Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 65 (2005) 1497–1514
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms

A. Legrand, L. Marchal, Y. Robert*

LIP, UMR CNRS-INRIA 5668, ENS Lyon, France

Received 10 September 2003; received in revised form 2 November 2004; accepted 31 May 2005
Available online 15 July 2005

Abstract

In this paper, we consider the communications involved by the execution of a complex application, deployed on a heterogeneous large-scale distributed platform. Such applications intensively use collective macro-communication schemes, such as scatters, personalized all-to-alls or gather/reduce operations. Rather than aiming at minimizing the execution time of a single macro-communication, we focus on the steady-state operation. We assume that there is a large number of macro-communications to perform in pipeline fashion, and we aim at maximizing the throughput, i.e., the (rational) number of macro-communications which can be initiated every time-step. We target heterogeneous platforms, modeled by a graph where resources have different communication and computation speeds. The situation is simpler for series of scatters or personalized all-to-alls than for series of reduces operations, because of the possibility of combining various partial reductions of the local values, and of interleaving computations with communications. In all cases, we show how to determine the optimal throughput, and how to exhibit a concrete periodic schedule that achieves this throughput.

Keywords: Scheduling; Cluster; Heterogeneity; Scatter; Reduce; Macro-communication; Steady-state mode

1. Introduction

In this paper, we consider the communications involved by the execution of a complex application, deployed on a heterogeneous "grid" platform. Such applications intensively use macro-communication schemes, such as broadcasts, scatters, all-to-all or reduce operations.

These macro-communication schemes have often been studied with the goal of minimizing their *makespan*, i.e., the time elapsed between the emission of the first message by the source, and the last reception. But in many cases, the application has to perform a large number of instances of the same operation (for example if data parallelism is used). When dealing with such a series of macro-communications, pipelining is mandatory to achieve good performance. The relevant objective becomes the optimization of the throughput, i.e., the average number of macro-communications executed per time-unit. In this paper, we focus on scatter and reduce operations (note that broadcasts are dealt with in the companion paper [6]). Here are the definitions of these operations:

Scatter: One processor P_{source} sends a distinct message to each processor belonging to the set $\mathcal{S} = \{P_{t_0}, P_{t_1}, \ldots, P_{t_N}\}$. The set $\mathcal{S}$ may well be a strict subset of all the processors composing the platform; the remaining processors may be involved by forwarding some messages, or may be not involved at all.

Series of scatters: The same source processor performs a series of SCATTER operations, i.e., consecutively sends a large number of different messages to the set $\mathcal{S}$ of target processors.

Reduce: Each processor P_i belonging to a set $\mathcal{R} = \{P_{r_0}, P_{r_1}, \ldots, P_{r_N}\}$ of participating processors has a local value v_i. The goal is to calculate $v = v_0 \oplus \cdots \oplus v_N$, where $\oplus$ is an associative operator. Note that $\oplus$ is not necessarily

* Corresponding author. Fax: +33 4 72 72 80 80.
E-mail addresses: arnaud.legrand@ens-lyon.fr (A. Legrand), loris.marchal@ens-lyon.fr (L. Marchal), Yves.Robert@ens-lyon.fr (Y. Robert).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.05.021

commutative.[1] The result v is to be stored on processor P_{target}. Just as before, the set $\mathcal{R}$ may well be a strict subset of all the processors composing the platform;

Series of reduces: A series of REDUCE operations is to be performed, from the same set $\mathcal{R}$ of participating processors, and to the same target P_{target}.

The efficient implementation of series of macro-communications primitives is of great practical importance. Consider for instance the class of problems that are addressed by collaborative computing efforts such as SETI@home [34], factoring large numbers [14], the Mersenne prime search [31], and those distributed computing projects organized by companies such as Entropia [16]: for these problems, the execution usually begins with the distribution of a large number of input files, which correspond to the input data needed by each participating computing resource. This distribution is nothing else than a SERIES OF SCATTERS operation. Similarly, SERIES OF REDUCES operations frequently occur in linear algebra (e.g., a conjugate-gradient algorithm involving scalar products at each step), in combinatorial optimization (e.g., to determine current maxima or minima in branch-and-bound executions), and to sort or gather data in a particular order: we refer to [32] for a detailed list of applications.

As already mentioned, for the SCATTER and REDUCE problems, the goal is to minimize the makespan of the operation. For the SERIES version of these problems, the goal is to pipeline the different scatter/reduce operations so as to reach the best possible throughput in steady-state operation. In this paper, we propose a new algorithmic strategy to solve this problem. The main idea is the same for the SERIES OF SCATTERS and SERIES OF REDUCES problems, even though the latter turns out to be more difficult, because of the possibility of combining various partial reductions of the local values, and of interleaving computations with communications. But in both cases, we succeed in deriving a schedule that optimizes the throughput, and this derivation is obtained with a complexity polynomial in the platform size, independently of the number of macro-communications to be executed.

The rest of the paper is organized as follows. Section 2 describes the model used for the target computing platform model, and states the one-port assumptions for the operation mode of the resources. Section 3 deals with the SERIES OF SCATTERS problem. Section 3.5 is devoted to the extension to personalized all-to-all operations. The more complex SERIES OF REDUCES problem is described in Section 4. Section 5 presents some experimental results. Section 6 gives an overview of related work. Finally, we provide some concluding remarks in Section 7.

[1] When the operator is commutative, there is more freedom to assemble the final result. Of course it is always possible to perform the reduction with a commutative operator, but without taking advantage of the commutativity. The MPI_Reduce function [35] includes predefined operations which are commutative, but allows for user-defined operations that are not.

2. Framework

In this section we introduce the platform model. Then, we give some background on steady-state scheduling techniques. Finally, we bring in some notations that will be used throughout the text.

2.1. Platform model

We adopt a model of heterogeneity close to the one developed by Bhat et al. [10]. The network is represented by an edge-weighted graph $G = (V, E, c)$. This graph may well include cycles and multiple paths. Each edge e is labeled with the value $c(e)$, the time needed to transfer of a message of unit size through the edge.

Among different scenarios found in the literature (see Section 6), we adopt the widely used (and realistic) one-port model: at each time-step, a processor is able to perform at most one emission and one reception. When computation is taken into account, we adopt a full overlap assumption: a processor can perform computations and (independent) communications simultaneously.

To state the model more precisely, suppose that processor P_i starts sending a message of length δ at time t. This transfer will last $\delta \times c(i, j)$ time-steps. Note that the graph is directed, so there is no reason to have $c(i, j) = c(j, i)$ (and even more, the existence of edge (i, j) does not imply that of link (j, i)). The one-port model imposes that between time-steps t and $t + \delta \times c(i, j)$:

- processor P_i cannot initiate another send operation (but it can perform a receive operation and an independent computation),
- processor P_j cannot initiate another receive operation (but it can perform a send operation and an independent computation),
- processor P_j cannot start the execution of tasks depending on the message being transferred.

2.2. Steady-state scheduling

Rather than minimizing the total execution time (i.e., the "makespan"), our goal is to maximize the throughput in steady-state mode, i.e., the number of SCATTER or REDUCE operations initiated per time-unit. This approach has been pioneered by Bertsimas and Gamarnik [8].

There are three main reasons for focusing on the steady-state operation. First is *simplicity*, as the steady-state scheduling is really a relaxation of the makespan minimization problem. Initialization and clean-up phases are ignored, and the emphasis is on the design of a *periodic* schedule. Precise ordering and allocation of tasks and messages are

not required. The key idea is to characterize the activity of each resource during each time-unit. For the SERIES OF SCATTERS problem, one only needs to determine which fraction of time each processor spends communicating with which neighbor. In addition, for the SERIES OF REDUCES problem, one needs to determine which fraction of time each processor spends computing which reduction. The actual schedule then arises "naturally" from these quantities, as explained in Section 3.3.

Second is *efficiency*, as steady-state scheduling provides, by definition, a periodic schedule, which is described in compact form and is thus possible to implement efficiently in practice. Third is *adaptability*: because the schedule is periodic, it is possible to dynamically record the observed performance during the current period, and to inject this information into the algorithm that will compute the optimal schedule for the next period. This makes it possible to react on the fly to resource availability variations.

In the following, we express both optimization problems (SERIES OF SCATTERS and SERIES OF REDUCES) as a set of linear constraints, in order to build a linear program. We solve the linear program in rational numbers, with standard tools like `lpsolve` [7] or Maple [13]. Once each activity variable has been computed, the final periodic schedule can be constructed owing to an automated procedure, systematic although technical. The key idea is to scale the rational values output by the solution of the linear program, to obtain integer numbers, and the length of the period of the schedule is determined by this scaling. Once we have the activity pattern during a period, we basically reproduce it to derive the final schedule. We outline the actual construction step-by-step in Section 3.3.

2.3. Common notations

A few variables and constraints are common to the SERIES OF SCATTERS and SERIES OF REDUCES problems, because they arise from the one-port model. Let $s(P_i \to P_j)$ be the fraction of time spent by processor P_i to send messages to P_j during one time-unit. This quantity is a rational number between 0 and 1

$$\forall P_i, \forall P_j, \quad 0 \leqslant s(P_i \to P_j) \leqslant 1. \tag{1}$$

When we express the communications of a given processor P_i during a time-unit, the one-port model imposes the following constraints:

$$\forall\, P_i, \quad \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1 \quad \text{(outgoing messages from } P_i\text{)}, \tag{2}$$

$$\forall\, P_i, \quad \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1 \quad \text{(incoming messages to } P_i\text{)}. \tag{3}$$

With steady-state scheduling we do not need to determine the precise ordering in which the different communications are executed by P_i: instead we take a macroscopic point of view and simply bound the total amount of data sent and received every time-unit.

We will later add further constraints corresponding to each specific problem under study. We first explain how to use the steady-state framework on the simpler SERIES OF SCATTERS problem.

3. Series of scatters

Recall that a scatter operation involves a source processor P_{source} and a set of target processors $\mathcal{S}$; the source processor has a message m_k, of size δ_k, to send to each processor $P_k \in \mathcal{S}$. We focus here on the pipelined version of this problem: processor P_{source} aims at sending a large number of different same-size messages to each target processor $P_k \in \mathcal{S}$.

3.1. Linear program

First, we introduce a few definitions for the steady-state operation

- m_k is the type of the messages whose destination is processor P_k. The size of such messages is δ_k,
- $send(P_i \to P_j, m_k)$ is the fractional number of messages of type m_k which are sent on the edge (i, j) within a time-unit.

The relation between $send(P_i \to P_j, m_k)$ and $s(P_i \to P_j)$ is expressed by the following equation:

$$\forall\, P_i, P_j, \quad s(P_i \to P_j) = \sum_k send(P_i \to P_j, m_k) \cdot \delta_k \cdot c(i, j), \tag{4}$$

where the sum is over indices k such that $P_k \in \mathcal{S}$.

The fact that some packets are forwarded by a node P_i can be seen as a "conservation law": all the packets reaching a node which is not their final destination are transferred to other nodes. For example, in Fig. 1, node P_i receives 7 messages for P_k, and forwards them all to other processors.

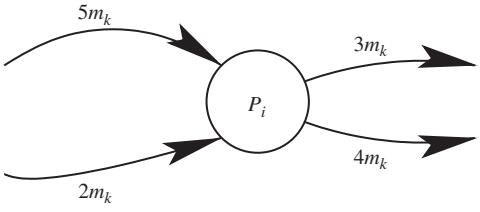


Fig. 1. Example of the conservation law, with $P_i \neq P_k$.

This idea is expressed by the following constraint:

$$\forall P_i, i \neq \text{source} \quad \forall m_k, k \neq i, \sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_k) = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_k). \tag{5}$$

As stated in Eq. (5), the conservation law does not apply to the source processor P_{source}, from which all messages originate.

Moreover, the throughput at processor P_k is the number of messages m_k received at P_k every time-unit. We have to compute the sum of all messages of type m_k received by P_k via all its incoming edges. We also have to impose that the same throughput TP is achieved at each target node. This leads to the following constraint:

$$\forall P_k \in \mathcal{S}, \quad \sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = TP. \tag{6}$$

We summarize all previous constraints in the following linear program:

STEADY-STATE SCATTER PROBLEM ON A GRAPH $SSSP(G)$

Maximize TP,

subject to

$\forall P_i, \forall P_j$, $0 \leqslant s(P_i \to P_j) \leqslant 1$

$\forall P_i$, $\sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1$

$\forall P_i$, $\sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1$

$\forall P_i, P_j$, $s(P_i \to P_j) = \sum_k send(P_i \to P_j, m_k) \cdot \delta_k \cdot c(i,j)$

$\forall P_i, \forall m_k, k \neq i$, $\sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_k) = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_k)$

$\forall P_k, k \in T$ $\sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = TP$.

This linear program can be solved in polynomial time by using tools like `lpsolve` [7], Maple [13] or MuPaD [36]. The solution will be a set $\mathcal{A}$ of rational numbers characterizing the activity on each edge. Before explaining how to construct an actual schedule from the values in $\mathcal{A}$, we deal with a little example.

3.2. Toy example

To illustrate the use of the linear program, consider the simple example described on Fig. 2. Fig. 2(a) presents the topology of the network, where each edge e is labeled with its communication cost $c(e)$. In this simple case, one source P_s sends a series of messages to two target processors P_0 and P_1. We let $\delta_0 = \delta_1 = 1$.

Figs. 2(b) and (c) show the values output by the linear program. We have multiplied all values by a factor 12 to obtain integer numbers. In Fig. 2(b), we represent the number of messages of each type going through the network, whereas Fig. 2(c) describes the occupation of each edge. The

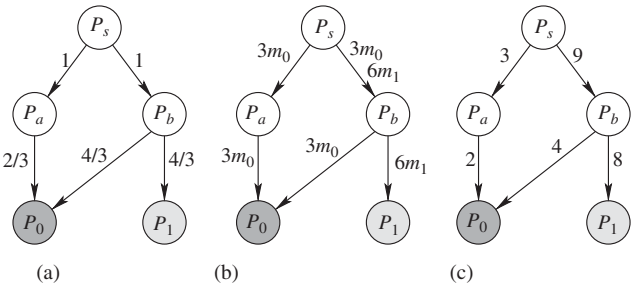


Fig. 2. Toy example for the SERIES OF SCATTERS problem. All values are given for a period of 12. The achieved throughput is 6 messages every 12 time-units: (a) topology, (b) *send* values (times 12) and (c) s values (times 12).

throughput achieved with this solution is $TP = \frac{1}{2}$, which means that one scatter operation is executed every two time-units. We point out that all the messages destined to processor P_0 do not take the same route: some are transferred by P_a, and others by P_b. Our framework allows for using multiple routes in order to reach the best throughput.

3.3. Constructing a schedule

A solution $\mathcal{A}$ to the linear program $SSSP(G)$ is a set of values $send(P_i \to P_k, m_k)$ and $s(P_i \to P_j)$. From this set, one needs to construct a (periodic) schedule, that is a way to decide which specific messages are transmitted by each edge during each period. We start by expressing all the rational numbers in $\mathcal{A}$ as the quotient of two relatively prime integers, and the period T of the schedule is set to the least common multiple of all denominators. Formally, we write $send(P_i \to P_k, m_k) = \frac{u_{i,j,k}}{v_{i,j,k}}$, $s(P_i \to P_j) = \frac{x_{i,j}}{y_{i,j}}$, and we let $T = lcm(lcm_{i,j,k}(v_{i,j,k}), lcm_{i,j}(y_{i,j}))$ (where lcm denotes the least common multiple).

Once we have the period T, we know the number of messages of each type that circulates along each edge during each period. Obviously, the first and last periods will be different, but let us concentrate on steady-state operation for the time being. We have to orchestrate all the required communications so that one-port constraints are always satisfied. This is not directly enforced by the linear program. Indeed, we know from the linear program that *locally*, each processor needs no more than T units of time to send all the messages that it is scheduled to send to all its neighbors. However, from a *global* point of view, several communications will have to be scheduled simultaneously on the platform. In the one-port model, two communications can be scheduled concurrently only if they involve edges with different sources and different targets. In other words neither two emissions nor two receptions should ever overlap on one node (but simultaneously sending and receiving is allowed). The global orchestration of the communications is achieved using a weighted-matching algorithm, as detailed in [5]. We recall the basic principles of this algorithm. From our platform graph G, and the result of the linear program,

we build a bipartite graph $G_B = (V_B, E_B, e_B)$ as follows:

- for each node P_i in G, create two nodes P_i^{send} and P_i^{recv}, one in charge of emissions, the other of receptions.
- for each non-zero value $send(P_i \rightarrow P_j, m_k)$, insert an edge between P_i^{send} and P_j^{recv} labeled with the time needed by corresponding transfers during a period, namely

$$send(P_i \rightarrow P_j, m_k) \cdot \delta_k \cdot c(i, j) \cdot T.$$

We are looking for a decomposition of this graph into a set of subgraphs where a node (sender or receiver) is occupied by at most one communication task. This means that at most one edge reaches each node in the subgraph. In other words, only communications corresponding to a matching in the bipartite graph can be performed simultaneously, and the desired decomposition of the graph is in fact an edge coloring. The weighted edge coloring algorithm of Schrijver [33, vol. A, Chapter 20] provides in time $\mathcal{O}(|E|)$ a polynomial number of matchings, which we use to perform the different communications.

Rather than going into technical details, we illustrate this algorithm on the previous example. The bipartite graph constructed when multiplying by T the $send(P_i \rightarrow P_j, m_k)$ and $s(P_i \rightarrow P_j)$ values returned by the linear program is represented on Fig. 3(a). It can be decomposed into four matchings, represented on Figs. 3(b)–(e).

These matchings explain how to split the communications to build a schedule. Such a schedule is described on Fig. 4(a). We assume that the transfer of a message can be split into several parts (for example, the fifth message transferred from P_b to P_1 is sent during the first and the third part of the period, corresponding to the first and third matchings. If needed, we can avoid splitting the transfer of a message by multiplying again by the least common multiple of all denominators appearing in the number of messages to be sent in the different matchings. In our example, since this least common multiple is 4, this produces a schedule of period 48, represented on Fig. 4(a).

The final period is potentially very large, but we discuss in Section 4.6 how to approximate the result for a smaller period.

There remains to discuss the initialization and clean-up phases. For the initialization, the simplest solution is to let the source processor P_{source} sequentially send to each processor P_i the total number of messages of each type that it receives during a period. This quantity is easy to compute from the solution $\mathcal{A}$ of the linear program: indeed, P_i receives $\sum_{P_j} send(P_j \rightarrow P_i, m_k)$ messages of type m_k. Of course such an initialization process takes longer than needed and could be easily improved. However, the key-point is that the total time I needed by P_{source} to send all these messages is bounded by a constant that depends on the platform and on T, but not on the total number of macro-communications to be performed. This simple observation will be important to prove the asymptotic optimality of the final schedule in Section 3.4. For the clean-up phase, we adopt a similar (slow) method and sequantially route the last messages circulating along the platform edges, again in constant time.

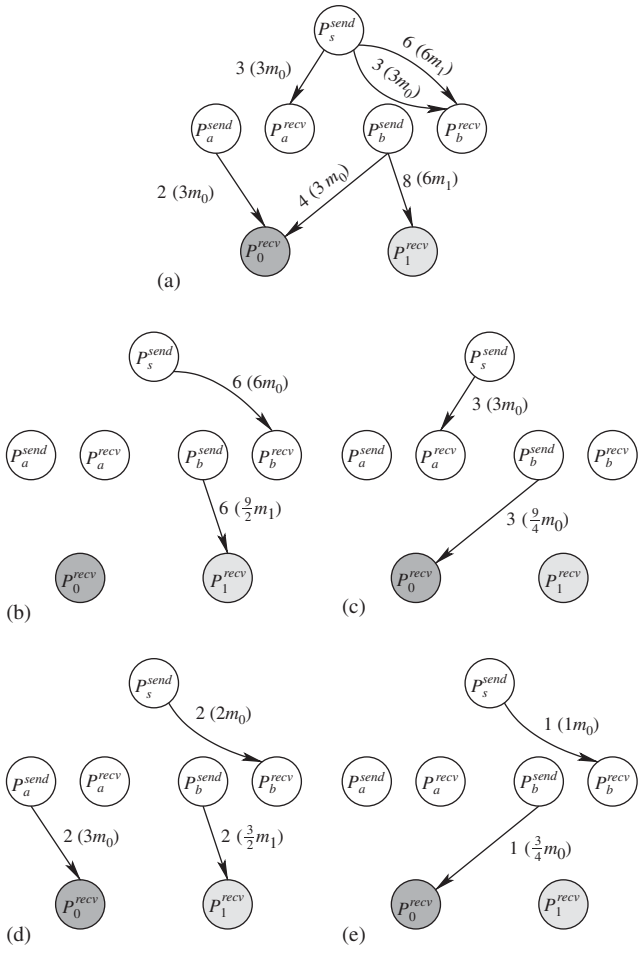


Fig. 3. Bipartite Graph of the example and its decomposition into matchings. Edges are labeled with the communication times for each type of message going through the edge. The corresponding number of messages is mentioned between brackets: (a) Bipartite Graph, (b) Matching 1, (c) Matching 2, (d) Matching 3 and (e) Matching 4.

Before moving to the proof of asymptotic optimality, we point out a technical but important point. Because the value of T arises from the linear program, $\log T$ is indeed a number polynomial in the problem size, but T itself is not. The description of what happens at every time-step during a time period would have a size exponential in the problem size. Fortunately, our description of the final schedule is expressed in a "compact" way: there is a polynomial number of intervals (corresponding to the matchings), and for each interval a polynomial (even linear) description of the activity of each resource. This also applies to the sequential initialization and clean-up phases, so altogether we have provided a polynomial characterization of the entire schedule (see [5] for further details).

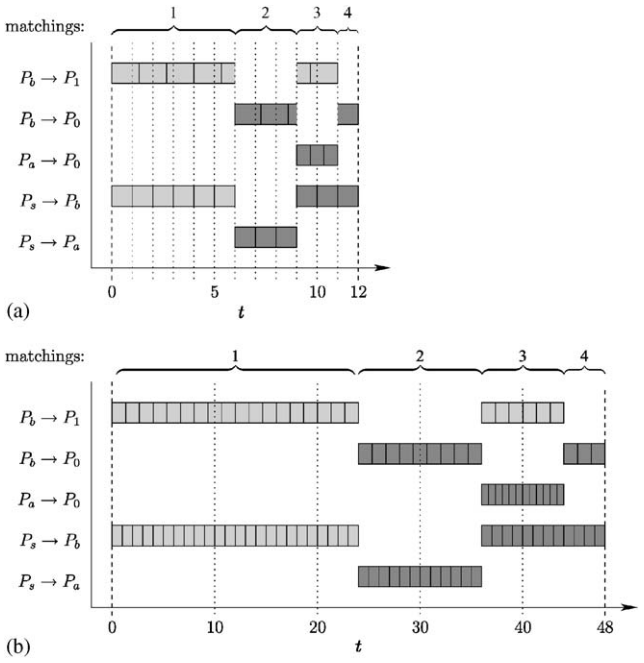


Fig. 4. Different possible schedules for the example: (a) schedule if we allow for splitting messages (period=12) and (b) schedule without any split message (period=48).

3.4. Asymptotic optimality

In this section, we prove that the previous periodic schedule is asymptotically optimal: basically, no scheduling algorithm (even non-periodic) can execute more scatter operations in a given time-frame than ours, up to a constant number of operations. This section is devoted to the formal statement of this result, and to the corresponding proof.

Given a platform graph $G = (P, E, c)$, a source processor P_{source} holding an infinite number of messages destined to a set $\mathcal{S}$ of target processors, 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 scatter operations, within K time-units. Let $TP(G)$ be the solution of the linear program $SSSP(G)$ of Section 3.1 applied to this platform graph G. We have the following result:

Lemma 1. $opt(G, K) \leqslant TP(G) \times K$.

Proof. Consider an optimal schedule (not necessarily periodic), such that the number of messages sent by the source processor within the K time-units is maximal. For each edge (P_i, P_j), let $N(P_i \to P_j, m_k)$ be the number of messages for P_k sent by P_i to P_j. Let $S(P_i \to P_j)$ be the total occupation time of the edge (P_i, P_j). Then the following equations hold true:

- $\forall P_i, \forall P_j, \quad S(P_i \to P_j) = \sum_{m_k} N(P_i \to P_j, m_k) \cdot \delta_k \cdot c(i, j)$ (by definition of $S(P_i \to P_j)$).
- $\forall P_i, \ \forall P_j, \quad 0 \leqslant S(P_i \to P_j) \leqslant K$ (schedule executed within K time-units).
- $\forall P_i, \quad \sum_{P_j,(i,j)\in E} S(P_i \to P_j) \leqslant K$ (time for P_i to send messages in the one-port model).
- $\forall P_i, \quad \sum_{P_j,(j,i)\in E} S(P_j \to P_i) \leqslant 1$ (time for P_i to receive messages in the one-port model).
- $\forall P_i, i \neq \text{source}, \forall m_k, k \neq i, \quad \sum_{P_j,(j,i)\in E} N(P_j \to P_i, m_k) = \sum_{P_j,(i,j)\in E} N(P_i \to P_j, m_k)$ (conservation law for messages forwarded by P_i to P_k).
- $\forall P_k \in \mathcal{S}, opt(G, K) = \sum_{P_j,(j,k)\in E} N(P_j \to P_k, m_k)$ (same number of messages received on each target node).

Let $send(P_i \to P_j, m_k) = \frac{N(P_i \to P_j, m_k)}{K}$ and $s(P_i \to P_j) = \frac{S(P_i \to P_j)}{K}$. All the equations of the linear program $SSSP(G)$ hold, hence $\frac{opt(G,K)}{K} \leqslant TP(G)$, since TP is the optimal value. □

Again, this lemma states that no schedule can send more messages that the steady-state. There remains to bound the loss due to the initialization and the clean-up phase in our periodic solution, in order to come up with a well-defined scheduling algorithm based upon steady-state operation. Consider the following algorithm (assume that K is large enough):

- Solve the linear program for $SSSP(G)$, compute the throughput $TP(G)$. Determine the period T such that every communication time is an integer.
- *Initialization phase*: P_{source} sequentially sends $\sum_{P_j} send(P_j \to P_i, m_k)$ messages of type m_k to each processor P_i. This requires a total time I, which is independent of K.
- Let J be the sequential time needed for each processor P_i to send $\sum_{P_j} send(P_i \to P_j, m_k)$ messages of type m_k to each target processor P_k. Again, J is a constant independent of K.
- Let $r = \lfloor \frac{K-I-J}{T} \rfloor$.
- *Steady-state scheduling*: during r periods of time T, operate the platform in steady state, according to the solution of $SSSP(G)$.
- *Clean-up*: sequentially circulate all messages to their final destination. This requires at most J time-units. Do nothing during the very last units ($K - I - J$ may not be evenly divisible by T).
- The number of messages sent to each node by this algorithm within K time-units is at least the number of messages sent during the steady-state phase, which is $steady(G, K) = r \times T \times TP(G)$.

Proposition 1. *The previous scheduling algorithm based on the steady-state operation is asymptotically optimal*:

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

Proof. Using the previous lemma, $opt(G, K) \leqslant TP(G) \times K$. From the description of the algorithm, we have $steady(G, K) = r \times T \times TP(G) = \left\lfloor \frac{K-I-J}{T} \right\rfloor \cdot T \cdot TP(G)$. Since $TP(G)$, I and T are constants independent of K, the result holds. □

This concludes the proof of the main result of this section:

Theorem 1. *The previous scheduling algorithm based on the steady-state operation provides a polynomial solution to the* SERIES OF SCATTERS *problem which is asymptotically optimal, among all possible schedules (not necessarily periodic).*

3.5. Extension to personalized all-to-alls

We have dealt with the SERIES OF SCATTERS problem, but the same equations can be used in the more general case of a SERIES OF PERSONALIZED ALL-TO-ALLS. In this context, a set of source processors sends a series of distinct messages to a set of target processors. The messages are now typed with the source and the destination processors: $m_{k,l}$ is a message emitted by P_k and destined to P_l; its size is $\delta_{k,l}$. The constraints stand for the one-port model, and for conservation of the messages. The throughput has to be the same for each sender, and for each target node. This leads to the linear program summarizing all these constraints:

STEADY-STATE PERSONALIZED ALL-TO-ALL PROBLEM ON A GRAPH *SSPA2A*(G)

Maximize TP,
subject to

$\forall P_i, \forall P_j,$	$0 \leqslant s(P_i \to P_j) \leqslant 1$
$\forall P_i,$	$\sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1$
$\forall P_i,$	$\sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1$
$\forall P_i, \forall P_j,$	$s(P_i \to P_j) = \sum_{k,l} send(P_i \to P_j, m_{k,l}) \cdot \delta_{k,l} \cdot c(i,j)$
$\forall P_i,$ $\forall m_{k,l}, k \neq i, l \neq i,$	$\sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_{k,l}) = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_{k,l})$
$\forall P_k, \forall m_{k,l}$	$\sum_{P_i,(k,i)\in E} send(P_k \to P_i, m_{k,l}) = TP$
$\forall P_l, \forall m_{k,l}$	$\sum_{P_i,(i,l)\in E} send(P_i \to P_l, m_{k,l}) = TP.$

After solving this linear system, we proceed exactly as for the SERIES OF SCATTERS. We compute the period of the schedule as the least common multiple of all denominators in the solution, and we build a valid schedule using the weighted-matching algorithm. Furthermore, we can prove the same result of asymptotic optimality:

Theorem 2. *The previous scheduling algorithm based on the steady-state operation provides a polynomial solution to the* SERIES OF PERSONALIZED ALL-TO-ALLS *problem which is asymptotically optimal, among all possible schedules (not necessarily periodic).*

4. Series of reduces

After introducing our framework on the (relatively) simple SERIES OF SCATTERS operation, we move on to a more complex collective communication primitive, the SERIES OF REDUCES operation, which involves both communications and computations. Achieving good performances is much more challenging (but more interesting) for the SERIES OF REDUCES problem. We focus our experiments on this problem (see Section 5).

We recall the sketch of a reduce operation: each processor P_{r_i} in the set $\mathcal{R} = \{P_{r_0}, \ldots, P_{r_N}\}$ initially owns a value v_i. The goal is to compute the reduction v of these values: $v = v_0 \oplus \cdots \oplus v_N$, where $\oplus$ is an associative, but not necessarily commutative operator. We impose that at the end, the result is stored in processor P_{target}. The reduce operation is more complex than the scatter operation, because computational tasks are inserted, in order to merge the different messages into new "reduced" ones. For $0 \leqslant k \leqslant m \leqslant N$, let $v_{[k,m]}$ denote the partial result corresponding to the reduction of the values $v_k, \ldots, v_m$:

$$v_{[k,m]} = v_k \oplus \cdots \oplus v_m.$$

The initial values $v_i = v_{[i,i]}$ will be reduced into partial results until the final result $v = v_{[0,N]}$ is reached. As $\oplus$ is associative, two partial results can be reduced as follows:

$$v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}.$$

We let $T_{k,l,m}$ denote the computational task associated to this reduction.

We start by giving an example of a non-pipelined reduce operation, in order to illustrate how to interpret this operation as a reduction tree. Next, we move to the SERIES OF REDUCES problem: we explain how to derive the linear program, and how to build a schedule using the result of this linear program.

4.1. Introduction to reduction trees

Consider the simple example of a network composed of three processors P_0, P_1, P_2 owning the values v_0, v_1, v_2, and linked by a fully connected topology (see Fig. 6(a)). The target processor is P_0. One way to perform the reduction of $\{v_0, v_1, v_2\}$ is the following schedule:

1. P_2 sends its value v_2 to P_1,
2. P_1 computes the partial reduction $v_{[1,2]} = v_1 \oplus v_2$ (task $T_{1,1,2}$),
3. P_0 sends its value v_0 to P_1,
4. P_1 computes the final result $v_{[0,2]} = v_0 \oplus v_{[1,2]}$ (task $T_{0,0,2}$),
5. P_1 sends the final result $v = v_{[0,2]}$ to P_0.

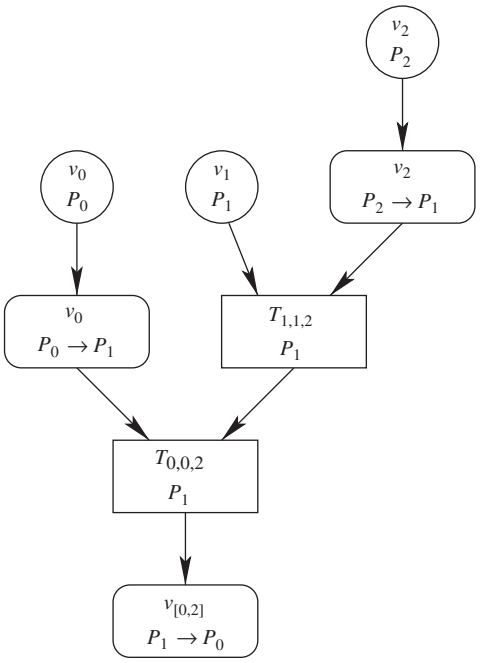


Fig. 5. Simple example of a reduction tree.

Obviously, this may well not be the shortest way to perform the reduction! But we merely use the above schedule to introduce reduction trees. Indeed, we represent the schedule by a tree. We create one node for each value v_i on processor P_i, and for each task (either a communication or a computation). We insert one edge $n_1 \to n_2$ when the result of node n_1 is an input data of node n_2. The reduction tree of the schedule described above is represented on Fig. 5.

A schedule for a single reduction operation uses a single reduction tree. As we are interested in the SERIES OF REDUCES problem, we assume that each processor P_i initially has a set of values, indexed with a time-stamp: one of these values is denoted as v_i^t. The series of reductions consists in the reduction of each set $\{v_0^t, \ldots, v_N^t\}$ for each time-stamp t. We can interpret each of these reductions as a reduction tree, but two different reductions (for distinct time-stamps t_1 and t_2) may well use two different reduction trees.

4.2. Linear program

To describe the linear constraints of the SERIES OF REDUCES problem, we use the following variables:

- $send(P_i \to P_j, v_{[k,l]})$ is the fractional number of messages of type $v_{[k,l]}$ and which are sent from P_i to P_j, within one time unit,
- $cons(P_i, T_{k,l,m})$ is the fractional number of tasks $T_{k,l,m}$ computed on processor P_i within one time unit. Note that $cons(P_i, T_{k,l,m})$ also is the number of messages of type $[k, l]$ and $[l + 1, m]$ consumed by P_i for this operation; similarly, $cons(P_i, T_{k,l,m})$ is the number of new messages of type $[k, m]$ generated by P_i during this operation.
- $\alpha(P_i)$ is the time spent by P_i computing all tasks within each time-unit. This quantity is obviously bounded:

$$\forall P_i, \quad 0 \leqslant \alpha(P_i) \leqslant 1. \tag{7}$$

- $size(v_{[k,l]})$ is the size of one message containing a value $v_{[k,l]}$.
- $w(P_i, T_{k,l,m})$ is the time needed by processor P_i to compute one task $T_{k,l,m}$.

The number of messages sent on edge (i, j) is related to the communication time on this edge

$$\forall P_i, P_j, \quad s(P_i \to P_j) = \sum_{v_{[k,l]}} send(P_i \to P_j, v_{[k,l]}) \cdot size(v_{[k,l]}) \cdot c(i, j). \tag{8}$$

In the same way, the number of tasks computed by P_i is related to the time spent for their computation

$$\forall P_i, \quad \alpha(P_i) = \sum_{T_{k,l,m}} cons(P_i, T_{k,l,m}) \cdot w(P_i, T_{k,l,m}). \tag{9}$$

The "conservation law" is more complicated than for the SERIES OF SCATTERS problem. The following equation deals with the number of messages of type $v_{[k,m]}$ that are received and/or processed by a given processor P_i. On the left-hand side, we add all possible sources for messages of type $v_{[k,m]}$: either they are sent to P_i by neighbors, or they are generated in place by P_i from the combination of two messages of type $v_{[k,l]}$ and $v_{[l+1,m]}$ (which is a task $T_{k,l,m}$, where $k \leqslant l < m$). The right-hand side of the equation deals with how these messages are utilized. Either they are forwarded by P_i to some of its neighbors, or they are consumed in place. In the latter case, a message of type $v_{[k,m]}$ can be combined with a message of type $v_{[m+1,n]}$, (which is a task $T_{k,m,n}$ with $n > m$) or with a message of type $v_{[n,m-1]}$ (which is a task $T_{n,k-1,m}$ with $n < k$). We are led to the equation

$$\begin{aligned} &\forall P_i, \forall v_{[k,m]} \text{ such that} \\ &[k, m] \neq [i, i] \text{ or } ([k, m], P_i) \neq ([0, N], P_{\text{target}}) \\ &\sum_{P_j,(j,i)\in E} send(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} cons(P_i, T_{k,l,m}) \\ &= \sum_{P_j,(i,j)\in E} send(P_i \to P_j, v_{[k,m]}) \\ &\quad + \sum_{n>m} cons(P_i, T_{k,m,n}) + \sum_{n<k} cons(P_i, T_{n,k-1,m}). \end{aligned} \tag{10}$$

As stated in the above equation, there are two exceptions for the conservation law: (i) it is not verified for messages $v_{[i,i]}$ on processor P_i, since we assume to have an unlimited number of such messages in place; and (ii) it is not verified for the final, completely reduced message $v = v_{[0,N]}$ on the target processor P_{target}. In fact, the number of messages v reaching P_{target} is the throughput TP that we want to maximize

$$TP = \sum_{P_j,(j,\text{target})\in E} send(P_j \to P_{\text{target}}, v_{[0,N]}) + \sum_{0\leqslant l<N-1} cons(P_{\text{target}}, T_{0,l,N}). \quad (11)$$

If we summarize all these constraints, we are led to the following linear program:

STEADY-STATE REDUCE PROBLEM ON A GRAPH $SSR(G)$

Maximize TP

subject to

$\forall P_i, \forall P_j,\ 0 \leqslant s(P_i \to P_j) \leqslant 1$

$\forall P_i,\ \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1$

$\forall P_i,\ \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1$

$\forall P_i,\ 0 \leqslant \alpha(P_i) \leqslant 1$

$\forall P_i, P_j,\ s(P_i \to P_j) = \sum_{v_{[k,l]}} send(P_i \to P_j, v_{[k,l]}) \cdot size(v_{[k,l]}) \cdot c(i,j)$

$\forall P_i,\ \alpha(P_i) = \sum_{T_{k,l,m}} cons(P_i, T_{k,l,m}) \cdot w(P_i, T_{k,l,m})$

$\forall P_i, \forall v_{[k,m]}$ such that $[k,m] \neq [i,i]$ or $([k,m], P_i)$
$\neq ([0,N], P_{\text{target}})$
$\sum_{P_j,(j,i)\in E} send(P_j \to P_i, v_{[k,m]}) + \sum_{k\leqslant l<m} cons(P_i, T_{k,l,m})$
$= \sum_{P_j,(i,j)\in E} send(P_i \to P_j, v_{[k,m]}) + \sum_{n>m} cons(P_i, T_{k,m,n}) + \sum_{n<k} cons(P_i, T_{n,k-1,m})$

$\sum_{P_j,(j,\text{target})\in E} send(P_j \to P_{\text{target}}, v_{[0,N]}) + \sum_{0\leqslant l<n-1} cons(P_{\text{target}}, T_{0,l,N}) = TP.$

As we did for the SERIES OF SCATTERS problem, we solve the linear program $SSR(G)$ in rational numbers. The solution $\mathcal{A}$ is the set of the values of all the variables in the linear program. We express each of the values in $\mathcal{A}$ as the quotient of two relatively prime denominators, and we compute the period T as the least common multiple of all denominators. We let $\mathcal{A}^T : \mathcal{A} \to \mathbb{N}$ denote the function that returns the product of each variable in $\mathcal{A}$ by T, hence returning integer values. For instance, $\mathcal{A}^T(send(P_1 \to P_2, v_{[1,1]})) = 2$ means that two messages of type $v_{[1,1]}$ are sent form P_1 to P_2 every period of length T.

4.3. Constructing a schedule

Once the solution of $SSR(G)$ is computed, we have to exhibit a concrete schedule that achieves the optimal throughput. As before, the intuitive idea is to start from the description of a period and to reproduce its pattern. This is illustrated in Fig. 6 for the simple example.

Reduction trees are the tool that we use to derive a compact (polynomial) description of the final schedule. The intuitive idea is the following: for each time-stamp t, a reduction tree is used to reduce the values $v_0^t, \ldots, v_{N-1}^t$. The same tree may well be used by many time-stamps t, so we introduce the weight of a reduction tree as the number of time-stamps which use it during a period. If we used a single tree during the period, the throughput would be equal to the weight of the tree divided by T. Using several trees concurrently, the total throughput will be the optimal value TP returned by the linear program $SSR(G)$. The set of weighted trees is the compact description that we are aiming at. The reduction trees corresponding to the example of Fig. 6 are illustrated on Fig. 7.

To formally define a reduction tree, we first define a task and its inputs. First, a *task* is either a computation $T_{k,l,m}$ on node P_i (written $cons(P_i, T_{k,l,m})$) or the transfer of a message $v_{[k,m]}$ from node P_i to node P_j (written $send(P_i \to P_j, v_{[k,m]})$). An input of a task is a pair (*message*, *location*). The inputs of a computational task $cons(P_i, T_{k,l,m})$ are $(v_{[k,l]}, P_i)$ and $(v_{[l+1,m]}, P_i)$, and its result is $(v_{[k,m]}, P_i)$. The single input of a communication task $send(P_i \to P_j, v_{[k,m]})$ is $(v_{[k,m]}, P_i)$, and its result is $(v_{[k,m]}, P_j)$.

Definition 1. A reduction tree $\mathcal{T}$ is a list of tasks (computations or communications), such that an input of a task in $\mathcal{T}$ is either the result of another task in $\mathcal{T}$, or a message $v_{[i,i]}$ on processor P_{r_i}.

To a reduction tree $\mathcal{T}$, we associate the incidence function $\chi_{\mathcal{T}}$, such that

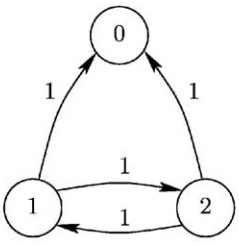
$$\forall task \in \{cons(P_i, T_{k,l,m}), send(P_i \to P_j, v_{[k,m]})\},\quad \chi_{\mathcal{T}}(task) = \begin{cases} 1 \text{ if } task \in \mathcal{T}, \\ 0 \text{ if } task \notin \mathcal{T}. \end{cases}$$

We state the following result, which states that the optimal throughput can be achieved by simultaneously using a polynomial number of weighted trees:

Lemma 2. *We can build in polynomial time a set of weighted trees* $\mathcal{W} = \{(\mathcal{T}, weight(\mathcal{T}))\}$, *with*

- $\forall \mathcal{T} \in \mathcal{W}$, $weight(\mathcal{T}) \in \mathbb{N}$,
- *the cardinal of* $\mathcal{W}$ (*denoted as* $card(\mathcal{W})$) *is polynomial in the size of the topology graph* G,
- $\sum_{\mathcal{T}\in\mathcal{W}} weight(\mathcal{T}) \cdot \chi_{\mathcal{T}} = \mathcal{A}^T$.

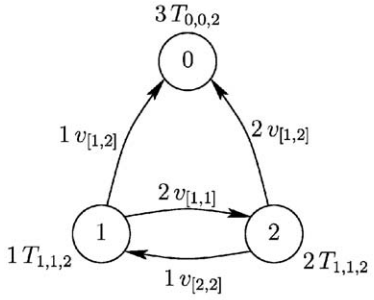
The constructive proof of this lemma will be given in Section 4.4, in the form of an algorithm to extract reduction trees from the solution $\mathcal{A}^T$. Assume for the moment that Lemma 2 is true. Using this decomposition of the solution into reduction trees, we can build a valid schedule for the SERIES OF REDUCES problem. We use the same approach as



(a) Topology

(b) Solution of linear program (period $T = 3$)

$$\mathcal{A}^{\mathcal{T}}\big(send(P_1 \to P_2, v_{[1,1]})\big) = 2$$
$$\mathcal{A}^{\mathcal{T}}\big(send(P_2 \to P_1, v_{[2,2]})\big) = 1$$
$$\mathcal{A}^{\mathcal{T}}\big(send(P_1 \to P_0, v_{[1,2]})\big) = 1$$
$$\mathcal{A}^{\mathcal{T}}\big(send(P_2 \to P_0, v_{[1,2]})\big) = 2$$
$$\mathcal{A}^{\mathcal{T}}\big(cons(P_1, T_{1,1,2})\big) = 1$$
$$\mathcal{A}^{\mathcal{T}}\big(cons(P_2, T_{1,1,2})\big) = 2$$
$$\mathcal{A}^{\mathcal{T}}\big(cons(P_0, T_{0,0,2})\big) = 3$$



(c) Results on topology

Link/Node	0	1	2	3	4	5
node 1		$T^1_{[1,1,2]}$	$T^2_{[1,1,2]}$			
$1 \to 2$	$v^0_{[1,1]}$					
$1 \to 0$			$v^1_{[1,2]}$	$v^2_{[1,2]}$		
node 2		$T^0_{[1,1,2]}$				
$2 \to 1$	$v^1_{[1,1]}$	$v^2_{[1,1]}$				
$2 \to 0$			$v^0_{[1,2]}$			
node 0				$T^0_{[0,0,2]}$	$T^1_{[0,0,2]}$	$T^2_{[0,0,2]}$

(d) Example of schedule - basic scheme

Link/Node	0	1	2	3	4	5	6	7	8	9	10	11
node 1		$T^1_{[1,1,2]}$	$T^2_{[1,1,2]}$		$T^4_{[1,1,2]}$	$T^5_{[1,1,2]}$		$T^7_{[1,1,2]}$	$T^8_{[1,1,2]}$		$T^{10}_{[1,1,2]}$	$T^{11}_{[1,1,2]}$
$1 \to 2$	$v^0_{[1,1]}$			$v^3_{[1,1]}$			$v^6_{[1,1]}$			$v^9_{[1,1]}$		
$1 \to 0$			$v^1_{[1,2]}$	$v^2_{[1,2]}$		$v^4_{[1,2]}$	$v^5_{[1,2]}$		$v^7_{[1,2]}$	$v^8_{[1,2]}$		$v^{10}_{[1,2]}$
node 2		$T^0_{[1,1,2]}$			$T^3_{[1,1,2]}$			$T^6_{[1,1,2]}$			$T^9_{[1,1,2]}$	
$2 \to 1$	$v^1_{[1,1]}$	$v^2_{[1,1]}$		$v^4_{[1,1]}$	$v^5_{[1,1]}$		$v^7_{[1,1]}$	$v^8_{[1,1]}$		$v^{10}_{[1,1]}$	$v^{11}_{[1,1]}$	
$2 \to 0$			$v^0_{[1,2]}$			$v^3_{[1,2]}$			$v^6_{[1,2]}$			$v^9_{[1,2]}$
node 0				$T^0_{[0,0,2]}$	$T^1_{[0,0,2]}$	$T^2_{[0,0,2]}$	$T^3_{[0,0,2]}$	$T^4_{[0,0,2]}$	$T^5_{[0,0,2]}$	$T^6_{[0,0,2]}$	$T^7_{[0,0,2]}$	$T^8_{[0,0,2]}$

(e) Example of schedule - pipelined

Fig. 6. Exhaustive schedule derived from the results of the linear program: (a) the topology of the network. All nodes are participating, and we let $P_{r_i} = P_i$. Initially, P_i owns the value v_i. Each edge e is labeled with its communication cost $c(e)$. Every processor can process any task in one time-unit, except node P_0 which can process any two tasks in one time-unit. The size of every message is 1. The target node is P_0. (b, c) the solution of the linear program (period $T = 3$), and the results of the linear program mapped on the topology graph and (d, e) the exhaustive description of a valid schedule using the values given in 6(c). Three reductions are performed every three time-units. The values being reduced are labeled with their time-stamp (upper index). Fig. 6(d) shows the non-pipelined schedule (for a single reduction), while Fig. 6(e) presents the pipelined version, leading to a throughput of one reduce operation per time-unit.

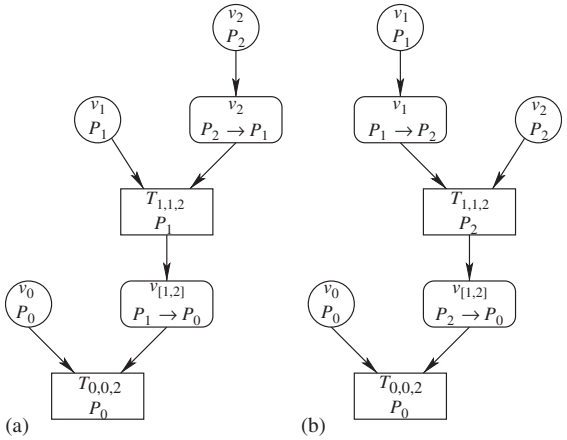


Fig. 7. Two reduction trees used in the schedule described in Fig. 6(d): (a) reduction tree T_0 (weight 1, throughput $\frac{1}{3}$) and (b) reduction tree T_1 (weight 2, throughput $\frac{2}{3}$).

for the SERIES OF SCATTERS to orchestrate all the communications induced by the different trees, namely a weighted-matching algorithm (note that all computational tasks can be executed in any order). We construct a bipartite graph $G_B = (V_B, E_B, e_B)$ as follows:

- for each processor P_i, we add two nodes to V_B: P_i^{send} and P_i^{recv},
- for each communication task $send(P_i \to P_j, v_{[k,m]})$ in each reduction tree $\mathcal{T}$, we add an edge between P_i^{send} and P_j^{recv} weighted by the time needed to perform the transfer

$$weight(\mathcal{T}) \cdot size(v_{[k,m]}) \cdot c(i, j).$$

The one-port constraints impose that the sum of the weights of edges adjacent to a processor is smaller than the period T. Using the same weighted-matching algorithm as in Sec-

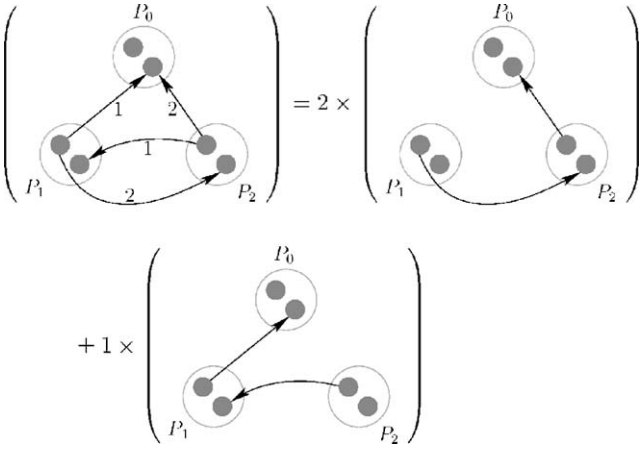
```
FIND_TREE(A)
 1: Inputs := {(v_[0,N] on node P_target)}
 2: Tasks := ∅
 3: while there is an input ∈ Inputs with input ≠ (v_[i,i] on node P_i) do
 4:    find such an input, write it as (v_[k,m] on node P_i)
 5:    if there is a index l such that A(cons(T_k,l,m, P_i)) > 0 then
          {the message v_[k,m] is computed in place}
 6:       Inputs ← Inputs − {input}
 7:       Inputs ← Inputs ∪ {(v_[k,l] on node P_i), (v_[l+1,m] on node P_i)}
 8:       Tasks ← Tasks ∪ {cons(T_[k,l,m], P_i)}
 9:       goto 3
10:    else if there is a processor P_j such that A(send(P_i → P_j, v_[k,m])) > 0
       then
          {the message v_[k,m] is received from P_j}
11:       Inputs ← Inputs − {input}
12:       Inputs ← Inputs ∪ {(v_[k,m] on node P_j)}
13:       Tasks ← Tasks ∪ {send(P_j → P_i, v_[k,m])}
14:       goto 3
15: return Tasks
```

```
EXTRACT_TREE(A)
1: Trees ← ∅
2: while Σ_{T∈Trees} weight(T) < TP do
3:    New_Tree ← FIND_TREE(A)
4:    weight(New_Tree) ← min{A(task), task ∈ New_Tree}
5:    for all task ∈ New_Tree do
6:       A(task) ← A(task) − weight(New_Tree)
7:    Trees ← Trees ∪ {(New_Tree, weight(New_Tree))}
8: return Trees
```

Fig. 8. Extracting reduction trees from a solution $\mathcal{A}$.

tion 3.3, we decompose the graph into a weighted sum of matchings such that the sum of the coefficient is less than T. As previously, this gives a schedule for achieving the throughput TP within a period T.

For the previous example, there are two reduction trees, as illustrated below



On this example, there are two steps corresponding to the two matchings. At each step, only the communications occurring for a single reduction tree take place. This is not true in the general case: each matching may well involve communications belonging to several reduction trees.

4.4. Extracting trees

We present here an algorithm to extract reduction trees from the solution $\mathcal{A}^T$. The algorithm is described on Fig. 8. It constructs a set *Trees* of reduction trees with a greedy approach: while we have not reached the throughput TP, we search for a reduction tree $\mathcal{T}$ in the remaining tasks; we weight this tree by the maximum throughput $weight(\mathcal{T})$ that it can produce, which is the minimum throughput of all tasks used in the tree. Then, we update the solution $\mathcal{A}^T$ by decreasing all tasks used in $\mathcal{T}$ by a factor $weight(\mathcal{T})$.

Proposition 2. *The algorithm* Extract_Trees($\mathcal{A}^T$) *produces a set of trees Trees such that*:

- $\mathcal{A}^T = \sum_{\mathcal{T}\in Trees} weight(\mathcal{T}) \times \chi_{\mathcal{T}}$,
- *the number of trees is polynomial in the size of the topology graph* G,
- *the complexity of the algorithm is polynomial in the size of* G.

Proof. The proof is technical but not difficult. We refer the reader to [24] for full details. □

4.5. Asymptotic optimality

We can prove the same result of asymptotic optimality as for the scatter and personalized all-to-all operations (see [24]):

Theorem 3. *The previous scheduling algorithm based on the steady-state operation provides a polynomial solution to the* SERIES OF REDUCES *problem which is asymptotically optimal, among all possible schedules (not necessarily periodic).*

4.6. Approximation for a fixed period

The framework developed here gives a schedule for a pipelined reduce problem with an integer throughput TP during a period T. However, as already pointed out, this period may be too large, from a practical viewpoint: for instance we may want to reevaluate platform parameters (CPU speeds and link bandwidths) every fixed amount of time. We propose here to approximate the solution with a periodic solution of period T_{fixed}.

Assume that we have the solution $\mathcal{A}^T$ and its decomposition into a set of weighted reduction trees $\{\mathcal{T}, weight(\mathcal{T})\}$. We compute the following values:

$$r(\mathcal{T}) = \left\lfloor \frac{weight(\mathcal{T})}{T} \times T_{\text{fixed}} \right\rfloor.$$

The one-port constraints are satisfied for $\{\mathcal{T}, weight(\mathcal{T})\}$ on a period T, so they are still satisfied for $\{\mathcal{T}, r(\mathcal{T})\}$ on a period T_{fixed}. So these new values can be used to build a valid schedule whose period is T_{fixed}.

We can bound the difference between the throughput $\frac{1}{T_{\text{fixed}}} \times \sum_{\mathcal{T}\in\text{TREES}} r(\mathcal{T})$ of the approximated solution and

the original throughput TP:

$$\begin{aligned} TP - \frac{1}{T_{\text{fixed}}} \times \sum_{\mathcal{T} \in \text{TREES}} r(\mathcal{T}) \\ = TP - \sum_{\mathcal{T} \in \text{TREES}} \frac{1}{T_{\text{fixed}}} \times \left\lfloor \frac{weight(\mathcal{T})}{T} \times T_{\text{fixed}} \right\rfloor \\ \leqslant TP - \sum_{\mathcal{T} \in \text{TREES}} \frac{1}{T_{\text{fixed}}} \times \left(\frac{weight(\mathcal{T})}{T} \times T_{\text{fixed}} - 1 \right) \\ \leqslant \frac{card(\text{TREES})}{t_{\text{fixed}}}. \end{aligned}$$

This shows that the approximated solution asymptotically approaches the best throughput as T_{fixed} grows. We have proven the following result:

Proposition 3. *We can derive a steady-state operation for periods of arbitrary length, whose throughput converges to the optimal solution as the period size increases.*

5. Experimental results

In this section we report results from the comparison of our multi-tree approach with the reduction algorithm of MPICH [17]. These results are obtained with the SimGrid simulator [23], which we briefly describe below. Beforehand, we work out an example of realistic size, to better illustrate the decomposition into several weighted trees.

5.1. Working out a realistic example

The platform used for this example is generated by Tiers, a random generator of topology [11]. The bandwidths of the links and the computing speeds of the processors are randomly chosen. The platform is represented on Fig. 9. We assume that all the $v_{[k,m]}$ have the same size (10 MB) and that all tasks need 10 Mflops to be computed on each processor. The nodes taking part to the computation are the nodes of the LAN networks generated by Tiers, they are shaded in gray on the figure. The other (white) nodes are routers.

Fig. 10 presents the results of the linear program mapped on the topology (the period is normalized to 1). The optimal throughput is $TP = \frac{2}{9}$. Two reduction trees can be extracted from these results with our algorithm, they are presented on Figs. 11(a) and (b).

5.2. Comparison with MPICH

To compare our approach for the SERIES OF REDUCES problem to existing solutions, we conduct simulations of our algorithm and of a standard MPI algorithm for the REDUCE operation. We choose to simulate the reduction algorithm of the current MPICH implementation (version 1.2.6) of MPI [17]. Although the operator of the reduction in the MPI standard may well be commutative, MPICH does not take advantage of this specification, which avoids having different results due to round-off or over/underflows errors. Thus, the MPICH algorithm can be compared fairly with our "non-commutative" reduction operation. MPICH builds a (single) binomial tree to perform a reduction operation. In the following we compare the throughput that can be achieved using the single MPICH tree with the throughput obtained using our set of weighted trees.

Using Tiers, we generate more than 200 platforms with 30 nodes. We choose among the LAN nodes some hosts that will participate to the SERIES OF REDUCES. Given a problem, consisting of a platform graph and a set of hosts, we compute the optimal throughput using the linear program of Section 4.2. Then we extract a set of weighted reduction trees which reaches this throughput with the algorithm presented in Section 4.4. We derive a distributed algorithm to perform the reduction operations, based on the local rules given by the tree decomposition. We also compute a set of local rules that allows to perform the reduction operations as MPICH does, that is using a single binomial tree. Then, the through-

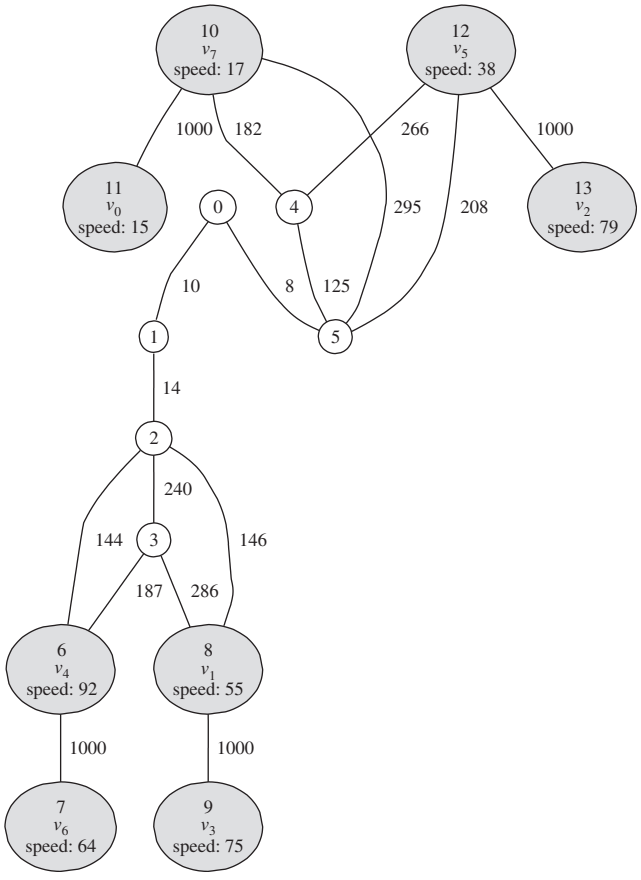


Fig. 9. A heterogeneous topology, generated by Tiers. Only gray-shaded processors have some value to be reduced, and participate in the computation. White nodes are routers. The target node is node 6. Links are labeled by bandwidth (in MB/s), and nodes are labeled by their computing speed (in Mflops/s).

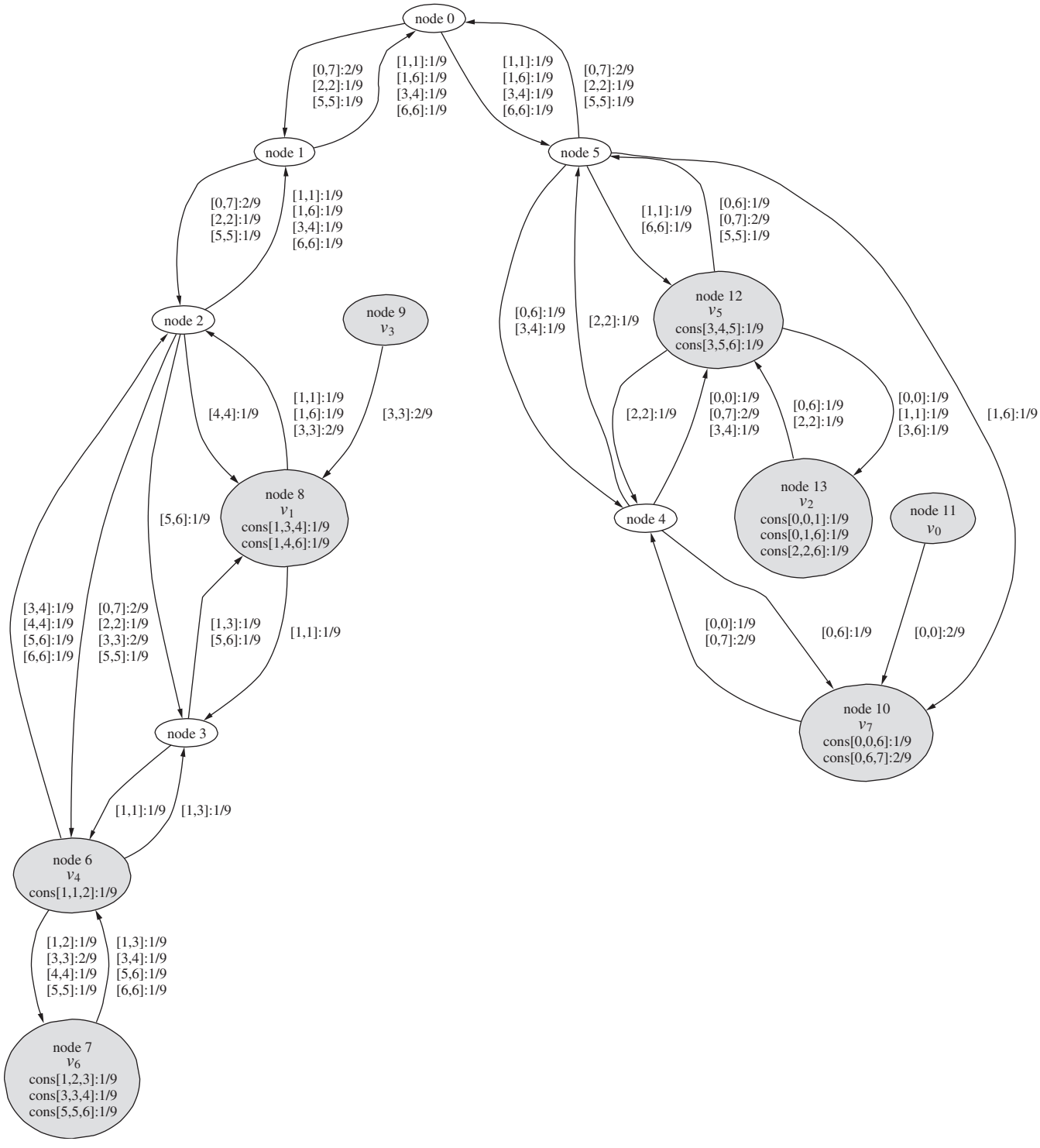


Fig. 10. Results of the linear program. The target node is node 6 with index 4. Each link is labeled with the transfers scheduled through it during one time-unit. For example, $[1, 6] : \frac{1}{9}$ means that $\frac{1}{9}$ message of type $v_{[1,6]}$ pass through the edge during one time-unit. In the same way, the computing nodes are labeled with the tasks which they execute.

put achieved by these two approaches (our "multi-tree" approach, and the MPICH single-tree approach) is computed by simulating both distributed algorithms over the SimGrid simulator [23].

The main advantage of using simulation is the ability to compare both algorithms on a wide variety of heterogeneous platforms, which would not be possible using real experiments. Indeed, experiments on distributed heteroge-

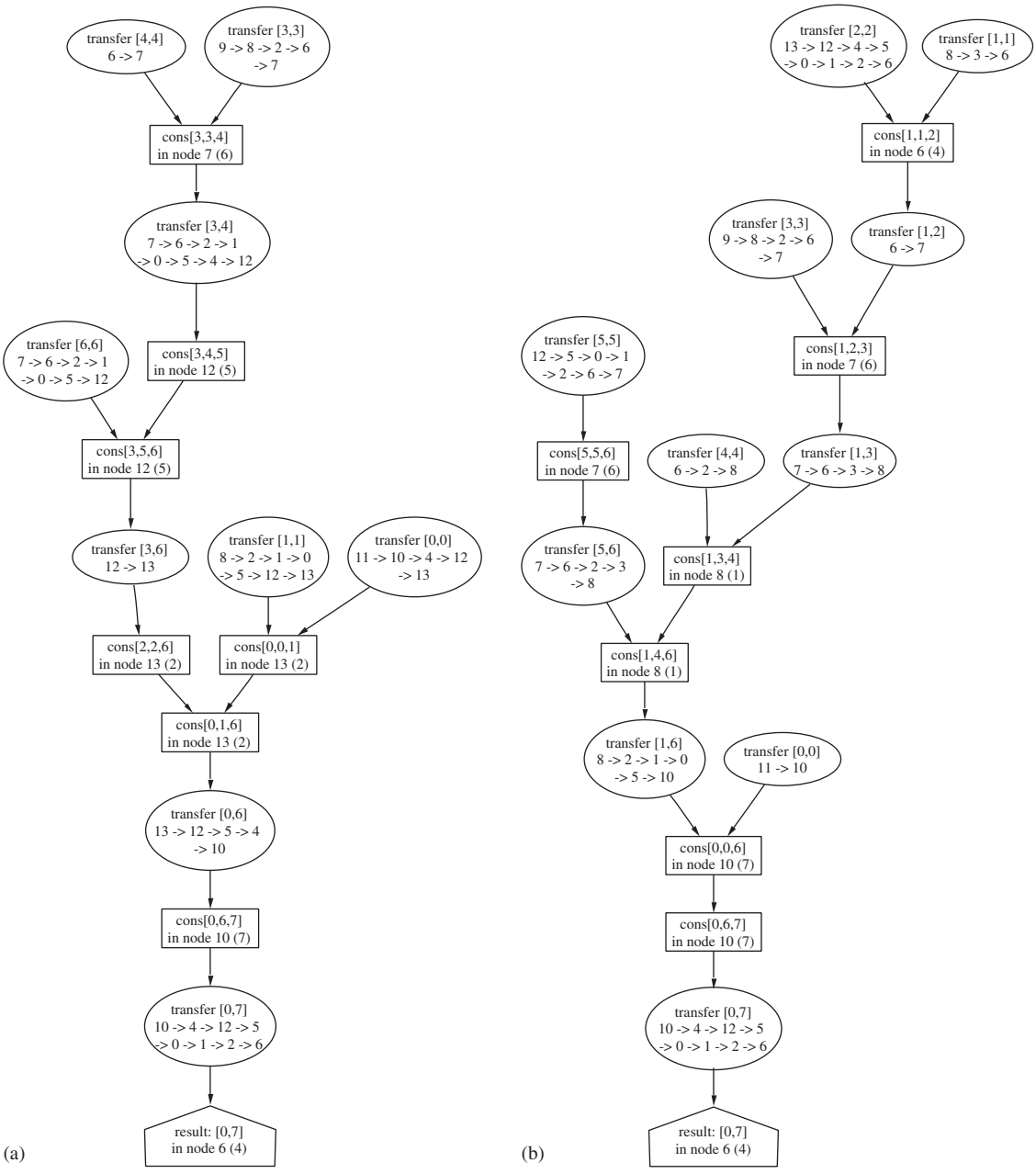


Fig. 11. The two reduction trees extracted from the solution of the linear program (Fig. 10). Each tree has a throughput of $\frac{1}{9}$: (a) first reduction tree and (b) second reduction tree.

neous platforms are technically difficult to drive, because of their genuine instability of the platform. For example, wide-area links are often shared with Internet traffic from other applications, and their performance is not as constant and reliable as the one of a dedicated cluster of workstations. In a word, it is almost impossible to guarantee that a platform which is not dedicated to the experiment, will remain exactly the same between two tests, thereby forbidding any meaningful comparison. Simulations are then used to replace real experiments, so as to ensure the reproducibility of measured data. Being faster than real experiments, simulations enable to test the algorithms in a variety of conditions. A key issue is the possibility to run the simulations against a realistic environment. SIMGRID [12,23] is an event-driven toolkit providing a set of core abstractions and functionalities that can be used to easily build simulators for specific application domains and/or computing environment topologies. Altogether, this motivated our choice of using SIMGRID for comparing our approach with MPICH.

The results of the SIMGRID simulations are presented in Fig. 12. We observed that the results are linked to the pro-

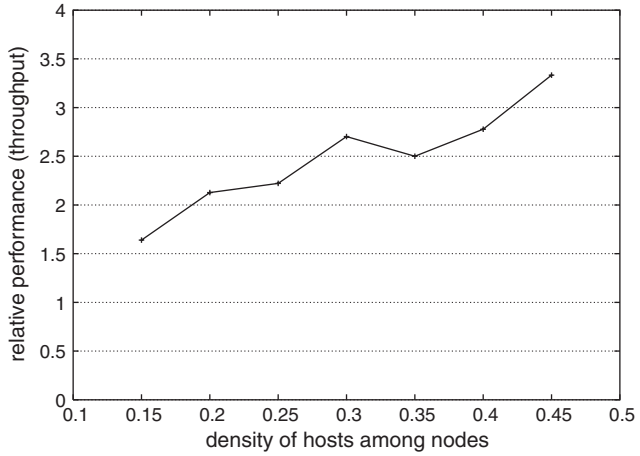


Fig. 12. Comparison of the throughput achieved by our multi-tree approach over the MPICH single-tree algorithm.

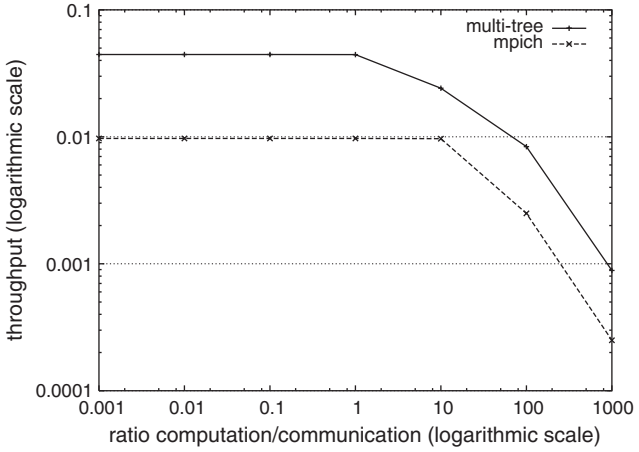


Fig. 13. Throughput achieved by our multi-tree approach and by the MPICH single-tree algorithm.

portion of participating nodes among all platform nodes, which is called *density of hosts* in the platform. These results show that our multi-tree approach reaches a throughput two to three times higher than MPICH, especially when the density of hosts among nodes gets higher.

The first explanation of these results is that we take topological informations into account, whereas MPICH does not: on a heterogeneous platforms, we know which hosts are suitable for computation, and which links have good bandwidth. These informations are taken into account in the linear program, and they help build an efficient set of reduction trees. On the contrary, MPICH builds its own tree without any of these informations, based only on the logical indices of the hosts.

As heterogeneous topologies lead to inefficient MPICH reduction trees, we compare the two approaches in the case that is supposed to be the most favorable to MPICH. We consider a homogeneous fully connected topology. The platform used for this simulation has 10 nodes with the same computing speed (100 Mflops/s) and each pair of processors is connected by a link with the same bandwidth (100 MB/s). The size of all messages is supposed to be the same (100 MB), as well as the time needed to perform all tasks. The rationale to this comparison with homogeneous CPU speeds, link bandwidths and message/task sizes is that the MPICH binomial tree is likely to be as efficient as any tree in our solution set. We perform several simulations with different computation to communication ratios: we let the size of tasks vary from 0.1 to 10^6 Mflops. The results of these simulations are shown in Fig. 13.

We observe that when computation costs gets higher, both approaches give a lower throughput, but our algorithm remains better than MPICH, and with a constant factor (about 4 times better in this case). This result comes from the ability of our multi-tree approaches to distribute the computation on all available hosts; it may result into more communications than in the MPICH algorithm, but in the case of a high computation/communication ratio, their cost is not important.

All these simulations highlight the main two advantages of our approach over MPICH: (i) we take topological information into account, thus we are able to schedule computation on the right hosts; and (ii) using several trees, we can distribute the computation of a single task on several processors. Both considerations help us improving the throughput of the steady-state reduction operation.

6. Related work

We briefly discuss related results from the literature, which we classify in the following three categories:

Models: Following the homogeneous LogP model [15] and its long-message extension LogGP [1], the parameterized LogP model [21] has been proposed for heterogeneous networks. However, these models involve many parameters and are rather difficult to instantiate. Several simpler models have been considered in the literature for heterogeneous platforms

- Banikazemi et al. [2] 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. Some theoretical results (NP-completeness and approximation algorithms) have been developed for the problem of broadcasting a message in this model: see [18,26,27].
- A more complex model is introduced in [3]: 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 [10,9]: 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, $\frac{m}{B_{i,j}}$.
- 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, hence the choice of the *full overlap, one-port* model in this paper.

Collective communication schemes: Macro-communications have been widely studied, in particular for homogeneous topologies. For instance, some papers address the problem of performing collective operation on meshes using a wormhole routing model. In [37], a pipelined broadcast is described for such a mesh, and its performances are tested on a Cray T3D. On the same topology, Barnett et al. [4] study another collective operation : the GLOBAL COMBINE operation, very close to our REDUCE operation, excepted that the operator used in the reduction is now associative and commutative (the order of the elements to reduce has no importance). In [4], the authors describe several efficient algorithms to perform this operation based on a wormhole routing model, but they are interested in the non-pipelined version of the operation, and their goal is to minimize the makespan of one COMBINE operation. Other collective communications, such as multicast,scatter, all-to-all, personalized all-to-all and gather/reduce have been studied in the context of heterogeneous platforms: see [19,25,28–30] among others.

Communication libraries: MPI and its extensions provide several routines for various macro-communications:

- The common standard MPI [35] describes many collective communications, such as BROADCAST, GATHER, ALLTOALL, and REDUCE.
- A recent implementation, called MPICH-G2 [20], is typically designed for clusters and the grid. To perform collective communications, the MPICH-G2 implementation groups processors into different subnets, gathered into layers, according to the communication possibilities available between to different processors (MPI, Globus and/or TCP), and then perform hierarchical communications using these layers. However, pipelining communication is still a project for a next implementation of MPI.
- There exist other communication libraries using the same hierarchical approach: the ECO library [29] measures the round-trip time between different processors to group them into subnets, and then perform the communications using this two-layer topology. The algorithms used inside a given subnet depends upon some of its characteristics: for example, the width of a broadcast tree will differ in a switch-based network and in a bus-based network. MagPIe [22] is another library which groups processors into subnets. The use of only two layers (inter-subnet and intra-subnet communications) is justified as follows in [22]: the high cost of a wide-area communication makes negligible the use of improvements of the communications inside a given cluster. To perform an efficient collective communication, the main goal is to minimize the use of inter-subnet communications.

7. Conclusion

In this paper, we have studied several collective communications, with the objective to optimize the throughout that can be achieved in steady-state mode, when pipelining a large number of operations. Focusing on series of scatters, personalized all-to-alls and reduces, we have shown how to explicitly determine the best steady-state scheduling in polynomial time. The best throughout can easily be found with linear programming, whereas a polynomial description of a valid schedule realizing this throughout is more difficult to exhibit. In particular, we had to use reduction trees to describe a polynomial schedule for the SERIES OF REDUCES problem. It is important to point out that the concrete scheduling algorithms based upon the steady-state operation are asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

One major contribution of this paper is the use of steady-state scheduling techniques to circumvent, so to speak, the intrinsic difficulty of makespan minimization problem. We have been successful for three problems, namely series of scatters, personalized all-to-alls and reduces, as well as for series of broadcasts [6]. However, there are other macro-communication primitives that cannot be captured in this framework: as shown in [5], it is NP-hard to determine the best throughput that can be achieved for the following two problems: (i) series of multicast operations, and (ii) series of general parallel prefix computations, where each node P_i must obtain the result $v_{[0,i]}$ of the reduction limited to those processors whose rank is lower that its own rank. We see that (ii) is very close the SERIES OF REDUCES problem. It would be very interesting to characterize which macro-communication primitives are amenable to a polynomial solution when relaxed by steady-state techniques.

Main notations: For the sake of convenience, we provide the list of the main notations used throughout the paper:

- P_{source}: source processor for scatter operations.
- P_{target}: destination processor for reduce operations.
- $c(e)$ or $c_{i,j}$: time to transfer a unit-size message on edge $e : P_i \to P_j$.
- $s(P_i \to P_j)$: time spent by P_i to send messages to P_j every time-unit.
- m_k: messages whose destination is processor P_k.
- δ_k: size of messages m_k.

- $send(P_i \rightarrow P_j, m_k)$: number of m_k messages sent on edge $P_i \rightarrow P_j$ every time-unit.
- TP: throughput, i.e., number of macro-communications initiated every time-step.
- v_i or $v_{[i,i]}$: initial value held by processor P_i.
- $v_{[k,m]}$: reduction of values $v_k, \ldots, v_m$.
- $size(v_{[k,m]})$: size of $v_{[k,m]}$ messages.
- $T_{k,l,m}$: computational task associated to reduction $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$.
- $w(P_i, T_{k,l,m})$: time needed by processor P_i to compute one task $T_{k,l,m}$.
- $send(P_i \rightarrow P_j, v_{[k,l]})$: number $v_{[k,l]}$ messages sent on edge $P_i \rightarrow P_j$ every time-unit.
- $cons(P_i, T_{k,l,m})$: number of tasks $T_{k,l,m}$ computed by P_i every time unit.
- $\alpha(P_i)$: time spent by P_i computing all tasks every time-unit.
- $\mathcal{T}$: reduction tree.
- $weight(\mathcal{T})$: weight of $\mathcal{T}$, i.e., number of time-stamps which use it during a period.

Acknowledgments

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

References

[1] A. Alexandrov, M.F. Ionescu, K.E. Schauser, C.J. Scheiman, LogGP: incorporating long messages into the LogP model for parallel computation, J. Parallel Distributed Comput. 44 (1) (1997) 71–79.

[2] M. Banikazemi, V. Moorthy, D.K. Panda, Efficient collective communication on heterogeneous networks of workstations, in: Proceedings of the 27th International Conference on Parallel Processing (ICPP'98), IEEE Computer Society Press, Silver Spring, MD, 1998.

[3] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, P. Sadayappan, Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations, in: HCW'99, the Eighth Heterogeneous Computing Workshop, IEEE Computer Society Press, Silver Spring, MD, 1999, pp. 125–133.

[4] M. Barnett, R. Littlefield, D.G. Payne, R. van de Geijn, On the efficiency of global combine algorithms for 2-D meshes with wormhole routing, J. Parallel Distributed Comput. 24 (2) (1995) 191–201.

[5] O. Beaumont, A. Legrand, L. Marchal, Y. Robert, Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogenous platforms, Research Report RR-2004-20, LIP, ENS Lyon, France, April 2004, Available at the url http://graal.ens-lyon.fr/~yrobert.

[6] O. Beaumont, A. Legrand, L. Marchal, Y. Robert, Pipelining broadcasts on heterogeneous platforms, in: International Parallel and Distributed Processing Symposium IPDPS'2004, IEEE Computer Society Press, Silver Spring, MD, 2004.

[7] M. Berkelaar, LP_SOLVE, http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml.

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

[9] P. Bhat, C. Raghavendra, V. Prasanna, Adaptive communication algorithms for distributed heterogeneous systems, J. Parallel Distributed Comput. 59 (2) (1999) 252–279.

[10] P. Bhat, C. Raghavendra, V. Prasanna, Efficient collective communication in distributed heterogeneous systems, in: ICDCS'99 19th International Conference on Distributed Computing Systems, IEEE Computer Society Press, Silver Spring, MD, 1999, pp. 15–24.

[11] K.L. Calvert, M.B. Doar, E.W. Zegura, Modeling internet topology, IEEE Comm. Mag. 35 (6) (June 1997) 160–163.

[12] H. Casanova, Simgrid: a toolkit for the simulation of application scheduling, in: Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01), IEEE Computer Society, Silver Spring, MD, May 2001.

[13] B.W. Char, K.O. Geddes, G.H. Gonnet, M.B. Monagan, S.M. Watt, Maple Reference Manual, 1988.

[14] J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery, J. Zayer, A world wide number field sieve factoring record: on to 512 bits, in: K. Kim, T. Matsumoto (Eds.), Advances in Cryptology—Asiacrypt '96, Lecture Notes in Computer Science, vol. 1163, Springer, Berlin, 1996, pp. 382–394.

[15] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, T.V. Eicken. LogP: towards a realistic model of parallel computation, in: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM Press, New York, 1993.

[16] Entropia. URL: http://www.entropia.com.

[17] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (September 1996) 789–828 (see also http://www-unix.mcs.anl.gov/mpi/mpich/).

[18] N. Hall, W.-P. Liu, J. Sidney, Scheduling in broadcast networks, Networks 32 (14) (1998) 233–253.

[19] J.-I. Hatta, S. Shibusawa, Scheduling algorithms for efficient gather operations in distributed heterogeneous systems, in: 2000 International Conference on Parallel Processing (ICPP'2000), IEEE Computer Society Press, Silver Spring, MD, 2000.

[20] N.T. Karonis, B. Toonen, I. Foster, Mpich-g2: a grid-enabled implementation of the message passing interface, J. Parallel Distributed Comput. 63 (5) (2003) 551–563.

[21] T. Kielmann, H.E. Bal, S. Gorlatch, K. Verstoep, R.F. Hofman, Network performance-aware collective communication for clustered wide area systems, Parallel Comput. 27 (11) (2001) 1431–1456.

[22] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, R.A.F. Bhoedjang, MagPIe: MPI's collective communication operations for clustered wide area systems, in: Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP'99, Atlanta, GA, 1999, pp. 131–140.

[23] A. Legrand, L. Marchal, H. Casanova, Scheduling distributed applications: the SIMGRID simulation framework, in: Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), May 2003.

[24] A. Legrand, L. Marchal, Y. Robert, Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms, Technical Report RR-2003-33, LIP, ENS Lyon, France, June 2003. Available at the url http://graal.ens-lyon.fr/~yrobert.

[25] R. Libeskind-Hadas, J.R.K. Hartline, P. Boothe, G. Rae, J. Swisher, On multicast algorithms for heterogeneous networks of workstations, J. Parallel Distributed Comput. 61 (11) (2001) 1665–1679.

[26] P. Liu, Broadcast scheduling optimization for heterogeneous cluster systems, J. Algorithms 42 (1) (2002) 135–152.

[27] P. Liu, T.-H. Sheng, Broadcast scheduling optimization for heterogeneous cluster systems, in: SPAA'2000 12th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 2000, pp. 129–136.

[28] P. Liu, D.-W. Wang, Reduction optimization in heterogeneous cluster environments, in: 14th International Parallel and Distributed

Processing Symposium (IPDPS'2000), IEEE Computer Society Press, Silver Spring, MD, 2000.
[29] B. Lowekamp, A. Beguelin, Eco: efficient collective operations for communication on heterogeneous networks, in: 10th International Parallel and Distributed Processing Symposium (IPDPS'96), IEEE Computer Society Press, Silver Spring, MD, 1996.
[30] F. Ooshita, S. Matsumae, T. Masuzawa, Efficient gather operation in heterogeneous cluster systems, in: Proceedings of the 16th International Symposium on High Performance Computing Systems and Applications (HPCS'02), IEEE Computer Society Press, Silver Spring, MD, 2002.
[31] Prime, URL: http://www.mersenne.org.
[32] J. Reif, Synthesis of Parallel Algorithms, Morgan Kaufmann, Los Altos, CA, 1993.
[33] A. Schrijver, Combinatorial Optimization: Polyhedra and Efficiency, Algorithms and Combinatorics, vol. 24, Springer, Berlin, 2003.
[34] SETI, URL: http://setiathome.ssl.berkeley.edu.
[35] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra, MPI the Complete Reference, The MIT Press, Cambridge, MA, 1996.
[36] The MuPAD Group (B. Fuchssteiner et al.), MuPAD User's Manual, Wiley, New York, 1996.
[37] J. Watts, R. Van De Geijn, A pipelined broadcast for multidimensional meshes, Parallel Process. Lett. 5 (2) (1995) 281–292.

Arnaud Legrand received the PhD degree from Ećole normale supérieure de Lyon in 2003. He is currently a CNRS permanent researcher in the ID-IMAG laboratory in Grenoble. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques.

Loris Marchal received the Master's degree from Ećole normale supérieure de Lyon in 2003. He 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.

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 IEEE and the IEEE Computer Society, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems.*