Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

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

Arnaud Legrand,
Loris Marchal,
Yves Robert

June 2003

Research Report N° 2003-33

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

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

Arnaud Legrand,
Loris Marchal,
Yves Robert

June 2003

Abstract

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 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-communication 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, steady-state, collective communications, heterogeneous platforms

Résumé

Nous nous intéressons ici aux communications qui ont lieu lors de l'exécution d'une application complexe distribuée sur un environnement hétérogène de type *"grille de calcul"*. De telles applications font un usage intensif de communications collectives, telles que des diffusions ou des échanges totaux personnalisés, ou encore des opération de réduction. Nous nous intéressons ici à optimiser le débit de telles opérations en régime permanent, en supposant qu'un grand nombre de communications collectives semblables doivent être effectuées successivement, comme c'est le cas pour le parallélisme de données. La plateforme hétérogène que nous visons est modélisée par un graphe où les différentes ressources (calcul ou communication) ont des vitesses différentes. Pour les opérations de communications précédentes, nous montrons comment calculer le débit optimal et comment construire un ordonnancement périodique qui réalise ce débit.

Mots-clés: Ordonnancement, régime permanent, communications collectives, plateforme hétérogène

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), and the makespan is not a significant measure for such problems. Rather, we focus on the optimization of the steady-state mode, and we aim at optimizing the throughput of a series of macro-communications instead of the makespan of each macro-communication taken individually.

In this paper, we focus on scatter and reduce operations (note that broadcasts are dealt with in the companion report [5]). Here are the definitions of these operations:

Scatter One processor P_{source} has to send a distinct message to each target processor $P_{t_1}, \dots, P_{t_N}$

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 of target processors $\{P_{t_0}, \dots, P_{t_N}\}$.

Reduce Each processor P_i among the set $P_{r_0}, \dots, P_{r_N}$ of participating processors has a local value v_i, and the goal is to calculate $v = v_0 \oplus \cdots \oplus v_N$, where $\oplus$ is an associative, non-commutative operator. The result v is to be stored on processor P_{target}.

Series of Reduces A series of Reduce operations is to be performed, from the same set of participating processors and to the same target.

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.

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 the gossiping problem. The more complex SERIES OF REDUCES problem is described in Section 4. Section 4.7 presents some experimental results. Section 5 gives an overview of related work. Finally, we state some concluding remarks in Section 6.

2 Framework

We adopt a model of heterogeneity close to the one developed by Bhat, Raghavendra and Prasanna [8]. 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 5), 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 to send a message of length m at time t. This transfer will last $m \times c(i,j)$ 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 $t + m \times c(i,j)$:

- 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.

Our framework is the following. We will express both optimization problems (Series of Scatters and Series of Reduces) as a set of linear constraints, so as to build a linear program. Basically, the linear constraints aim at determining which fraction of time does each processor spend communicating which message on which edge. We solve the linear program (in rational numbers) with standard tools (like `lpsolve` [6] or Maple [10]), and we use the solution to build a schedule that implements the best communication scheme.

Notations A few variables and constraints are common to all problems, because they arise from the one-port model assumption. We call $s(P_i \to P_j)$ 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:

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

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

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

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

We will later add further constraints corresponding to each specific problem under study. We first illustrate how to use this framework on the simple 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 $\{P_t, t \in T\}$. The source processor has a message m_t to send to each processor P_t. 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_t.

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,

- *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:

$$\forall P_i, P_j, \qquad s(P_i \to P_j) \;=\; \sum_{m_k} \mathit{send}(P_i \to P_j, m_k) \times c(i,j) \tag{4}$$

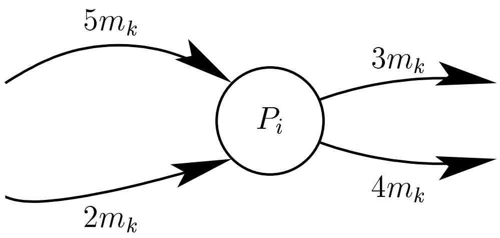


Figure 1: Conservation Law
$(P_i \neq P_k)$

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

$$\forall P_i, \forall m_k, k \neq i, \qquad \sum_{P_j, (j,i) \in E} \mathit{send}(P_j \to P_i, m_k) \;=\; \sum_{P_j, (i,j) \in E} \mathit{send}(P_i \to P_j, m_k) \tag{5}$$

Moreover, let the throughput at processor P_k be the number of messages m_k received at this node, i.e. the sum of all messages of type m_k received by P_k via all its incoming edges. We impose that the same throughput TP is achieved at each target node, and we write the following constraint:

$$\forall P_k, k \in T, \qquad \sum_{P_i, (i,k) \in E} \mathit{send}(P_i \to P_k, m_k) \;=\; \mathrm{TP} \tag{6}$$

We can summarize the previous constraints in a linear program:

STEADY-STATE SCATTER PROBLEM ON A GRAPH SSSP(G)
Maximize TP,
subject to

$$\begin{array}{ll}
\forall P_i, \forall P_j, & 0 \leqslant s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1 \\
\forall P_i, P_j, & s(P_i \to P_j) = \sum_{m_k} \mathit{send}(P_i \to P_j, m_k) \times c(i,j) \\
\forall P_i, \forall m_k, k \neq i, & \sum_{P_j,(j,i)\in E} \mathit{send}(P_j \to P_i, m_k) = \sum_{P_j,(i,j)\in E} \mathit{send}(P_i \to P_j, m_k) \\
\forall P_k, k \in T & \sum_{P_i,(i,k)\in E} \mathit{send}(P_i \to P_k, m_k) = \mathrm{TP}
\end{array}$$

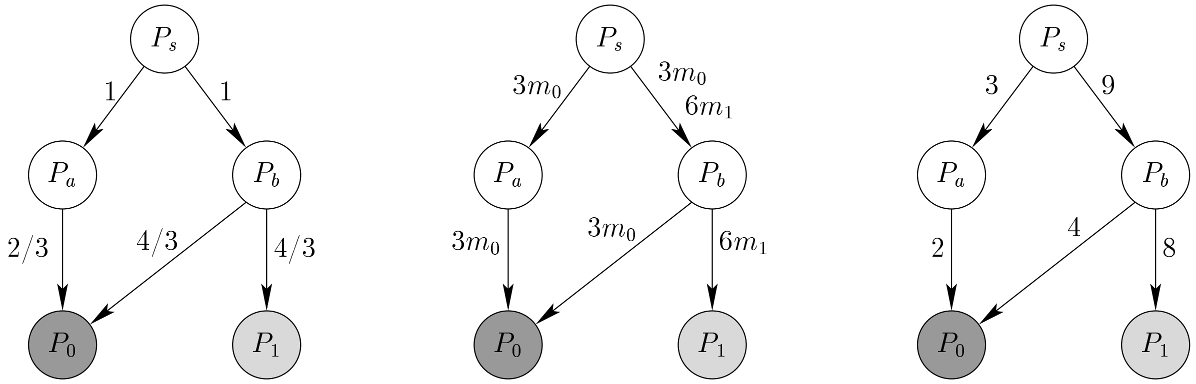
This linear program can be solved in polynomial time by using tools like `lpsolve`[6], Maple [10] or MuPaD [12]. We solve it over the rational numbers. Then we compute the least common multiple of the denominators of all the variables, which leads to a periodic schedule where all quantities are integers. This period is potentially very large, but we discuss in Section 4.6 how to approximate the result for a smaller period.

3.2 Toy example

To illustrate the use of the linear program, consider the simple example described on Figure 2. Figure 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 messages to two target processors P_0 and P_1.

Figures 2(b) and 2(c) show the results of the linear program: on Figure 2(b) we represent the number of messages of each type going through the network, whereas Figure 2(c) describes the occupation of each edge.

The throughput achieved with this solution is $\mathrm{TP} = 1/2$, 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. The linear constraints allow for using multiple routes in order to reach the best throughput.



(a) Topology (b) *send* values (c) *s* values

Figure 2: Toy example for the SERIES OF SCATTERS problem. The values are given for a period of 12: the achieved throughput is 6 messages every 12 time-units.

3.3 Building a schedule

Once the linear program is solved, we get the period T of the schedule and the integer number of messages going through each link. We still need to exhibit a schedule of the message transfers where emissions (resp. receptions) never overlap on one node. This is done using a weighted-matching algorithm, as explained in [4]. 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 transfer $send(P_i \to P_j, m_k)$, insert an edge between P_i^{send} and P_j^{recv} labeled with the time needed by the transfer: $send(P_i \to P_j, m_k) \times c(i, j)$.

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 [23, vol.A chapter 20] provides in polynomial time a polynomial number of matchings, which we are used to perform the different communications. Rather than going into technical details, we illustrate this algorithm on the previous example. The bipartite graph constructed with the previous *send* and s values (as returned by the linear program) is represented on Figure 3(a). It can be decomposed into four matchings, represented on Figures 3(b) to 3(e).

These matchings explain how to split the communications to build a schedule. Such a schedule is described on Figure 4(a). We assume that the transfer of a message can be split into several parts (for example, the fourth 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 multiplicating 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 Figure 4(a).

3.4 Asymptotic optimality

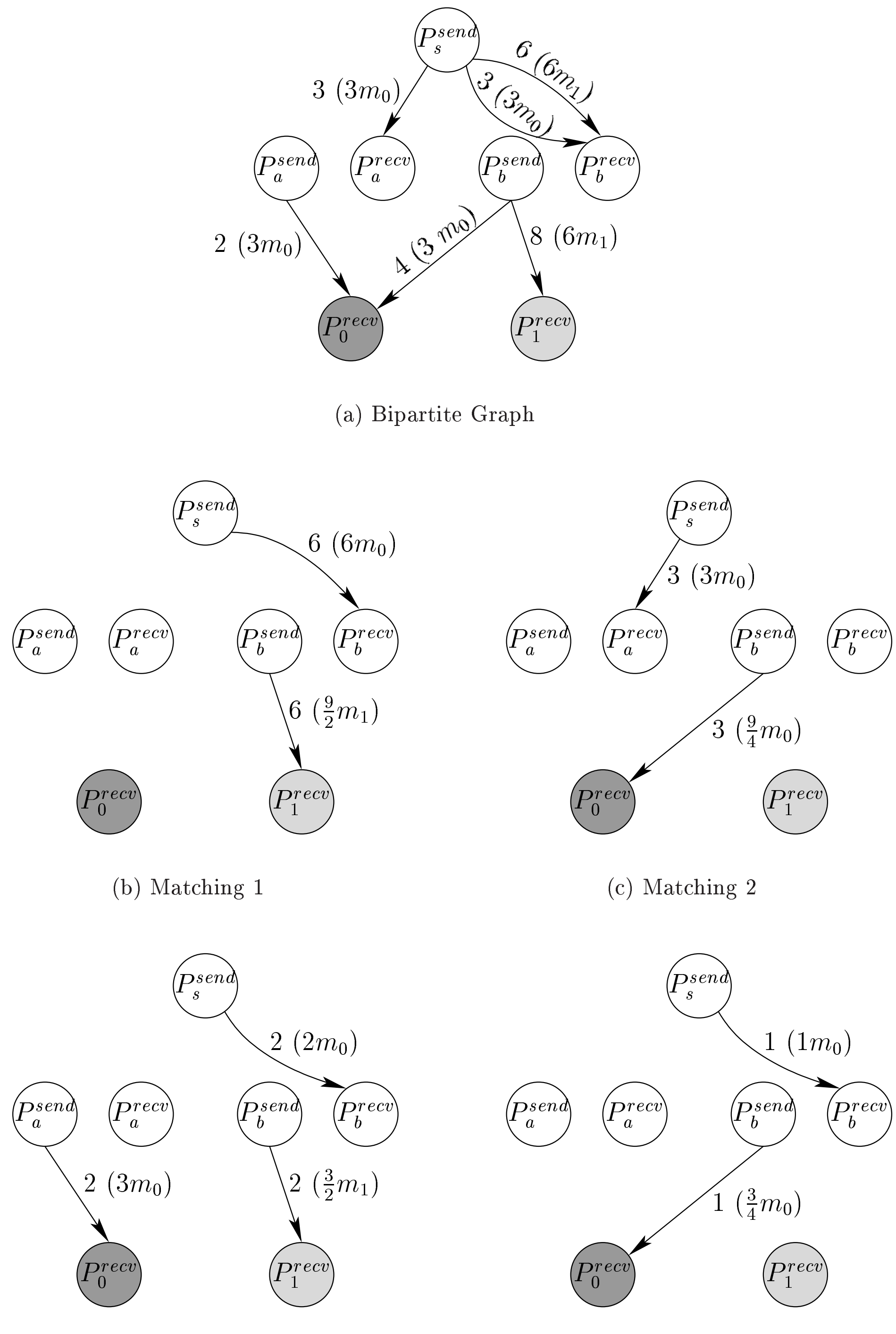
In this section, we prove that the previous periodic schedule is asymptotically optimal: basically, no scheduling algorithm (even non periodic) can execute more scatter operations in a given time-frame than ours, up to a constant number of operations. This section is devoted to the formal statement of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

Given a platform graph $G = (V, E, c)$, a source processor P_{source} holding an infinite number of unit-size messages, a set of target processors $\mathcal{P}_T = \{P_{t_1}, \ldots, P_{t_N}\}$ 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 $\mathrm{TP}(G)$ 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. $opt(G, K) \leqslant \mathrm{TP}(G) \times K$

Proof. Consider an optimal schedule, 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:

- $\forall P_i, P_j, S(P_i \to P_j) = \sum_{m_k} N(P_i \to P_j, m_k) \times c(i, j)$
- $\forall P_i, \forall P_j, 0 \leqslant S(P_i \to P_j) \leqslant K$
- $\forall P_i, \sum_{P_j, (i,j) \in E} S(P_i \to P_j) \leqslant K$ (time for P_i to send messages in the one-port model)
- $\forall P_i, \sum_{P_j, (j,i) \in E} S(P_j \to P_i) \leqslant K$ (time for P_i to receive messages in the one-port model)
- $\forall P_i, \forall m_k, k \neq i, \sum_{P_j, (j,i) \in E} N(P_j \to P_i, m_k) = \sum_{P_j, (i,j) \in E} N(P_i \to P_j, m_k)$ (conservation law for messages forwarded by P_i to P_k)
- $\forall P_k \in \mathcal{P}_T, opt(G, K) = \sum_{P_j, (j,k) \in E} N(P_j \to P_k, m_k)$ (same number of messages received on each target node)



(a) Bipartite Graph

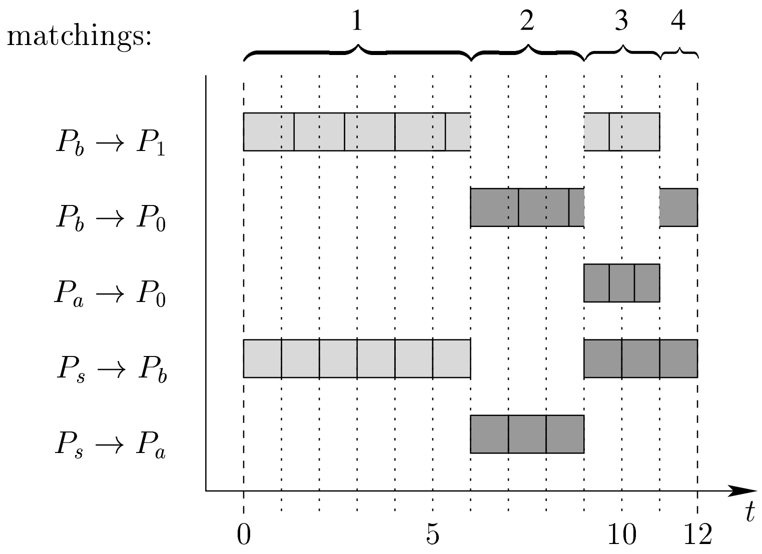
(b) Matching 1

(c) Matching 2

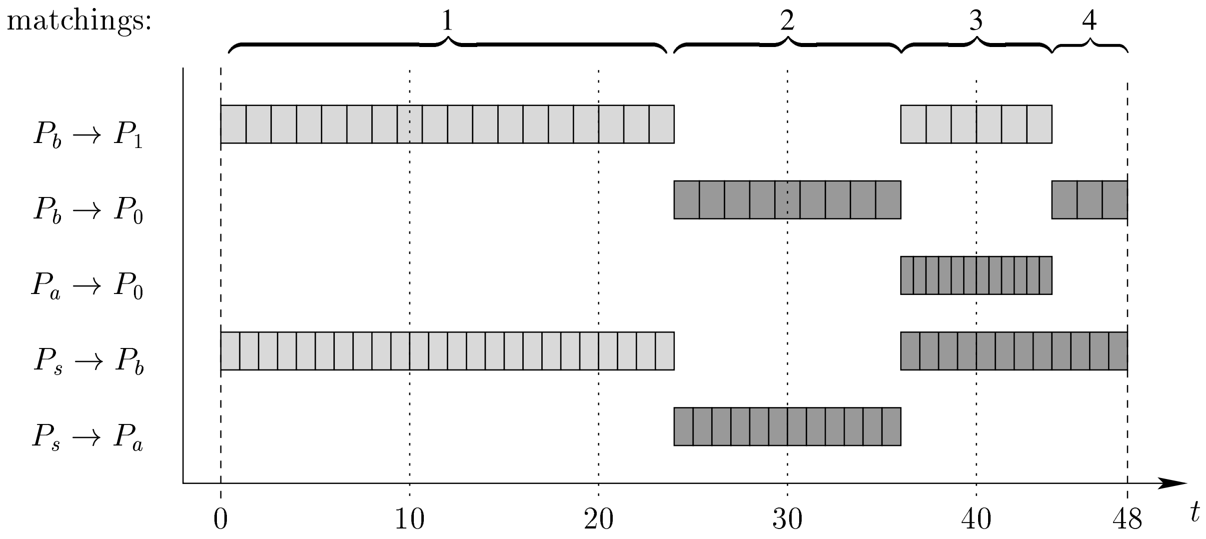
(d) Matching 3

(e) Matching 4

Figure 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) Schedule if we allow for splitting messages (period = 12)



(b) Schedule without any split message (period = 48)

Figure 4: Different possible schedules for the example.

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 hold, hence $\frac{opt(G,K)}{K} \leqslant \mathrm{TP}(G)$, 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, 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 $\mathrm{TP}(G)$. Determine the period T such that every communication time is an integer. We describe the schedule in terms of periods, i.e. in steps of length T.
- For each processor P_i, for each type of message m_k $(i \neq k)$, we use a buffer of messages of type m_k on processor P_i. We call $\mathit{buffer}_{P_i,m_k}$ the number of messages of type m_k in the buffer of processor P_i. In steady-state mode, the buffer should contain at least as many messages of each type as the number sent during one period, that is, the minimum size of a buffer $\mathit{buffer}_{P_i,m_k}$ is $\mathit{buff\text{-}min\text{-}size}_{P_i,m_k} = \sum_{P_j} send(P_i \to P_j, m_k)$. Note this is the same quantity as the number of messages of type m_k received by P_i within each period.
- Initialization phase: at each period, if the buffer is not filled (that is while $\mathit{buffer}_{P_i,m_k} < \mathit{buff\text{-}min\text{-}size}_{P_i,m_k}$), P_i sends no message m_k. After that the number of messages has reached its minimum capacity, the sending policy of node P_i is the same as in the steady-state: it sends $send(P_i \to P_j, m_k)$ messages m_k to P_j, using the communication schedule of the steady-state solution. As P_i receives not more messages m_k than $\mathit{buff\text{-}min\text{-}size}_{P_i,m_k}$ in one period, the buffer will never exceed a maximal capacity of $2 \times \mathit{buff\text{-}min\text{-}size}_{P_i,m_k}$.
- Let I be the maximal width of the graph G (its diameter) times the duration of a period. I is a constant independent of K. As the maximum latency between the source and any node is not greater than the maximal width of the graph G, after I time-steps, $\mathit{buffer}_{P_i,m_k} \geqslant \sum_{P_j} send(P_i \to P_j, m_k)$ for each processor P_i and each message type m_k.
- This is the beginning of the following steady-state phase, all processors send as many message as computed earlier, during $r = \lfloor \frac{K-2I-T}{T} \rfloor$ period of time T.
- Clean-up phase: the source processor stops sending any message, and the other processors send messages as in the previous phase until their buffers get empty. As each buffer contains not more messages than $2 \times \mathit{buff\text{-}min\text{-}size}_{P_i,m_k}$, and since the maximum time for a message to reach its destination node is I, this may not take a time greater than $I + T$.
- The number of messages sent to each node by this algorithm within K time-units is not less than the number of messages sent during the steady-state phase, which is $steady(G,K) = r \times T \times \mathrm{TP}(G)$.

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, $opt(G,K) \leqslant \mathrm{TP}(G) \times K$. From the description of the algorithm, we have $steady(G,K) = r \times T \times \mathrm{TP}(G) = \lfloor \frac{K-2I-T}{T} \rfloor \times T \times \mathrm{TP}(G)$. Since $\mathrm{TP}(G)$, I and T are constants independent of K, the result holds. ■

3.5 Extension to gossiping

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 GOSSIPS, i.e. a series of personalized all-to-all problems. In this context, a set of source processors $\{P_s, s \in \mathcal{S}\}$ has to send a series of messages to a set of target processors $\{P_t, t \in \mathcal{T}\}$. 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 to P_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 at each target node. We give the linear program summarizing all this constraints:

STEADY-STATE PERSONALIZED ALL-TO-ALL PROBLEM ON A GRAPH SSPA2A(G)
Maximize TP,
subject to

$$
\begin{array}{ll}
\forall P_i, \forall P_j, & 0 \leqslant s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1 \\
\forall P_i, P_j, & s(P_i \to P_j) = \sum_{m_{k,l}} send(P_i \to P_j, m_{k,l}) \times c(i,j) \\
\forall P_i, \forall m_k, k \neq i, l \neq i, & \sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_{k,l}) = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_{k,l}) \\
\forall P_k, \forall m_{k,l} & \sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = \mathrm{TP}
\end{array}
$$

After solving this linear system, we have to compute the period of a schedule as the least common multiple of all denominators in the solution, and then to build a valid schedule, using the weighted-matching algorithm just as previously. Furthermore, we can prove the same result of asymptotic optimality:

Proposition 2. *For the* SERIES OF GOSSIPS *problem, the scheduling algorithm based on the steady-state operation is asymptotically optimal.*

4 Series of Reduces

We recall the sketch of a reduce operation: some processors $P_{r_0}, \dots, P_{r_N}$ own a value $v_0, \dots, v_N$. The goal is to compute the reduction of these values: $v = v_0 \oplus \dots \oplus v_N$, where $\oplus$ is an associative, non-commutative[1] operator. This operation is useful for example to compute a maximum/minimum, sort or gather data in a particular order (see [11] for other applications). 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 we add computational tasks to merge the different messages into new ones. Let $v_{[k,m]}$ denote the partial result corresponding to the reduction of the values $v_k, \dots, v_m$:

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

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:

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

[1]When the operator is commutative, we have 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.

We let $T_{k,l,m}$ denote the computational task needed for 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. 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

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 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 Figure 5.

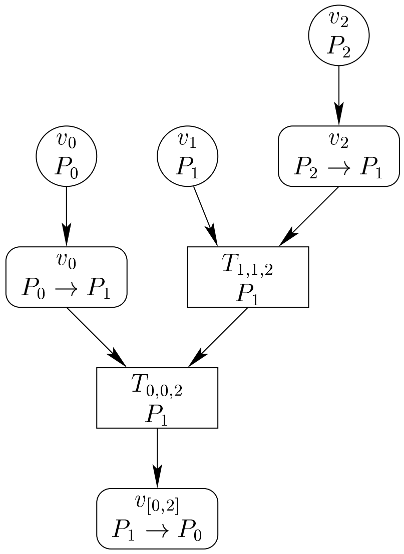


Figure 5: Simple example of a reduction tree

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 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 $\{v_0^t, \dots, v_N^t\}$ 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 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 containing $v_{[k,l]}$ values and which 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,
- $\alpha(P_i)$ is the time spent by P_i computing tasks within each time-unit. This quantity is obviously bounded:
$$\forall P_i, \qquad 0 \leqslant \alpha(P_i) \leqslant 1 \tag{7}$$
- $size(v_{[k,l]})$ 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:

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

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

$$\forall P_i, \qquad \alpha(P_i) \;=\; \sum_{T_{k,l,m}} cons(P_i, T_{k,l,m}) \times w(P_i, T_{k,l,m}) \tag{9}$$

We can write the following "conservation law" which expresses that the number of packets of type $v_{[k,m]}$ reaching a node (either created by a local computation of a task $T_{k,l,m}$ or by a transfer from another node) is used in a local computation ($T_{n,k,m}$ or $T_{k,m,n}$) or sent to another node:

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

Note that this equation is not verified for the message $v_{[i,i]}$ on processor P_i (we assume we have an unlimited number of such messages). It is is also not verified for the final complete message $v = v_{[0,n-1]}$ on the target processor. In fact, the number of messages v reaching the target processor P_{target} is the throughput TP that we want to maximize:

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

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

$$
\begin{array}{ll}
\forall P_i, \forall P_j, & 0 \leqslant s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1 \\
\forall P_i, & \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1 \\
\forall P_i, & 0 \leqslant \alpha(P_i) \leqslant 1 \\
\forall P_i, P_j, & s(P_i \to P_j) = \sum_{v_{[k,l]}} send(P_i \to P_j, v_{[k,l]}) \times size(v_{[k,l]}) \times c(i,j) \\
\forall P_i, & \alpha(P_i) = \sum_{T_{k,l,m}} cons(P_i, T_{k,l,m}) \times w(P_i, T_{k,l,m})
\end{array}
$$

$\forall P_i, \forall v_{[k,m]}$ with $(k \neq i$ or $m \neq i)$ and $(target \neq i$ or $k \neq 0$ or $m \neq n-1)$,

$$
\begin{aligned}
&\sum_{P_j,(j,i)\in E} send(P_j \to P_i, v_{[k,m]}) + \sum_{k\leqslant l<m} cons(P_i, T_{k,l,m}) \\
&\quad = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, v_{[k,m]}) + \sum_{n>m} cons(P_i, T_{k,m,n}) + \sum_{n<k} cons(P_i, T_{n,k-1,m})
\end{aligned}
$$

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

As for the Series of Scatters problem, after solving this linear program in rational numbers, we compute the least common multiple of all denominators, and we multiply every variable by this quantity. We then obtain an integer solution during a period T. We formally define the integer solution as an application $\mathcal{A}$ which associates an integer value to each variable.

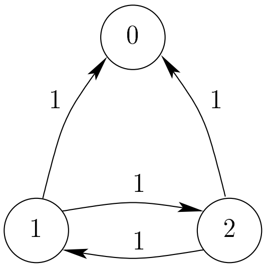
4.3 Building a schedule

Once the solution is computed, we have to exhibit a concrete schedule that achieves it. To complicate matters, the description of the schedule during a single period is not enough, we need to explicit the initialization and termination phases. A naive way would be to describe a schedule for a duration T' multiple of T in extension, explaining how the values v_0 to $v_{\mathrm{TP}.\frac{T'}{T}-1}$ can be computed in time T', and to prove that this schedule can be pipelined. This is done on Figure 6 for the simple example, where $T = 3$ and $T' = 6$. The main problem of this approach is that the period T is not polynomially bounded[2] in the size of the input parameters (the size of the graph), so describing the schedule in extension cannot be done in polynomial time. Furthermore, it might not even be feasible from a practical point of view, if T is too large.

To circumvent the extensive description of the schedule, we use reduction trees. For each time-stamp t between 0 and $T'-1$ a reduction tree is used to reduce the values $v_0^t, \ldots, v_{N-1}^t$. The reduction trees corresponding to the example of Figure 6 are illustrated on Figure 7. A given tree $\mathcal{T}$ might be used by many time-stamps t. We will see that the description of a schedule as a family of trees weighted by the throughput of each tree is more compact than the extensive description of Figure 6(d).

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 $cons(T_{k,l,m}, P_i)$) 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 couple *(message, location)*. The inputs of a computational task $cons(T_{k,l,m}, P_i)$ 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 $(v_{[k,m]}, P_j)$.

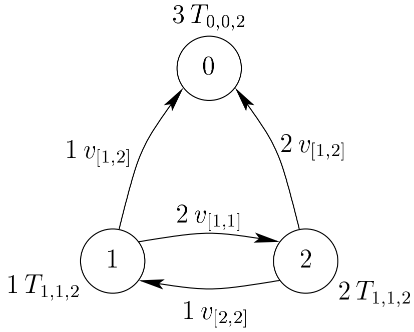
[2]In fact, because it arises from the linear program, $\log T$ is indeed a number polynomial in the problem size, but T itself is not, and describing what happens at every time-step would be exponential in the problem size.



(a) Topology

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

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



(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

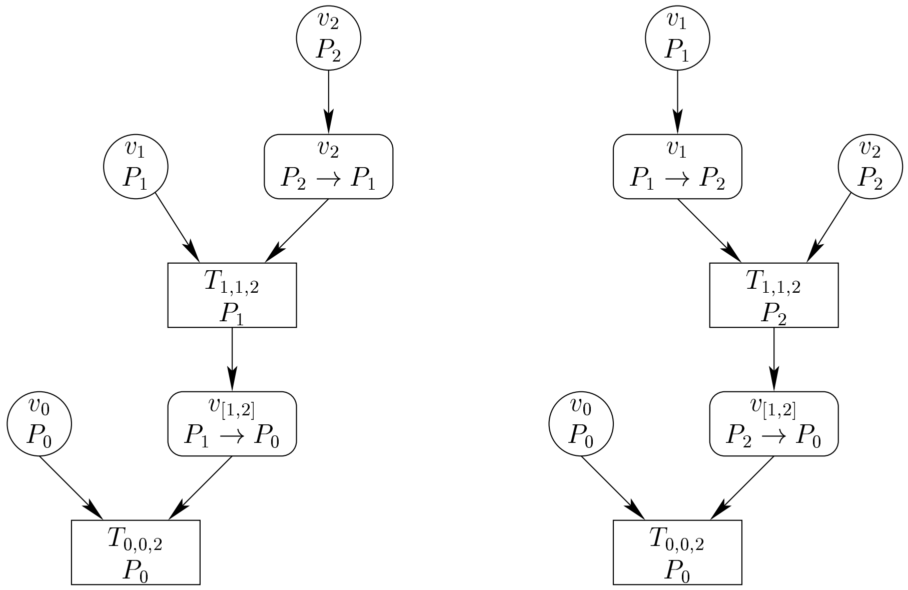
6(a) The topology of the network. Each edge e is labeled with its communication cost $c(e)$. Every processor can process any task in one time-unit, except node 0 which can process any two tasks in one time-unit. The size of every message is 1. The target node is node 0.

6(b) The solution of the linear program.

6(c) The results of the linear program mapped on the topology graph.

6(d) and 6(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 reduced are labeled with their time-stamp (upper indice). Figure 6(d) shows the non-pipelined schedule, while Figure 6(e) presents the pipelined version, leading to a throughput of one reduce operation per time-unit.

Figure 6: Exhaustive schedule derived from the results of the linear program



(a) Reduction tree T_0 (throughput 1/3)

(b) Reduction tree T_1 (throughput 2/3)

Figure 7: Two reduction trees used in the schedule described on Figure 6(d)

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_i.*

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

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

We state the following result:

Lemma 2. *We can build in polynomial time a set of weighted trees $\mathcal{S} = \{(\mathcal{T}, w(\mathcal{T}))\}$, such that:*

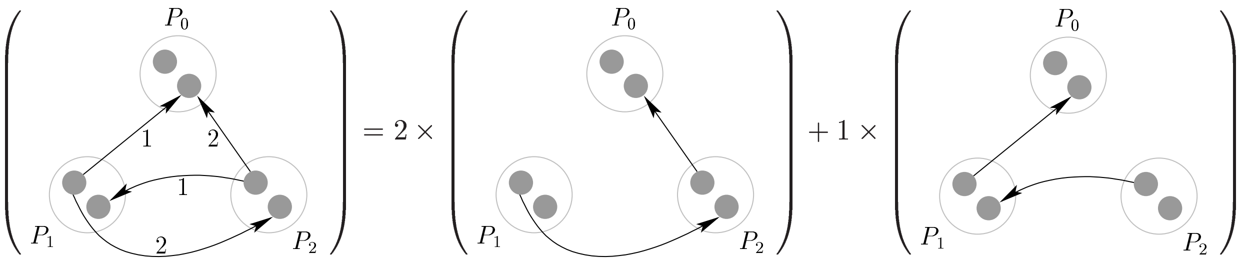
- $\forall \mathcal{T} \in \mathcal{S}, w(\mathcal{T}) \in \mathbb{N}$
- *$card(\mathcal{S})$ is polynomial in the size of the topology graph G,*
- $\displaystyle\sum_{\mathcal{T} \in \mathcal{S}} w(\mathcal{T}) \times \chi_{\mathcal{T}} = \mathcal{A}$

The constructive proof of this lemma will be given in Section 4.4, as an algorithm to extract reduction trees from a solution $\mathcal{A}$. 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 pipelined reduce operations. We use the same approach as for the scatter operation, based on a weighted-matching algorithm. 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 need to perform the transfer: $w(\mathcal{T}) \times size(v_{[k,m]}) \times c(i,j)$.

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, 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 simple 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 a solution $\mathcal{A}$. We assume that $\mathcal{A}$ is a integer solution of period T. The algorithm is described on Figure 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 $w(\mathcal{T})$ that it can produce, which is the minimum throughput of all tasks used in the tree. Then, we update the solution $\mathcal{A}$ by decreasing all tasks used in $\mathcal{T}$ by a factor $w(\mathcal{T})$. We will now prove the correctness and the termination of the algorithm:

Theorem 1. *The algorithm* EXTRACT_TREES*($\mathcal{A}$) produces a set of trees* TREES *such that:*

- $\mathcal{A} = \sum_{\mathcal{T} \in \textsc{Trees}} w(\mathcal{T}) \times \chi_{\mathcal{T}}$,
- *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. We call $\mathcal{A}^{orig}$ the solution at the beginning of the algorithm, and $\mathcal{A}$ the solution updated at each step. We prove that the following property is verified during the execution of the algorithm:

$$\mathcal{H} := \left\{ \begin{array}{c} \mathcal{A}^{orig} = \mathcal{A} + \sum_{\mathcal{T} \in \textsc{Trees}} w(\mathcal{T}) \times \chi_{\mathcal{T}} \\ \mathcal{A} \text{ is a valid solution to reach a throughput of TP} - \sum_{\mathcal{T} \in \textsc{Trees}} w(\mathcal{T}) \\ \forall \mathcal{T} \in \textsc{Trees}, \mathcal{T} \text{ is a valid reduction tree} \end{array} \right\}$$

At the beginning of the program, we have $\mathcal{A}^{orig} = \mathcal{A}$ and $\textsc{Trees} = \emptyset$, so that $\mathcal{H}$ is true. We prove that every step of the loop in EXTRACT_TREES preserves this property.

FIND_TREE computes a list of tasks such that each input of every task is produced by another task in the list or is a value $v_{[i,i]}$ on processor P_i, and the output of these tasks is $v_{[0,N]}$. So FIND_TREE

FIND_TREE($\mathcal{A}$)

$inputs := (v_{[0,N]}$ on node $P_{target})$
$tasks := ()$
while $\exists input \in inputs$ with $input \neq (v_{[i,i]}$ on node $P_i)$ **do**
  find a $input \in inputs$ such that $input \neq (v_{[i,i]}$ on node $P_i)$
  $(v_{[k,m]}$ on node $P_i) := input$
  if $\exists l$ such that $\mathcal{A}(cons(T_{k,l,m}, P_i)) > 0$ **then**
    {the message $v_{[k,m]}$ is computed in place}
    suppress $input$ from $inputs$
    add two inputs to $inputs$: $(v_{[k,l]}$ on node $P_i)$ and $(v_{[l+1,m]}$ on node $P_i)$
    add the task $cons(T_{[k,l,m]}, P_i)$ to $tasks$
    next 6
  else if $\exists P_j$ such that $\mathcal{A}(send(P_i \rightarrow P_j, v_{[k,m]})) > 0$ **then**
    {the message $v_{[k,m]}$ is received from P_j}
    suppress $input$ from $inputs$
    add one input to $inputs$: $(v_{[k,m]}$ on node $P_j)$
    add the task $send(P_j \rightarrow P_i, v_{[k,m]})$ to $tasks$
    next 6
RETURN T

EXTRACT_TREE($\mathcal{A}$)

TREES := ()
while $\sum_{T \in \text{TREES}} w(T) < \text{TP}$ **do**
  $T :=$ FIND_TREE($\mathcal{A}$)
  $w(T) = \min\{\mathcal{A}(task), task \in T\}$
  for all $task \in T$ **do**
    $\mathcal{A}(task) = \mathcal{A}(task) - w(T)$
  PUSH($(T, w(T))$, TREES)
RETURN TREES

Figure 8: Extracting reduction trees from a solution $\mathcal{A}$

computes a valid reduction tree $\mathcal{T}$. Moreover, in $\mathcal{T}$ all the tasks have a positive value in $\mathcal{A}$. The throughput $w(\mathcal{T})$ computed is such that for each $cons(T_{k,l,m}, P_i)$ and each $send(P_i \to P_j, v_{[k,m]})$ appearing in $\mathcal{T}$, we have $\mathcal{A}(cons(T_{k,l,m}, P_i)) \geqslant w(\mathcal{T})$ and $\mathcal{A}(send(P_i \to P_j, v_{[k,m]})) \geqslant w(\mathcal{T})$. As $\mathcal{T}$ is a reduction tree, the conservation law stands for the values given by $\chi_{\mathcal{T}}$, that is:

$$\begin{aligned}
&\forall P_i, \forall v_{[k,m]} \text{ with } (k \neq i \text{ or } m \neq i) \text{ and } (target \neq i \text{ or } k \neq 0 \text{ or } m \neq n-1)\\
&\quad \sum_{P_j,(j,i)\in E} \chi_{\mathcal{T}}(send(P_j \to P_i, v_{[k,m]})) + \sum_{k\leqslant l<m} \chi_{\mathcal{T}}(cons(P_i, T_{k,l,m}))\\
&= \sum_{P_j,(i,j)\in E} \chi_{\mathcal{T}}(send(P_i \to P_j, v_{[k,m]})) + \sum_{n>m} \chi_{\mathcal{T}}(cons(P_i, T_{k,m,n})) + \sum_{n<k} \chi_{\mathcal{T}}(cons(P_i, T_{n,k-1,m}))
\end{aligned}$$

As $\mathcal{A}$ is a valid solution, this equation is also true for the value of $\mathcal{A}$. So it also true for $\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}}$:

$$\begin{aligned}
&\text{(under the same conditions)}\\
&\quad \sum_{P_j,(j,i)\in E} (\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}})(send(P_j \to P_i, v_{[k,m]})) + \sum_{k\leqslant l<m} (\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}})(cons(P_i, T_{k,l,m}))\\
&= \sum_{P_j,(i,j)\in E} (\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}})(send(P_i \to P_j, v_{[k,m]})) + \sum_{n>m} (\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}})(cons(P_i, T_{k,m,n}))\\
&\qquad + \sum_{n<k} (\mathcal{A} - w(\mathcal{T}) \times \chi_{\mathcal{T}})(cons(P_i, T_{n,k-1,m}))
\end{aligned}$$

So $\mathcal{A}$ is a valid solution after being updated. Besides, we updated the value of $\mathcal{A}$ for the tasks appearing in $\mathcal{T}$ such that the $\mathcal{A}$ after modifications is the sum of $\mathcal{A}$ before modifications and of $w(\mathcal{T}) \times \chi_{\mathcal{T}}$. So $\mathcal{H}$ is verified after the execution of a step of the algorithm. At the end, we get a set of trees such that $\sum_{\mathcal{T}\in \text{Trees}} w(\mathcal{T}) = \text{TP}$.

We now prove that we extract only a polynomial number of trees. At each step, we compute the minimum throughput of each task on a tree to get $w(\mathcal{T})$, and we decrease the values of all these tasks in $\mathcal{A}$ by $w(\mathcal{T})$. So there is at least one task realizing the minimum whose new value in $\mathcal{A}$ will be 0. In other words, we delete at least one task for every new tree extracted. The total number of tasks is not greater than

- $N^3 \times n$ for the computational tasks, where N is the number of processors participating to the reduction and $n \geqslant N$ is the total number of processors: there are N^3 possible values for $T_{k,l,m}$ on each of the n processors,
- $N^2 \times n^2$ for the communication tasks: there are N^2 possible message types on each link, and the number of links is bounded by n^2.

Therefore, the algorithm extracts at most $2n^4$ reduction trees. Finally, a new task is added to the current tree at each step of FIND_TREE, so the algorithm can be executed in polynomial time. ■

4.5 Asymptotic optimality

We can prove the same result of asymptotic optimality as for the scatter and gossip operations:

Proposition 3. *For the* Series of Reduces *problem, the scheduling algorithm based on the steady-state operation is asymptotically optimal.*

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. We propose here to approximate the solution with a periodic solution of period T_{fixed}.

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

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

The one-port constraints are satisfied for $\{\mathcal{T}, w(\mathcal{T})\}$ 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 $\frac{1}{T_{fixed}} \times \sum_{\mathcal{T} \in \text{Trees}}$ of the approximated solution and the original throughput TP:

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

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

Proposition 4. *We can derive a steady-state operation for periods of arbitrary length, whose throughput converges to the optimal solution as the period size increases.*

4.7 Experimental Results

We work out a complete example in this section. The platform used is generated by Tiers, a random generator of topology [9]. The bandwidths of the links and the computing speeds of the processors are randomly chosen. The platform is represented on Figure 9. We assume that all the $v_{[k,m]}$ have the same size (10) and that the time needed to compute a task on processor P_i is $10/s_i$, where s_i is the speed of P_i shown in the figure. 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.

Figure 10 presents the results of the linear program mapped on the topology (so the period is normalized to 1). The optimal throughput is TP = 2/9. Two reduction trees can be extracted from these results with our algorithm, they are presented on Figures 11 and 12.

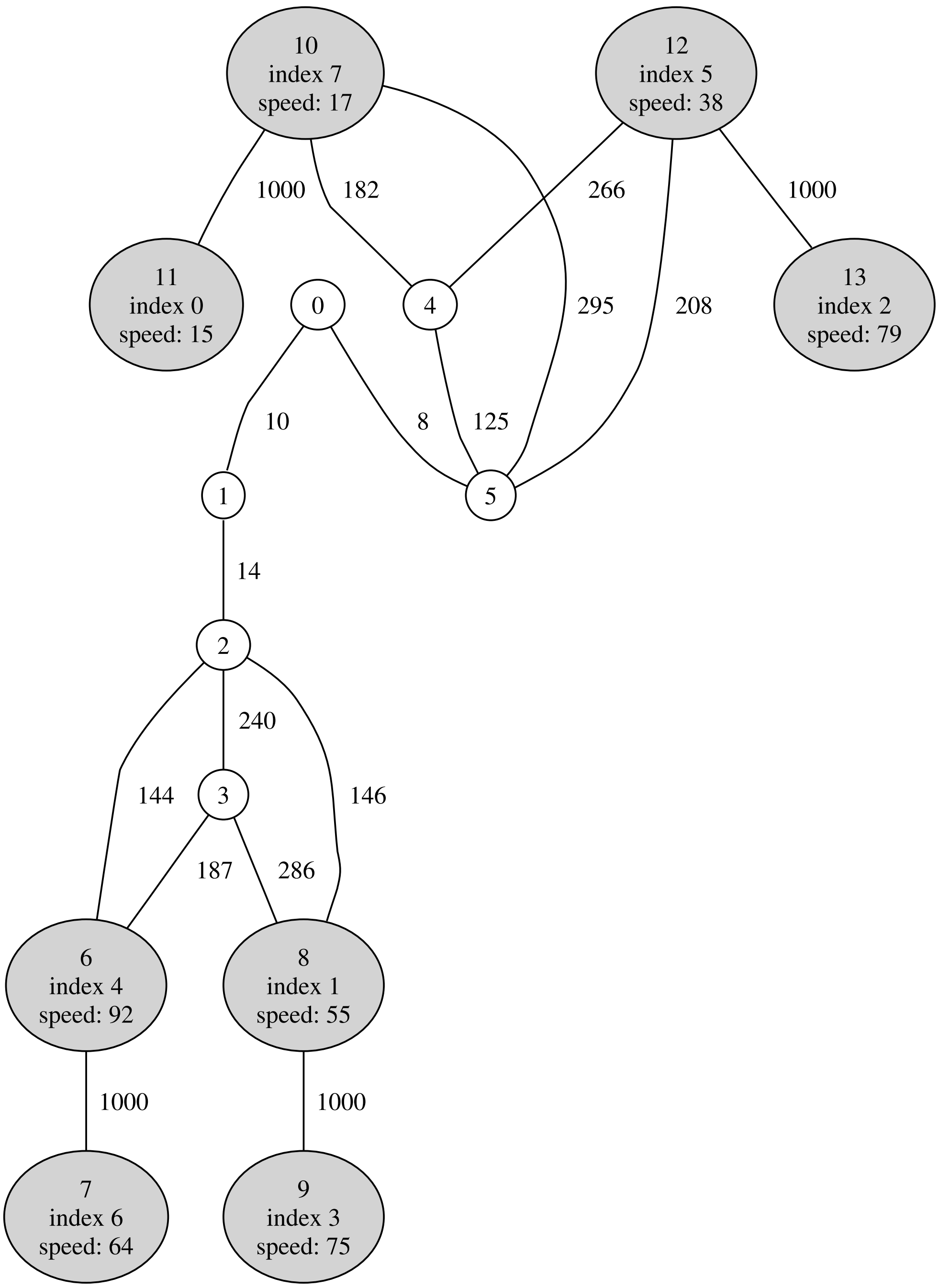


Figure 9: A complex topology, generated by Tiers. Each processor in gray has some value v_i to be reduced, and takes part in the computation. The logical index i of the processors is mentioned. The target node is node 6 (whose logical index is 4).

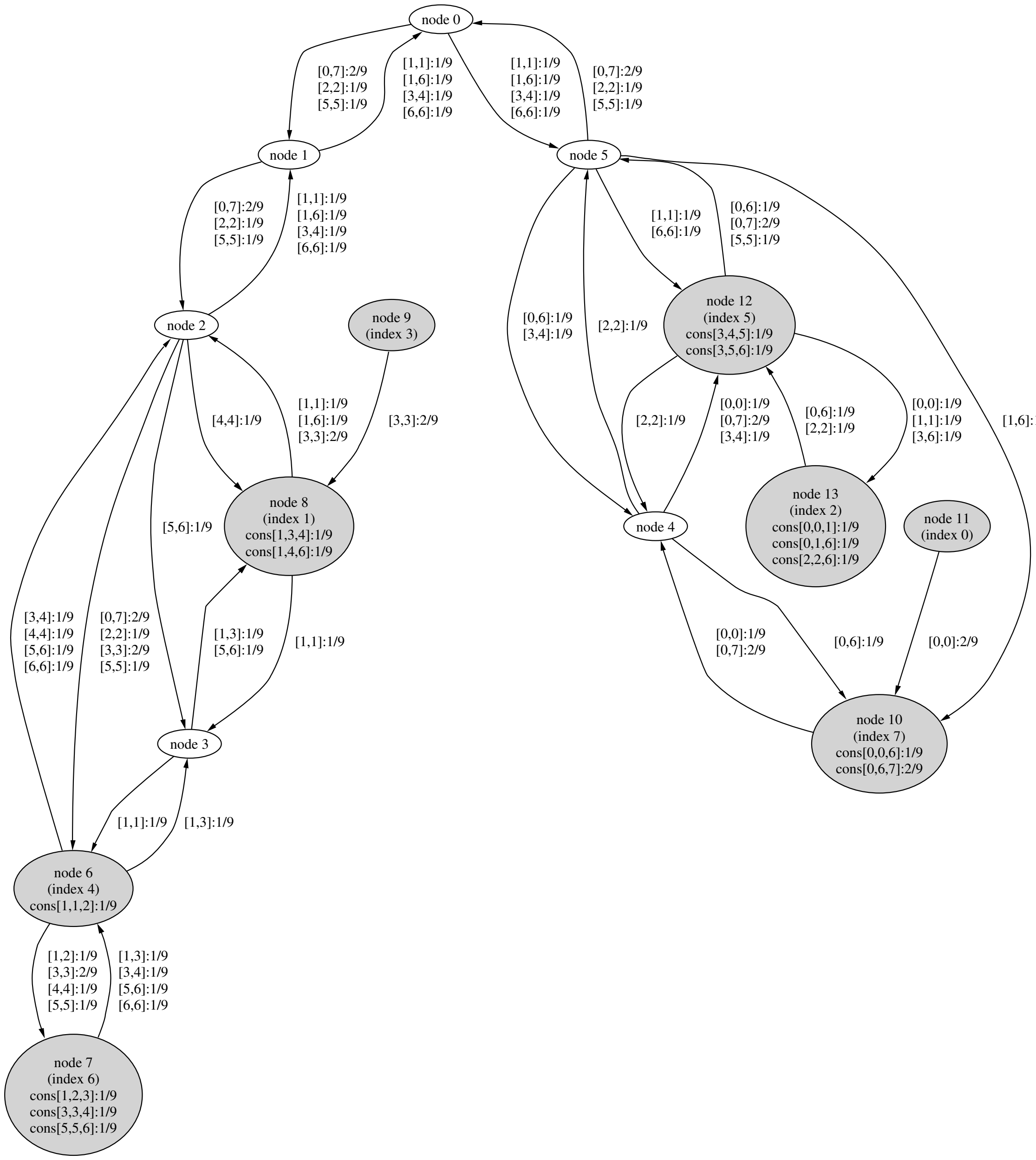


Figure 10: Results of the linear program. The target node is node 6 with index 4. Each link is labeled with the transfers scheduled through it during one time-unit. For example, $[1,6] : 1/9$ means that $1/9$ message of type $v_{[1,6]}$ pass through the edge during one time-unit. In the same way, the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

