graph of the $Nn_{j,k} = \max_i (Nx_i^{j,k})$



(d) First broadcast tree, $\lambda_1 = 1$

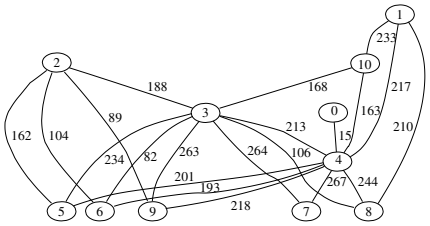


(e) Second broadcast tree, $\lambda_2 = 1$

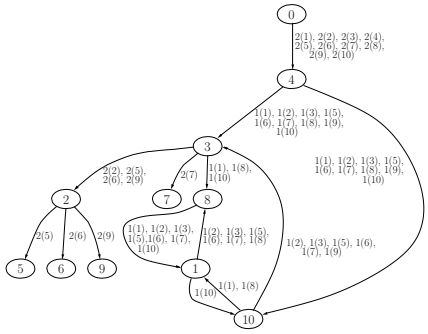


(f) Sum of both trees - graph of the $m_{j,k}$'s

Figure 7. Example where $m_{j,k} < Nn_{j,k}$. The optimal steady-state broadcast time T^* for one message is 5 time-units, due to edge (P_a, P_b). Figure 7(b) describes the results multiplied by the least common multiple $N = 2$, and Figure 7(c) reports the maximum values of $Nx_i^{j,k}$ on each edge. Figures 7(d) and 7(e) are the two broadcast trees extracted from the previous figure, each of them with a weight of $\lambda_l = 1$. Finally, Figure 7(f) represents the sum of these trees. On the edge (P_c, P_e), we have $m_{c,e} < Nn_{c,e}$: this edge is used by only one broadcast tree, so $m_{c,e} = 1$, whereas $Nn_{c,e} = 2$ because all messages targeting P_f are supposed to go through this edge in the optimal solution given by the linear solver, which is not the choice made when we use trees.

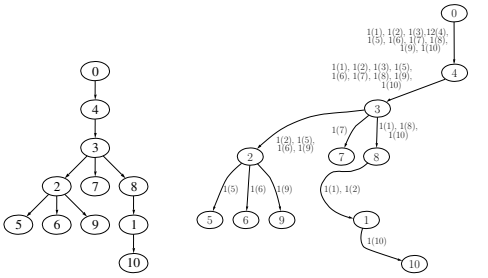


(a) Topology. Edge e is labeled by its bandwidth $bw(e)$. The cost of a transfer is $c(e) = 1000/bw(i)$ for a single message.



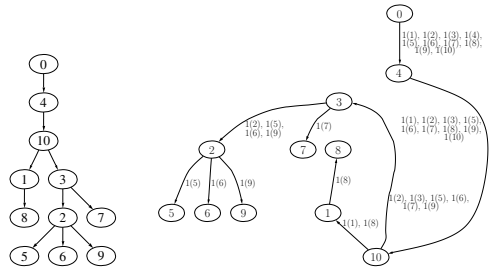
(b) Communication graph

Figure 8. Experiments on a given topology.



logical tree extracted communications

(a) First broadcast tree (broadcasting 1 message)



logical tree extracted communications

(b) Second broadcast tree (broadcasting 1 message)

Figure 9. Broadcast trees.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE