Pipelining Broadcasts on Heterogeneous Platforms

Olivier Beaumont, Arnaud Legrand, Loris Marchal, *Student Member*, *IEEE*, and Yves Robert, *Senior Member*, *IEEE*

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 macrocommunication 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, and we aim at maximizing the throughput, i.e., the (rational) number of messages which can be broadcast every time-step. We target heterogeneous platforms, modeled by a graph where resources have different communication and computation speeds. 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. It is important to point out that a concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

Index Terms—Scheduling, collective communications, NP-completeness, broadcast, heuristics, heterogeneous clusters, grids.

1 Introduction

Broadcasting in computer networks is the focus of a vast literature. The one-to-all broadcast, or single-node broadcast [1], 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 (see, among others, [2], [3], [4], [5], [6]). The *one-to-all* MPI routine [7] is widely used, and a particular case has been given to its efficient implementation on a large variety of platforms [8]. 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, involving messages of the same size. The processing of these broadcasts can be pipelined.

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.

In the case of the *atomic broadcast*, there is no reason why a processor (distinct from the source) would receive the message twice. Therefore, the atomic broadcast is frequently 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 fractions of the total message. Along each spanning tree, the message fraction is divided into packets, which are sent in a pipelined fashion, so as to minimize start-up idle times. See [3] for an illustration with two-dimensional meshes.

The *series of broadcasts* problems has been considered by Moore and Quinn [9], and by Desprez et al. [10], 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, for instance, 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*

• *O. Beaumont is with LaBRI, UMR CNRS 5800, Bordeaux, France. E-mail: Olivier.Beaumont@labri.fr.*
• *A. Legrand, L. Marchal, and Y. Robert are with LIP, UMR CNRS-INRIA 5668, ENS Lyon, France. E-mail: {Arnaud.Legrand, Loris.Marchal, Yves.Robert}@ens-lyon.fr.*

Manuscript received 11 Sept. 2003; revised 17 May 2004; accepted 19 Aug. 2004; published online 23 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0160-0903.

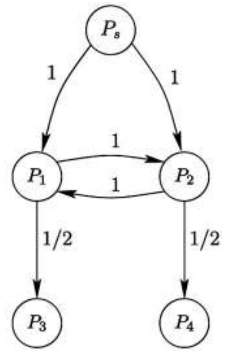

Fig. 1. Simple network topology. The value of $c_{j,k}$ is indicated along each edge. The node P_s is the source of the broadcasts.

broadcast problem would require to determine the size of the packets as a function of the total message length.

In this paper, we revisit the *series of broadcasts* problem (and very briefly the *pipelined broadcast* problem) 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 (see Section 8 on related work), but they mainly restricted to the *atomic broadcast* problem. Our approach focuses on the *steady-state* operation, and can be viewed as a fluid relaxation of the makespan minimization problem, which concentrates on the average quantities of messages sent through each links and forget about initialization and clean-up phases. Our algorithm, relying on tools such as linear programming, network flows, and graph theory, provides a periodic schedule, described in a compact form, which reaches the optimal throughput. Thanks to the periodicity of the schedule, it is possible to dynamically record the observed performance, and to inject this information into the algorithm to compute the optimal schedule for the next period. This makes it possible to react on the fly to resource availability variations, which is a key characteristic of nondedicated Grid platforms.

The rest of the paper is organized as follows: The next section (Section 2) is devoted to the formal specification of our broadcast problems and of the target heterogeneous network. Section 3 is devoted to comparing topologies for the *series of broadcasts* problem. In Section 4, we move to the design of the optimal steady-state algorithm, when the target network is a directed acyclic graph (DAG). Our major result, in Section 5, is the extension of this result to the latter case of an arbitrary network graph. Next, in Section 6, we informally state two asymptotic results, for the *series of broadcasts* and the *pipelined broadcast* problems. We report some experimental data in Section 7. We briefly survey related work in Section 8, and we state some concluding remarks in Section 9.

2 Framework

The target architectural platform is represented by an edge-weighted directed graph $G = (V, E, c)$, as illustrated in Fig. 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, $1 \leq i \leq p, i \neq s$, are destination nodes which must receive all the data sent by P_s.

There are several scenarios for the operation of the processors, which will be discussed in Section 8. 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 $e_{j,k} : P_j \rightarrow P_k$ 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 1) P_k cannot initiate another receive operation before time-step $t + c_{j,k}$ (but, it can perform a send operation), and 2) 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-up costs are included in the values of the link capacities $c_{j,k}$. The optimization problem SERIES(V, E, c) is to maximize the throughput. We work out a little example in Section 3, using the platform represented in Fig. 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 PIPE(V, E, c, β, L) 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 in Fig. 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 the 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

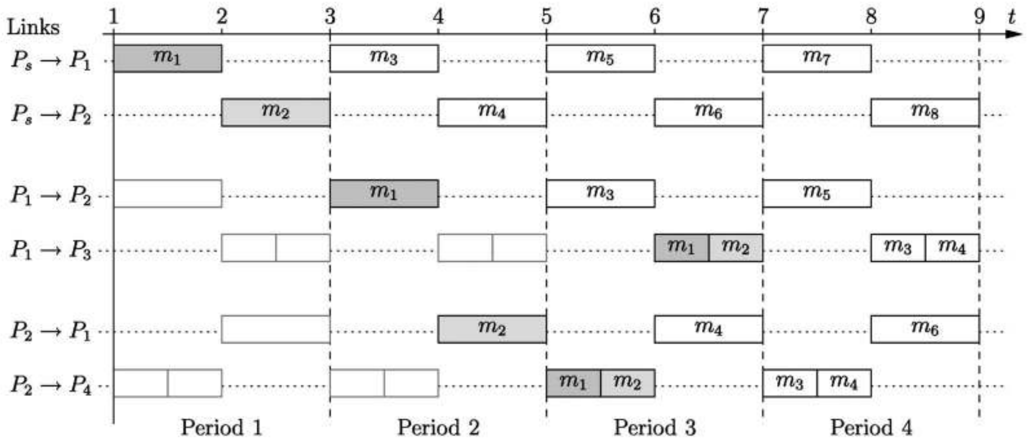

Fig. 2. An optimal schedule for the network of Fig. 1, achieving a throughput of 1 message broadcast every time-step.

hand, a feasible schedule for a series of broadcasts realizing this throughput is given in Fig. 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 Fig. 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 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 empty boxes), 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.

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:

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

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 $r \times c_{f(i),i} \leq 1$, where $f(i)$ is the parent of i in T. This constraint is satisfied for i as soon as (1) is verified for $f(i)$, so we can discard this constraint. In the following, we let TP(T) denote the throughput of a broadcast tree T.

What is the maximal throughput TP(T) that can be achieved using a subtree of the platform described on Fig. 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 Fig. 3a. Obviously, because of the one-port constraint for the source processor, this is the best throughput that can be achieved using this tree. A schedule reaching this throughput is represented in Fig. 3b.

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 Fig. 4a, whose optimal throughput is TP(T) $= 2/3$. Indeed, 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 TP(T) $= 2/3$, as illustrated in Fig. 4b. Overall, this is the best throughput that can be obtained with a broadcast tree in this network. The best throughput has been determined by an exhaustive search among all possible trees (what is easy on such a small platform). Note that finding the best spanning tree in a platform, with respect to throughput maximization, is a NP-Complete problem [11].

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

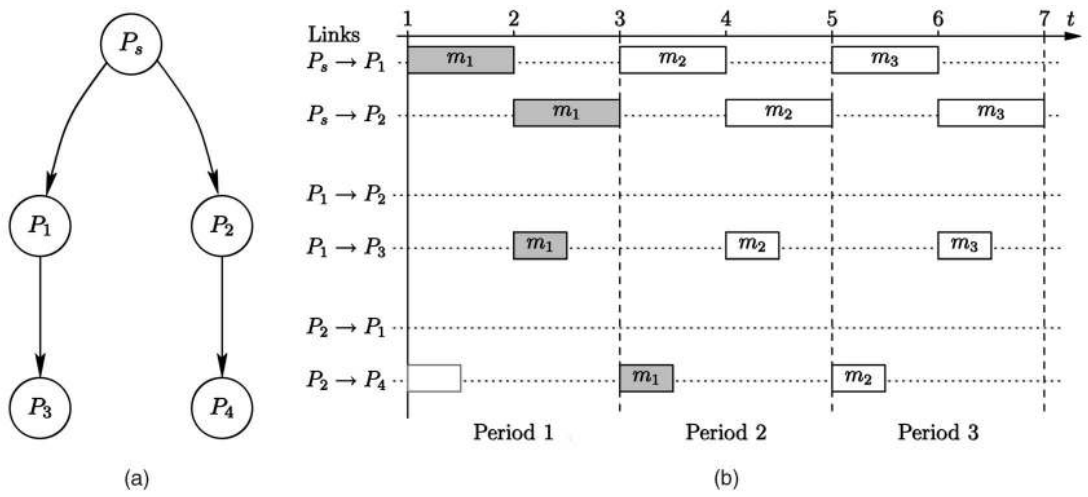

Fig. 3. Broadcasting a message from P_s using the first spanning tree (throughput: 1/2). (a) First broadcast tree and (b) corresponding schedule.

entry vertex, namely, the source processor. Can we 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 in Fig. 5a, 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 Fig. 5a. Because the first broadcast tree of Fig. 3a is a subgraph of the DAG, we can achieve a throughput at least $1/2$. However, it is possible to achieve an even better throughput. Fig. 5b 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 using DAGs (throughput of $\frac{4}{5}$ instead of $\frac{2}{3}$), and restricting to DAGs is less powerful than using the full network graph (throughput of 1 instead of $\frac{4}{5}$).

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). A first equation links the two previous quantities:

$$t_{j,k} = n_{j,k} \times c_{j,k}. \qquad (2)$$

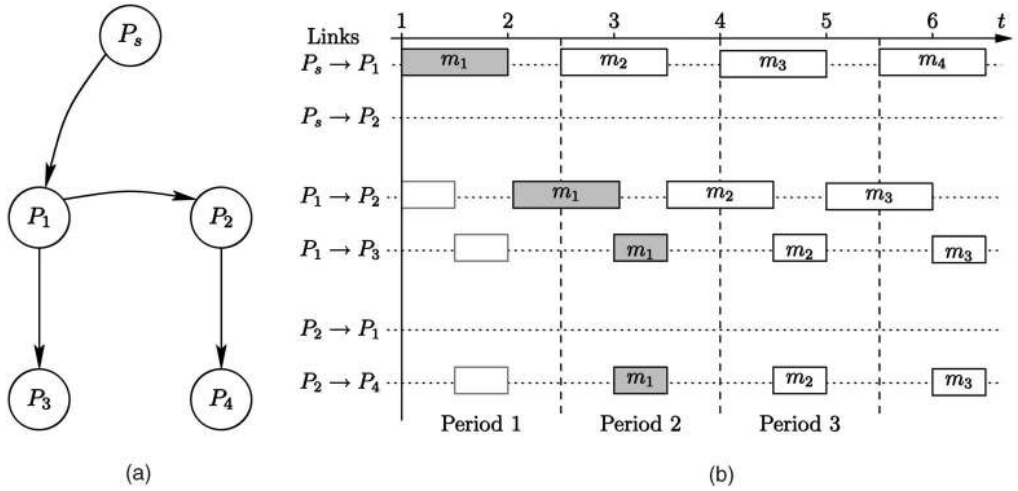

Fig. 4. Broadcasting a message from P_s using the second spanning tree (throughput: 2/3). (a) Second broadcast tree and (b) corresponding schedule.

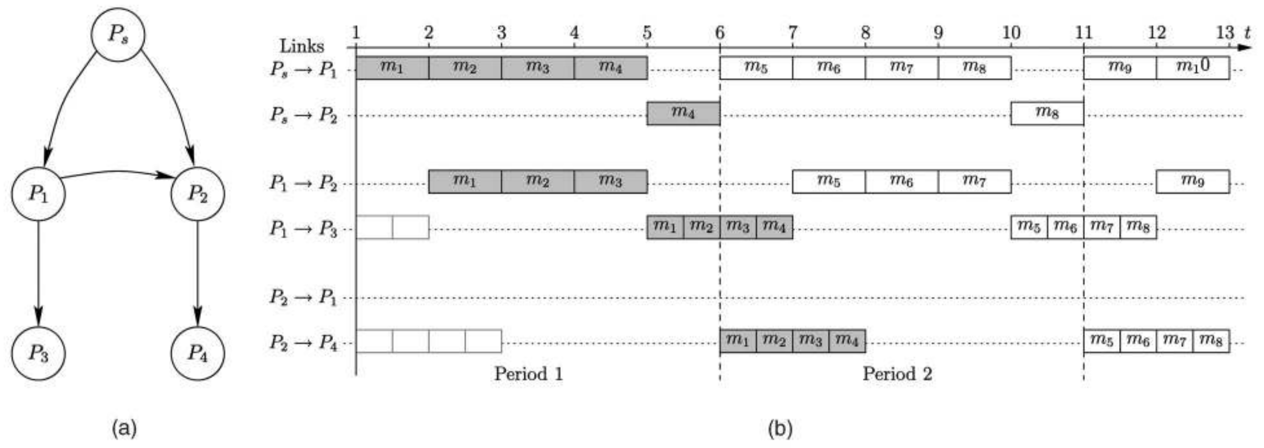

Fig. 5. Broadcasting a message from P_s using a DAG (throughput: 4/5). (a) Using a DAG and (b) corresponding schedule.

The activity on edge (P_j, P_k) in one time-unit is bounded:

$$\forall P_j, \forall P_k \qquad 0 \leq t_{j,k} \leq 1. \tag{3}$$

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

$$\forall P_j, \qquad \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leq 1 \qquad \text{(outgoing messages)} \tag{4}$$

$$\forall P_j, \qquad \sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leq 1 \qquad \text{(incoming messages).} \tag{5}$$

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

$$\forall P_j \text{ with } j \neq s, \qquad \sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP}. \tag{6}$$

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

$$\left\{\begin{array}{llll} \forall P_j, \forall P_k & t_{j,k} = n_{j,k} \times c_{j,k} & \forall P_j, & \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leq 1 \\ \forall P_j, \forall P_k & 0 \leq t_{j,k} \leq 1 & \forall P_j, & \sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leq 1 \\ & & \forall P_j \text{ with } j \neq s, & \sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP}. \end{array}\right.$$

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 [11]. 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 quantity by T. We get integer results for a steady-state operation with period T. There remains to show that 1) the schedule can be actually implemented, and 2) the schedule admits a *compact* description, i.e., of size polynomial in the input data.

For 1), 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 (4) and (5) are satisfied by each processor during the time interval:

$$\forall P_j, \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leq T \text{ (outgoing messages) and}$$
$$\sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leq T \text{ (incoming messages).}$$

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 2), because T is the least common multiple of values of the linear program solution, $\log(T)$ has polynomial size bit not T itself, so a time-step by time-step description of the schedule would be too large.

We solve both 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 [12, 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

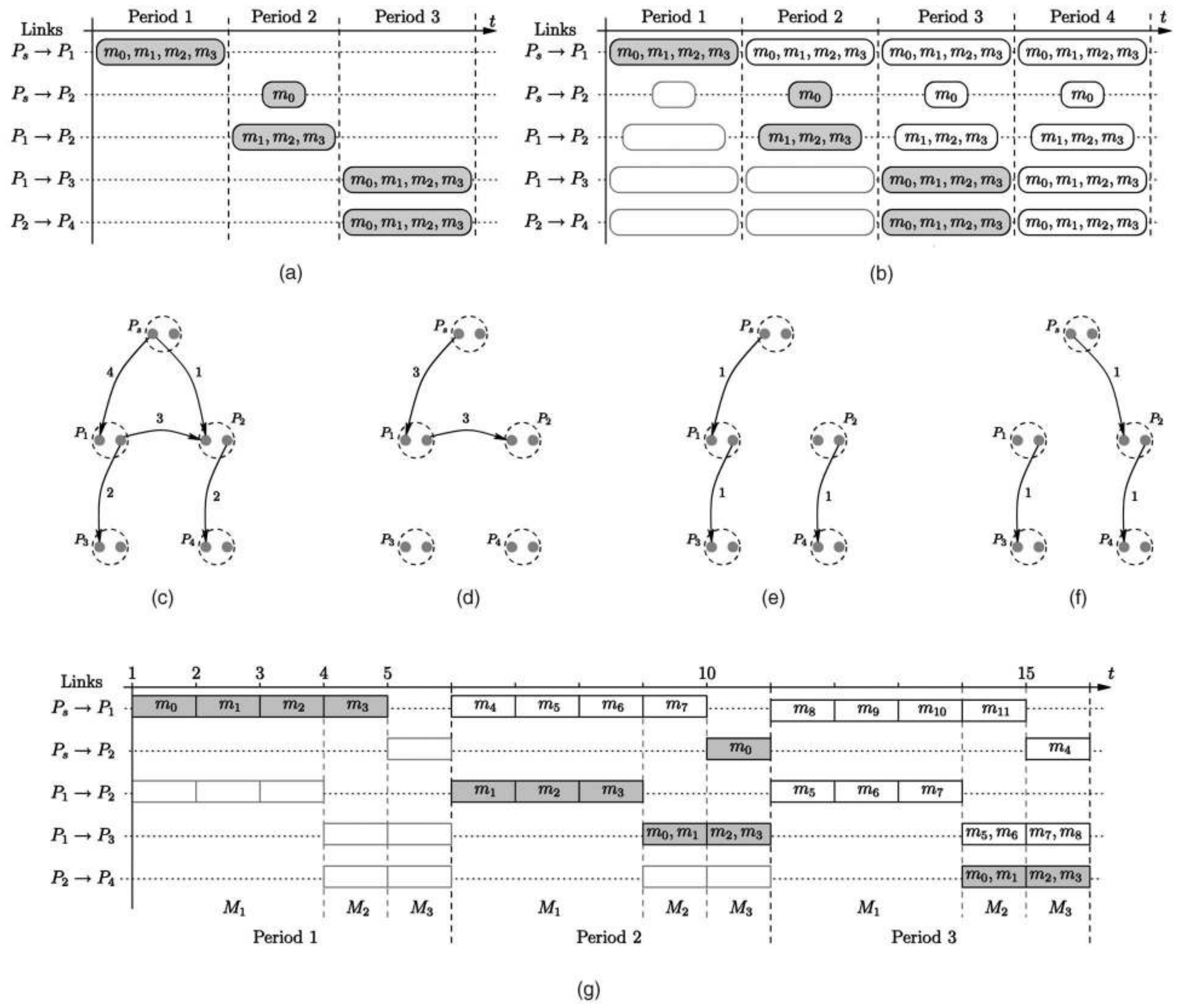

Fig. 6. Solution for the example of a broadcast on a DAG. (a) Basic schedule, (b) pipelined communications, (c) bipartite graph, (d) matching M_1, (e) matching M_2, (f) matching M_3, and (g) final schedule.

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. □

We come back to the example given in Fig. 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 schedule constructed in the proof [11] is represented in Fig. 6. Fig. 6a is a basic step of the schedule. Once pipelined, it gives the schedule of Fig. 6b. The last step is to use the edge-coloring algorithm to create a schedule where several receptions or emissions never overlap on a node. This algorithm decomposes the bipartite graph of the communications (Fig. 6c) into three matchings (Figs. 6d, 6e, and 6f). This leads to the final schedule of Fig. 6g.

5 Series of Broadcasts on a General Platform

In this section, we give the optimal solution to the SERIES problem for an arbitrary platform graph, which may include cycles. We proceed in several steps, using technically involved theoretical results from linear programming, network flows, and graph theory.

5.1 Sketch of Proof

As before, the target platform graph is modeled by a directed graph $G = (V, E, c)$. Each edge $(P_j, P_k) \in E$ is labeled by its capacity $c_{j,k}$, i.e., the time needed to transfer a unit-size message from P_j to P_k. The transfer time for Z different messages between P_j and P_k is equal to $Zc_{j,k}$. Each node operates under the one-port model, so that both incoming and outgoing communications have to be performed sequentially.

There is a large number of unit-size messages to broadcast. Initially, the source processor P_s holds all these messages. Our aim is to derive a periodic algorithm that achieves the optimal throughput TP, defined as the ratio of the number of messages broadcast per time-period T in steady-state, over the duration T of the period. Not only do we have to compute the optimal throughput TP, but also, we have to provide the actual construction of the periodic schedule. Our goal is to obtain a compact description of this schedule: the description of the behavior of each node during one period (i.e., the size of the code) must be polynomial in the size of the initial data. The sketch of our approach is the following:

1. We express the conditions that must be fulfilled at steady state by any periodic solution to the SERIES problem by means of a linear program. The solution of this linear program provides a lower bound for the completion time.
2. From the solution of the linear program, we derive a set of weighted trees that will be used to broadcast the different messages. We prove that the total weight of the trees enables us to reach the lower bound computed at the previous step.
3. From the set of trees, we derive a periodic solution, and we prove that it is possible to write the code of the broadcast algorithm with a size polynomial in the size of the initial data.

5.2 Lower Bound

In what follows, we give a set of linear constraints that must be fulfilled by any periodic solution at steady-state. We normalize the solution so that one unit-size message is broadcast to each processor every T^* time-steps, and we aim at minimizing the period T^*. Note that this is the dual problem of Section 4, where we aimed at maximizing the number of messages broadcast per time-unit. However, we (try to) keep similar notations: $n_{j,k}$ denotes the number of messages that transit along edge (P_j, P_k), and $t_{j,k}$ is the total occupation time of that edge. But things get more complicated, and we need new variables $x_i^{j,k}$, as explained below.

For any node P_j, we denote by $\mathcal{N}^{out}(P_j)$ its output neighbors, i.e., the set of nodes P_k such that $(P_j, P_k) \in E$; similarly, $\mathcal{N}^{in}(P_j)$ is the set of the input neighbors of P_j, i.e., nodes P_k such that $(P_k, P_j) \in E$.

Since we deal with broadcast operations, the same messages are sent to all the nodes. But, because of the pipelining, several different messages are likely to circulate simultaneously in the network. We fictitiously distinguish the messages that are sent by the source P_s to each processor P_i, even in the end the same messages will have been sent, but maybe according to a different ordering, and via different routes. More precisely, we denote by $x_i^{j,k}, \forall P_i \in V, \forall (P_j, P_k) \in E$ the fractional number of unit-size messages sent by the source P_s to P_i and that transit on the edge between P_j and P_k:

Source and destination: The first set of constraints states that the total number of messages destined to P_i and which are sent from the source P_s every period is indeed 1; also, the total number of messages which are actually received by P_i every period is also equal to 1:

$$\forall i, \qquad \sum_{P_j \in \mathcal{N}^{out}(P_s)} x_i^{s,j} = 1 \tag{7}$$

$$\forall i \neq s, \qquad \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1. \tag{8}$$

Conservation law: The second set of constraints states a conservation law at any intermediate processor $P_j \neq P_s, P_i$ for the messages sent to P_i:

$$\forall j, P_j \neq P_s \text{ and } P_j \neq P_i, \qquad \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k}. \tag{9}$$

This constraint reads: for each index i and each intermediate processor P_j, $j \neq i$, the number of messages destined to P_i which arrive at P_j each time-period is the same as the number of same type messages that go out of P_j. This conservation law is only valid in steady-state operation, it does not apply to the initialization and clean-up phases.

Link occupation: The following set of constraints is related to the number of distinct messages that are transferred through each edge. Let us denote by $n_{j,k}$ the total number of messages that transit on the communication link between P_j and P_k. We know that for each i, the fraction $x_i^{j,k}$ of the messages sent to P_i does transit on this link. The main difficulty is that the messages transiting on the link and sent to different P_i's may be partly the same, since the same messages are overall sent to all the nodes. Therefore, the constraint $n_{j,k} = \sum_i x_i^{j,k}$, that would hold true for a scatter operation, may be too pessimistic. Since our aim is to find a lower bound for the execution time, we consider that all the messages transiting between P_j and P_k are all subsets of the same set, namely, the largest one. In other words, we write the following constraints for the occupation time $t_{j,k}$ of the link (P_j, P_k):

$$\forall (P_j, P_k) \in E, \quad n_{j,k} = \max_i x_i^{j,k} \tag{10}$$

$$\forall (P_j, P_k) \in E, \quad t_{j,k} = n_{j,k} c_{j,k}. \tag{11}$$

We also need to write down the constraints stating that communication ports for both incoming and outgoing communications are not saturated (one-port model). Let $t_j^{(in)}$ be the time spent by P_j for incoming communications, and $t_j^{(out)}$ the time spent for outgoing ones:

$$\forall j, \quad t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \tag{12}$$

$$\forall j, \quad t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k}. \tag{13}$$

Execution time: The last set of constraints is related to the overall period length T^* required for broadcasting a unit size message. The constraints simply state that T^* is larger than the occupation time of any edge and any incoming or outgoing communication port:

$$\forall j, k, \quad T^* \geq t_{j,k}, \tag{14}$$

$$\forall j, \quad T^* \geq t_j^{(in)}, \tag{15}$$

$$\forall j, \quad T^* \geq t_j^{(out)}. \tag{16}$$

Finally, we gather all the constraints into the following linear program, which provides a lower bound for T^*, the time needed to broadcast one unit-size message:

Steady-State Broadcast Problem on a Graph SSB(G)
Minimize T^*,
subject to

$$\begin{cases} \forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} x_i^{s,j} = 1 & (7)\\ \forall i \neq s, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 & (8)\\ \forall j, P_j \neq P_s \text{ and } P_i, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \\ & \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} & (9)\\ \forall (P_j, P_k) \in E, & n_{j,k} = \max_i x_i^{j,k} & (10)\\ \forall (P_j, P_k) \in E, & t_{j,k} = n_{j,k} c_{j,k} & (11)\\ \forall j, & t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} & (12)\\ \forall j, & t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} & (13)\\ \forall j,k, & T^* \geq t_{j,k} & (14)\\ \forall j, & T^* \geq t_j^{(in)} & (15)\\ \forall j, & T^* \geq t_j^{(out)} & (16). \end{cases}$$

5.3 Weighted Broadcast Trees

The solution of the linear program clearly provides a lower bound for the period length needed to broadcast one unit-size message. Nevertheless, it is not clear that this bound can be achieved, because of the assumption stating that all the messages transiting on a given edge are all subsets of the largest set (11). In this section, we first prove that it is possible to find a set of broadcast trees realizing exactly the lower bound, using Edmond's Branching theorem. Unfortunately, the number of trees produced by this theorem may be exponential in the problem size. Fortunately, there exists a weighted version of Edmond's Branching theorem, that produces the desired polynomial number of trees.

5.3.1 Broadcast Trees and Edmond's Branching Theorem

Edmond's Branching theorem applies to nonweighted graphs only, so we transform the previous graph, weighted by the $n_{j,k}$, into a multigraph. Let us denote by N the least common multiple of all the denominators of the $n_{j,k}$'s and the $x_i^{j,k}$'s, so that $\forall i,j,k$, $Nn_{j,k}$, and $Nx_i^{j,k}$ have integer values. Moreover, let us denote by $G^{(m)} = (V, E)$ the multigraph such that there exists exactly $Nn_{j,k}$ edges between P_j and P_k.

Edmond's branching theorem [13] shows the relationship between the number (denoted as $\kappa(G, P_0)$) of edges whose deletion makes some vertex P_i unreachable from P_s and the number of edge-disjoint spanning trees rooted at P_s.

Theorem 2 (Edmond's Branching Theorem). *The number of edge-disjoint spanning trees rooted at P_0 is exactly $\kappa(G, P_0)$.*

We know prove that the number of edges whose deletion makes some vertex unreachable from the source is in fact N.

Theorem 3. $\kappa(G, P_0) = N$.

Proof. We prove this theorem by in two steps:

- $\kappa(G, P_0) \geq N$: Consider any $P_i \in V$ distinct from the source P_s. The values $x_i^{j,k}$ define a flow of total weight N between P_s and P_i. Indeed, we have:

$$\begin{cases} \forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} N x_i^{s,j} = N & \text{by (7)}\\ \forall j, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} N x_i^{j,i} = N & \text{by (8)}\\ \forall j, P_j \neq P_0 \text{ and } P_i, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} N x_i^{j,k} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} N x_i^{k,j} & \text{by (9).} \end{cases}$$

  Therefore, by the Max-flow, Min-cut Theorem of Ford and Fulkerson [14], the minimal cut of G between P_s and P_i is at least N, so that at least N edges have to be deleted in order to disconnect P_s and P_i. Since the above property holds true for any P_i, then $\kappa(G, P_0) \geq N$.
- $\kappa(G, P_0) \leq N$. Suppose that $\kappa(G, P_0) = N' > N$. Then, by the Max-flow, Min-cut Theorem of Ford and Fulkerson, for each P_i, there exists a flow a weight N' in G between P_s and P_i. Let $y_i^{j,k}$ denote the value of this flow on the edge between P_j and P_k (clearly, $y_i^{j,k} \leq Nn_{j,k}$ by construction), and let us denote by $z_i^{j,k} = \frac{y_i^{j,k}}{N'}$, so that the $z_i^{j,k}$'s define a flow of weight 1 between P_s and P_i. Then,

$$\begin{cases} \forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} z_i^{s,j} = 1 & (7)\\ \forall i, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} z_i^{i,j} = 1 & (8)\\ \forall j, P_j \neq P_s \text{ and } P_j \neq P_i & & \\ \quad \sum_{P_k \in \mathcal{N}^{in}(P_j)} z_i^{j,k} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} z_i^{k,j} & & (9)\\ \forall (P_j, P_k) \in E, & n'_{j,k} = \max_i z_i^{j,k} \leq \frac{N}{N'} n_{j,k} & (10)\\ \forall (P_j, P_k) \in E, & t'_{j,k} = n'_{j,k} c_{j,k} \leq \frac{N}{N'} t_{j,k} & (11)\\ \forall j, & t_j'^{(in)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} & \\ & t'_{j,k} \leq \frac{N}{N'} t_j^{(in)} & (12)\\ \forall j, & t_j'^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} & \\ & t'_{j,k} \leq \frac{N}{N'} t_j^{(out)} & (13). \end{cases}$$

  Therefore, there would exist a solution of the linear program with a completion time of $\frac{N}{N'} T^* < T^*$, which is a contradiction. Thus, $\kappa(G, P_0) \leq N$. □

Therefore, by Edmond's Branching theorem, there exist N disjoint broadcast trees in G^m. There exist several implementations of Edmond's Branching theorem, but the number of different trees is of order $O(N)$. Unfortunately, a solution consisting of N broadcast trees is not compact enough for our purpose, since its encoding would take at least of order $O(N|V|)$. Indeed, since N is the least common multiple of the denominators of the $x_i^{j,k}$s and the $n_{j,k}$s, it can be encoded in size of order $|V||E|\log(\max(x_i^{j,k}, n_{j,k}))$. Moreover, the $x_i^{j,k}$s and the $n_{j,k}$s are the solution of a linear system, whose right-hand side and left-hand size matrix coefficients are initial data. Therefore, N can be encoded in polynomial size. Nevertheless, the encoding of the trees would take at least $|V|N$ bits, and would therefore be exponential in the size of original data. Fortunately, there exists a weighted version of Edmond's Branching theorem which produces a polynomial number of trees, as shown in next section.

5.3.2 Weighted Version of Edmond's Branching Theorem

We use the following result, whose proof can be found in [12, vol. B, chapter 53].

Theorem 4. *Let $G = (V, E, Nn_{j,k})$ denote a weighted directed graph. There exist k_T trees $T_1, \ldots, T_{k_T}$ trees, with integer*

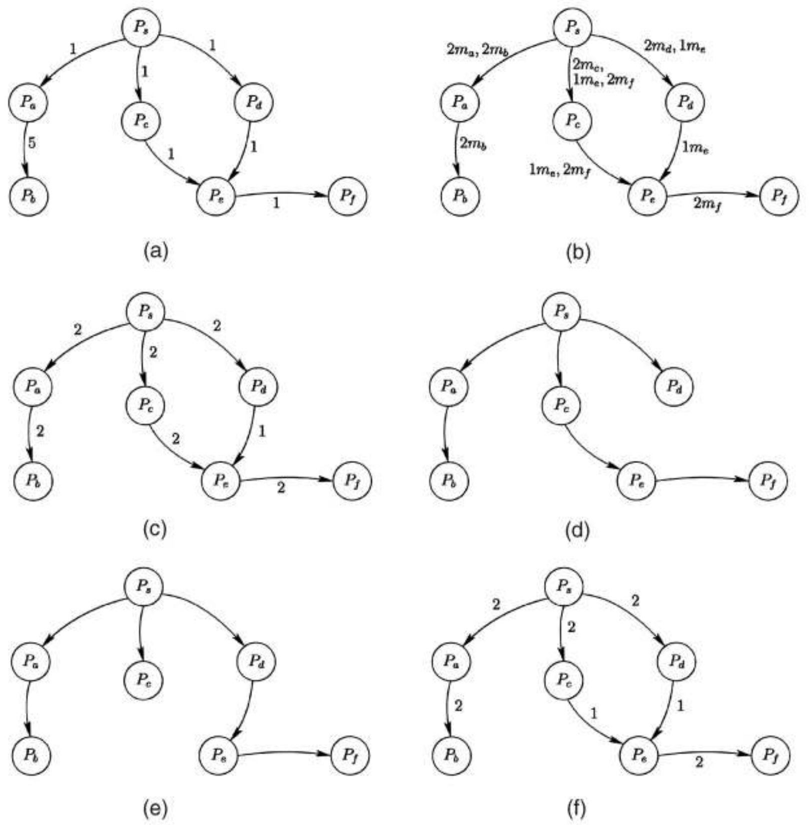

Fig. 7. Example where $m_{j,k} < Nn_{j,k}$. The optimal steady-state broadcast time T^* for one message is 5 time-units, due to edge (P_a, P_b). Fig. 7b describes the results multiplied by the least common multiple $N = 2$, and Fig. 7c reports the maximum values of $Nx_i^{j,k}$ on each edge. Figs. 7d and 7e are the two broadcast trees extracted from the previous figure, each of them with a weight of $\lambda_l = 1$. Finally, Fig. 7f 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 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, $\lambda_2 = 1$, and (f) sum of both trees—graph of the $m_{j,k}$'s.

weights $\lambda_1, \ldots, \lambda_{k_T}$, such that $\forall j, k, \sum_l \lambda_l \chi_{j,k}^T(T_l) \leq Nn_{j,k}$, where $\chi_{j,k}^T(T_l) = 1$ if $(P_j, P_k) \in T_l$ and 0 otherwise, and such that $\sum_l \lambda_l$ is maximized. Moreover, the trees can be found in strongly polynomial time and by construction, $k_T \leq |V|^3 + |E|$.

We then have the following lemma, whose proof is quite similar to the proof of Theorem 3 and uses the transformation of G into a multigraph.

Lemma 1. $\sum_l \lambda_l = \kappa(G, P_0) = N$.

Finally, we prove that the description using a set of weighted trees is not too large:

Lemma 2. *The set of trees can be encoded in polynomial size with respect to initial data.*

Proof. The number of trees is bounded by $|V|^3 + |E|$ and, therefore, the set of trees can be encoded in size of order $|V|(|V|^3 + |E|)$. Moreover, $\forall l, \lambda_l \leq N \max n_{j,k}$, and both N and $\max n_{j,k}$, can be encoded in polynomial size with respect to the initial data, as proved above. □

Therefore, the weighted version of Edmond's Branching theorem produces in polynomial time a set of weighted trees, whose encoding is compact enough, for our purpose. We will use these trees in order to broadcast the different messages. In what follows, let $m_{j,k}$ be the overall number of messages that transit between P_j and P_k on the different trees, i.e.,

$$m_{j,k} = \sum_l \lambda_l \chi_{j,k}^T(T_l) \leq Nn_{j,k}. \qquad (17)$$

Moreover, since the overall weight of the trees is N, and all the trees span the whole platform, we have:

$$\forall k, \sum_{P_j \in \mathcal{N}^{in}(P_k)} m_{j,k} = N. \qquad (18)$$

To conclude this section, we point out that we may have $m_{j,k} < Nn_{j,k}$ on some edges. Consider the toy-example in Fig. 7. Not all communications arising from the linear program $\mathrm{SSB}(G)$ are actually used in the trees: Some are discarded, because they do not improve the throughput of the broadcasts; but they do not interfere with other communications either. In other words, these communications are "useless" but "harmless."

5.4 Communication Scheduling

Our goal is to use the broadcast trees defined above to perform the series of broadcasts. Thus, we need to find a schedule for communications. Indeed, since several broadcast trees will be used, node P_k will receive messages from

```
Step $i$ (communications corresponding to matching $M_i$)
  for all $(P_j, P_k)$, $i \in M^{(j,k)}$ do
    for all $l$, $(P_j, P_k) \in T_l$ do
      $P_j$ sends the $\frac{\mu_i s \lambda_l}{\sum_{i \in M^{(j,k)}} \mu_i}$ messages of the set $m_j^l(r-1)$ to $P_k$
      $m_k^l(p) = m_k^l(r) \cup m_j^l(r-1)$ the messages sentby $P_j$
```

Fig. 8. Sketch of the scheduling algorithm during the ith period.

several nodes P_j and, since P_k is only able to handle one receiving operation at the same time, communications to P_k (and from P_j) need to be scheduled carefully. We revisit the edge coloring theorem used in the proof of Theorem 1 with more details, so as to extract disjoint matchings out of the set of communications: In a word, the situation is more complex here, because of the need to partition the matchings themselves into the different broadcast trees which they intersect with.

5.4.1 *Weighted Bipartite Graph*

As in the proof of Theorem 1, we construct a weighted bipartite graph $G^M = (V', E', m_{j,k}c_{j,k})$ to represent the set of communications. Let us denote

$$V' = V^{out} \cup V^{in} = (P_1^{out}, \dots, P_p^{out}) \cup (P_1^{in}, \dots, P_p^{in}),$$

where $p = |V|$ is the number of processors. In the bipartite graph, the edge between P_j^{out} and P_k^{in} is weighted by the quantity $m_{j,k}c_{j,k}$, which is the time necessary to transfer the overall amount of data transiting on this edge on the different trees. In order to schedule the communications, we use the refined version of the *Edge Coloring Lemma* (see [12, vol. A, chapter 20]).

Theorem 5. *Let $G^M = (V, E', m_{j,k}c_{j,k})$ be a bipartite weighted graph. There exist k_M matchings $M_1, \dots, M_{k_M}$, with integer weights $\mu_1, \dots, \mu_{k_M}$, such that*

$$\forall j, k, \sum_i \mu_i \chi_{j,k}^M(M_i) = m_{j,k}c_{j,k}, \tag{19}$$

where $\chi_{j,k}^M(M_i) = 1$ if $(P_j, P_k) \in M_i$ and 0 otherwise, and

$$\sum_i \mu_i = \max(\max_j \sum_k m_{j,k}c_{j,k}, \max_k \sum_j m_{j,k}c_{j,k}).$$

Moreover, the matchings can be found in strongly polynomial time and by construction,

$$k_M \leq |E|.$$

We now prove that $\sum_i \mu_i$ is not greater than NT^*.

Lemma 3. $\sum_i \mu_i \leq NT^*$.

Proof. By (17), $m_{j,k} \leq N n_{j,k}$. Thus,

$$\sum_j m_{j,k}c_{j,k} \leq N \sum_j n_{j,k}c_{j,k} \leq NT^* \text{ by (13) and (16)}$$

$$\text{and } \sum_k m_{j,k}c_{j,k} \leq N \sum_k n_{j,k}c_{j,k} \leq NT^* \text{ by (12) and (15).}$$

Thus, since $\sum_i \mu_i = \max(\max_j \sum_k m_{j,k}c_{j,k}, \max_k \sum_j m_{j,k} c_{j,k})$, then $\sum_i \mu_i \leq NT^*$.

In fact, the inequality is indeed an equality, but the simplest way to show it is to exhibit the periodic schedule (see below). □

5.4.2 *Broadcasting Algorithm*

In this section, we give the precise communication scheduling during one period, i.e., the sketch of the code used to implement the broadcasts in steady state. Let us define, $\forall (P_j, P_k)$ such that $m_{j,k} \neq 0$,

$$M^{(j,k)} = \{i, (P_j^{out}, P_k^{in}) \in M_i\}$$
$$\text{the set of matchings containing } (P_j^{out}, P_k^{in})$$

and

$$T^{(j,k)} = \{l, (P_j, P_k) \in T_l\} \text{ the set of trees containing } (P_j, P_k).$$

Thus, we can notice that

$$\text{by (19)}, \forall (P_j, P_k), \sum_{i \in M^{(j,k)}} \mu_i = m_{j,k}c_{j,k}$$
$$\text{and by (17)}, \forall (P_j, P_k), \sum_{l \in T^{(j,k)}} \lambda_l = m_{j,k}.$$

Let us denote by

$$s = \operatorname{lcm}_{j,k} \left(\sum_{i \in M^{(j,k)}} \mu_i \right). \tag{20}$$

In the following, we exhibit an optimal periodic schedule: the period length is $T^{\text{per}} = NsT^*$, and Ns messages are broadcast every T^{per} time-steps, thereby achieving the optimal throughput $1/T^*$.

Let $m_j^l(q)$ be the set of messages received by node P_j from its father in the tree T_l during the qth period. The sketch of the scheduling algorithm during the ith period is depicted in Fig. 8.

We prove the correctness of this algorithm as follows:

Duration of step i: In order to estimate the duration of step i, we need to evaluate, for each P_j such that $(P_j^{out}, P_k^{in}) \in M_i$, the time needed by P_j to send all the messages:

$$\begin{aligned} \sum_{l \in T^{(j,k)}} \frac{\mu_i s \lambda_l c_{j,k}}{\sum_{i \in M^{(j,k)}} \mu_i} &= \frac{\mu_i s}{\sum_{i \in M^{(j,k)}} \mu_i} \left(\sum_{l \in T^{(j,k)}} \lambda_l \right) c_{j,k} \\ &= \frac{\mu_i s}{\sum_{i \in M^{(j,k)}} \mu_i} m_{j,k}c_{j,k} \qquad \text{by (17)} \\ &= \mu_i s \qquad \text{by (19).} \end{aligned}$$

This result does not depend on j. Furthermore, the communications involving different P_j's can be handled in parallel, because they belong to a matching. Therefore, step i can be executed within $\mu_i s$ time-units.

Length of the period: The duration of the period T^{per} is the sum of the duration of the different steps:

$$\sum_i \mu_i s \leq NT^* s = T^{\text{per}}.$$

Number of messages $M(r,j,k)$ received by P_k and coming from P_j: During the rth period:

$$M(r,j,k) = \sum_{i \in M^{(j,k)}} \sum_{l \in T^{(j,k)}} \frac{\mu_i s \lambda_l}{\sum_{i \in M^{(j,k)}} \mu_i} = s \sum_{l \in T^{(j,k)}} \lambda_l = s m_{j,k}$$

by (17).

Total number of messages received by P_k: During the rth period. Since all the messages are sent along the edges of the different trees, all the messages received by P_k are different, and are different from those received during previous periods. Therefore, the overall number of messages received by node P_k during one period is given by

$$s \sum_j m_{j,k} = sN \text{ by (18)}.$$

Therefore, during one period of duration $T^{\text{per}} = NsT^*$, each node receives exactly Ns new different messages. Therefore, the overall throughput of the SERIES algorithm during one period is $\frac{1}{T^*}$, hence its optimality. Finally, because the actual length of the period is the sum of the duration of the different steps, we derive that $\sum_i \mu_i s = T^{\text{per}}$, hence $\sum_i \mu_i = NT^*$, as claimed in the proof of Lemma 3.

6 Asymptotic Optimality

Due to the lack of space, we informally state two important results, which are both detailed (formal statement and complete proof) in the extended version [11] of this paper. These results are inspired by the work of Bertsimas and Gamarnik [15], who use a fluid relaxation technique to prove the asymptotic optimality of a simpler packet routing problem.

6.1 Asymptotic Optimality for the Series Problem

The periodic schedule described in Section 5.4.2 is asymptotically optimal: Basically, no scheduling algorithm (even nonperiodic) can execute more broadcast operations in a given time-frame than ours, up to a constant number of operations.

6.2 Asymptotic Optimality for the Pipelined Problem

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 PIPE(V, E, c, β, L) 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 schedule described in Section 5.4.2, we can prove a result of asymptotic optimality for the PIPE optimization problem. This is a surprising result, because the PIPE problem deals with makespan minimization, not throughput optimization. The key idea is to determine a number of packets ν such that both 1) the size of each packet L/ν is large enough so that start-up times have a little overhead on the execution time, and 2) the number ν of packets is large enough so that the initialization and the clean-up phase can be neglected in front of the duration of steady-state operation. It is shown in [11] how the choice of $\nu = O(\sqrt{L})$ enables to achieve both goals 1) and 2), thereby leading to an asymptotically optimal schedule.

7 Experiments

In this section, we work out a complete example. The platform is generated by Tiers, a random generator of topology [16]. The bandwidth of the links are randomly chosen, and the topology is represented on Fig. 9a.

Fig. 9b 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 $Nx_k^{i,j} = y$, so 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 in Fig. 10, where both the logical tree and the communications extracted from Fig. 9b 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). The same observation was made for the toy example at the end of Section 5.3.2. 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 Related Work

The *atomic broadcast* problem has been studied under different models to deal with the heterogeneity of the target architecture. Banikazemi et al. [17] consider a simple model in which the heterogeneity among processors is characterized by the speed of the sending processors. In this model, the interconnection network is fully connected (a complete graph), and each processor P_i requires t_i time-units to send a (normalized) message to any other processor. The authors discuss that this simple model of heterogeneity can well describe the different communication delays in a heterogeneous cluster. They introduce the Fastest Node First

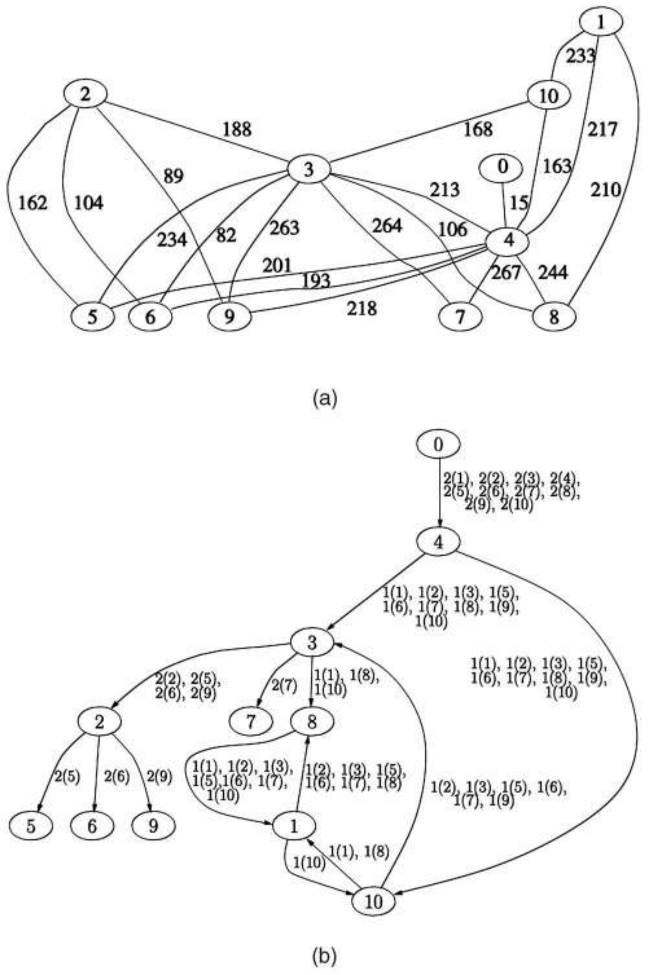

Fig. 9. Experiments on a given topology. (a) Topology. Edge e is labeled by its bandwidth $bw(e)$. The cost of a transfer is $c(e) = 1,000/bw(i)$ for a single message. (b) Communication graph.

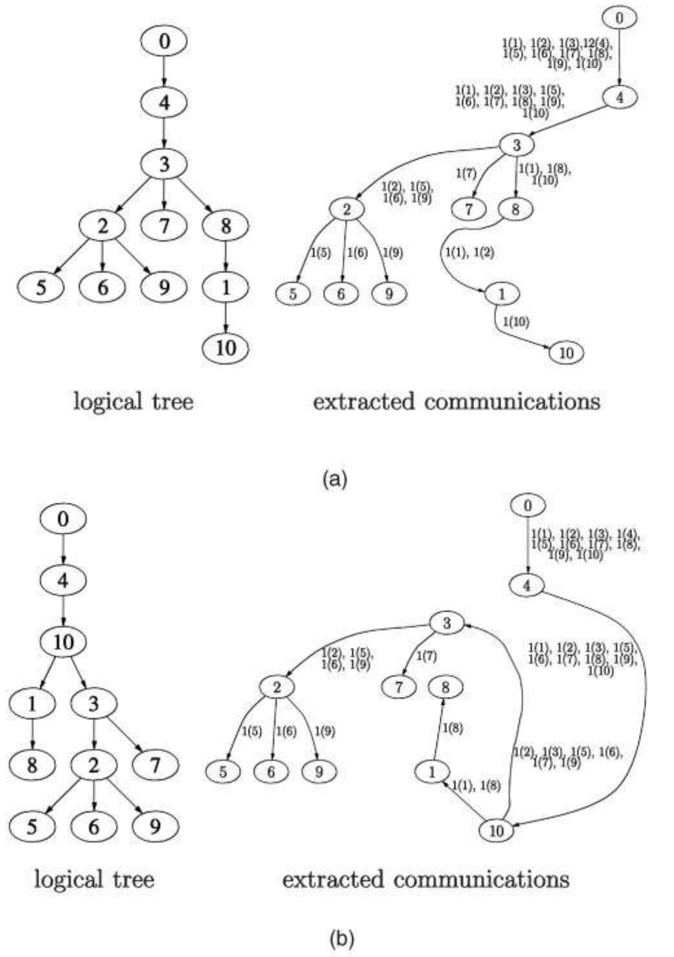

Fig. 10. Broadcast trees. (a) First broadcast tree (broadcasting 1 message) and (b) second broadcast tree (broadcasting 1 message).

(FNF) heuristic: To construct a good broadcast tree, it is better to put fastest processors (processors that have the smallest sending time) at the top of tree. Some theoretical results (NP-completeness and approximation algorithms) have been developed for the problem of broadcasting a message in this model: see [18], [19], [20].

A more complex model is introduced in [21]: it takes not only the time needed to send a message into account, but also the time spent for the transfer through the network, and the time needed to receive the message. All these three components have a fixed part, and a part proportional to the length of the message.

Yet, another model of communication is introduced in [22], [23]: the time needed to transfer the message between any processor pair (P_i, P_j) is supposed to be divided into a start-up cost $T_{i,j}$ and a part depending on the size m of the message and the transmission rate $B_{i,j}$ between the two processors, $\frac{m}{B_{i,j}}$. Since the message size is a constant in the case of a broadcast, the total communication time between P_i and P_j is $C_{i,j} = T_{i,j} + \frac{m}{B_{i,j}}$. In [22], some heuristics are proposed for the broadcast and the multicast using this model.

All previous models assume the *one port* protocol, which we used throughout this paper: a given processor can send data to at most one neighbor processor at a time. Usually, overlapping this operation with one receiving (of independent data) is allowed.

Other collective communications, such as multicast, scatter, all-to-all, gossiping, and gather (or reduce) have been studied in the context of heterogeneous platforms: see [24], [25] and the references provided in [11].

As mentioned in the introduction, Moore and Quinn [9] and Desprez et al. [10] already investigated the *Series of broadcasts* problems, but with a different perspective: they focus on optimizing their performance of a series of broadcast operations from distinct source nodes. In this problems, either we look for the optimal order (that is the case in the paper of Moore and Quinn), or the order of the sending source nodes is fixed (in the paper of Desprez et al.), but in all cases, the goal is to minimize the contention between several concurrent broadcasts from distinct sources. These studies are done on homogeneous networks, most of the examples and simulations are conducted on hypercube. The measure of the performance is the makespan of the (short) series of broadcasts.

The main difference in our approach is that we aim at maximizing the throughput of a series of broadcasts from the same source, which is close to a fluid broadcast from a source node, by taking into account the heterogeneity of the platforms: If several paths connect a node to another, they might be used concurrently to increase the throughput of the operation. Although both problems are known as *series of broadcasts*, they strike different questions and call for distinct answers.

9 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 pipelined 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 [11].

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. It is important to point out that a concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

The recognition of broadcasting as a key communication primitive is widely established. Because our approach applies to the broadcast of a single (long) message as well as to a succession of broadcasts, we believe that this is a key improvement over existing results for heterogeneous platforms. There have been several papers dealing with broadcasting on heterogeneous platforms, however, they only deal with heuristics devoted to the design of a single spanning tree. We show that several trees should be used in parallel, and we provide an efficient (polynomial) way to determine the best way to orchestrate the communications so as to squeeze the most out of the available platform bandwidth.

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.

Acknowledgments

The authors thank the reviewers for their helpful comments and suggestions, which greatly improved the final version of the paper.

References

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

[2] S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computers,* vol. 38, no. 9, pp. 1249-1268, Sept. 1989.

[3] J. Watts and R. Van De Geijn, "A Pipelined Broadcast for Multidimensional Meshes," *Parallel Processing Letters,* vol. 5, no. 2, pp. 281-292, 1995.

[4] Y.-C. Tseng, S.-Y. Wang, and C.-W. Ho, "Efficient Broadcasting in Wormhole-Routed Multicomputers: A Network-Partitioning Approach," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 1, pp. 44-61, Jan. 1999.

[5] H. Ko, S. Latifi, and P. Srimani, "Near-Optimal Broadcast in All-Port Wormhole-Routed Hypercubes Using Error-Correcting Codes," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 3, pp. 247-260, Mar. 2000.

[6] S.-Y. Wang and Y.-C. Tseng, "Algebraic Foundations and Broadcasting Algorithms for Wormhole-Routed All-Port Tori," *IEEE Trans. Computers,* vol. 49, no. 3, pp. 246-258, Mar. 2000.

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

[8] K. Hwang and Z. Xu, *Scalable Parallel Computing.* McGraw-Hill, 1998.

[9] J. Moore and M. Quinn, "Generating an Efficient Broadcast Sequence Using Reflected Gray Codes," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 11, pp. 1117-1122, Nov. 1997.

[10] F. Desprez, P. Fraigniaud, and B. Tourancheau, "Successive Broadcast on Hypercube," Technical Report CS-93-210,The Univ. of Tennessee—Knoxville, 1993.

[11] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Optimizing the Steady-State Throughput of Broadcasts on Heterogeneous Platforms Heterogeneous Platforms," Technical Report RR-2003-34LIP, ENS Lyon, France, June 2003.

[12] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency,* series on algorithms and combinatorics, Springer-Verlag, vol. 24, 2003.

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

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* The MIT Press, 1990.

[15] D. Bertsimas and D. Gamarnik, "Asymptotically Optimal Algorithm for Job Shop Scheduling and Packet Routing," *J. Algorithms,* vol. 33, no. 2, pp. 296-318, 1999.

[16] K. Calvert, M. Doar, and E. Zegura, "Modeling Internet Topology," *IEEE Comm. Magazine,* vol. 35, no. 6, pp. 160-163, June 1997.

[17] M. Banikazemi, V. Moorthy, and D.K. Panda, "Efficient Collective Communication on Heterogeneous Networks Of Workstations," *Proc. 27th Int'l Conf. Parallel Processing (ICPP '98),* 1998.

[18] N. Hall, W.-P. Liu, and J. Sidney, "Scheduling in Broadcast Networks," *Networks,* vol. 32, no. 14, pp. 233-253, 1998.

[19] P. Liu and T.-H. Sheng, "Broadcast Scheduling Optimization for Heterogeneous Cluster Systems," *Proc. SPAA 2000, 12th Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 129-136, 2000.

[20] P. Liu, "Broadcast Scheduling Optimization for Heterogeneous Cluster Systems," *J. Algorithms,* vol. 42, no. 1, pp. 135-152, 2002.

[21] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan, "Communication Modeling of Heterogeneous Networks of Workstations for Performance Characterization of Collective Operations," *Proc. HCW '99, Eighth Heterogeneous Computing Workshop,* pp. 125-133, 1999.

[22] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient Collective Communication in Distributed Heterogeneous Systems," *Proc. ICDCS '99 19th Int'l Conf. Distributed Computing Systems,* pp. 15-24, 1999.

[23] "Adaptive Communication Algorithms for Distributed Heterogeneous Systems," *J. Parallel and Distributed Computing,* vol. 59, no. 2, pp. 252-279, 1999.

[24] P. Liu and D.-W. Wang, "Reduction Optimization in Heterogeneous Cluster Environments," *Proc. 14th Int'l Parallel and Distributed Processing Symp. (IPDPS 2000),* 2000.

[25] R. Libeskind-Hadas, J.R.K. Hartline, P. Boothe, G. Rae, and J. Swisher, "On Multicast Algorithms for Heterogeneous Networks of Workstations," *J. Parallel and Distributed Computing,* vol. 61, no. 11, pp. 1665-1679, 2001.

Olivier Beaumont received the PhD degree from the Université de Rennes in 1999. He is currently an associate professor in the LaBRI laboratory in Bordeaux. His main research interests are parallel algorithms on distributed memory architectures.

Arnaud Legrand received the PhD degree from Ecole normale supérieure de Lyon in 2003. He is currently a postdoctoral researcher in the LIP laboratory at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques.

Loris Marchal is currently a PhD student in the LIP laboratory at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques. He is a student member of the IEEE and the IEEE Computer Society.

Yves Robert received the PhD degree from Institut National Polytechnique de Grenoble in 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 90 papers published in international journals, and 110 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a senior member of the IEEE and the IEEE Computer Society, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*.