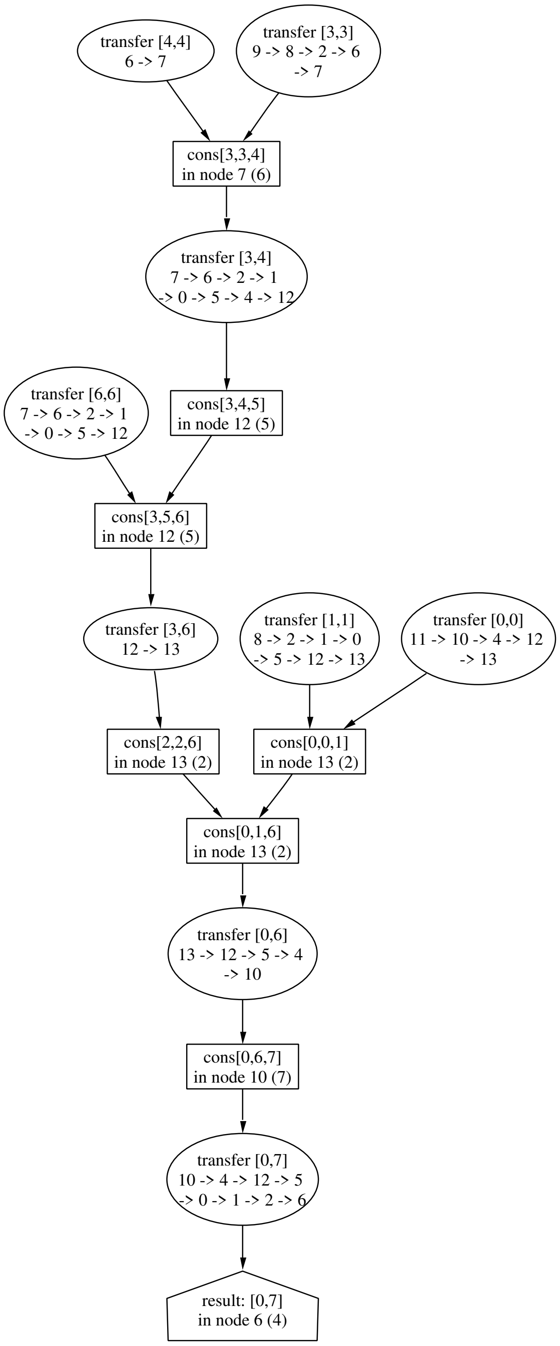


Figure 11: First reduction tree, with throughput 1/9 (= TP/2)

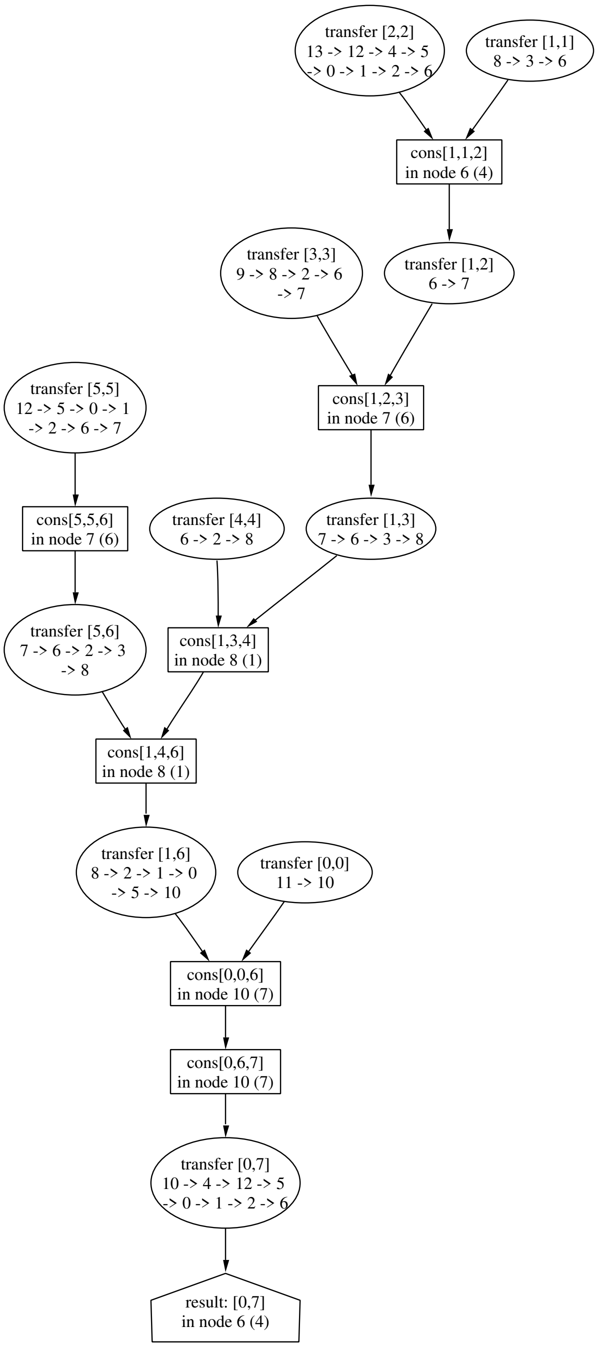


Figure 12: Second reduction tree, with throughput 1/9 (= TP/2)

5 Related Work

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

Models Several models have been considered in the literature:

- Banikazemi et al. [1] 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 [13, 19, 18].
- A more complex model is introduced in [2]: 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 [8, 7]: 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.

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 [25], 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. [3] 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 [3], 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, gossiping and gather/reduce have been studied in the context of heterogeneous platforms: see [21, 14, 20, 17, 22] among others.

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

- The common standard MPI [24] describes many collective communications, such as BROADCAST, GATHER, ALLTOALL, and REDUCE.
- A recent implementation, called MPICH-G2 [15], 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 [21] 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 [16] 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 [16]: 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.

6 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, gossips 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).

An interesting problem is to extend the solution for reduce operations to 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.

References

[1] M. Banikazemi, V. Moorthy, and 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, 1998.

[2] M. Banikazemi, J. Sampathkumar, S. Prabhu, D.K. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *HCW'99, the 8th Heterogeneous Computing Workshop*, pages 125–133. IEEE Computer Society Press, 1999.

[3] M. Barnett, R. Littlefield, D. G. Payne, and R. Van de Geijn. Global combine algorithms for 2-D meshes with wormhole routing. *J. Parallel and Distributed Computing*, 24(2):191–201, 1995.

[4] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Optimal algorithms for the pipelined scheduling of task graphs on heterogeneous systems. Technical report, LIP, ENS Lyon, France, April 2003.

[5] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining broadcasts on heterogeneous platforms. Technical report, LIP, ENS Lyon, France, June 2003.

[6] Michel Berkelaar. LP_SOLVE: Linear Programming Code. URL: `http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml`.

[7] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 59(2):252–279, 1999.

[8] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.

[9] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at `http://citeseer.nj.nec.com/calvert97modeling.html`.

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

[11] J. Reif (editor). *Synthesis of Parallel Algorithms*. Morgan Kaufmann, 1993.

[12] The MuPAD Group (Benno Fuchssteiner et al.). *MuPAD User's Manual*. John Wiley and sons, 1996.

[13] N.G. Hall, W.-P. Liu, and J.B. Sidney. Scheduling in broadcast networks. *Networks*, 32(14):233–253, 1998.

[14] J.-I. Hatta and 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, 2000.

[15] N. T. Karohis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *J.Parallel and Distributed Computing*, 63(5):551–563, 2003.

[16] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjan. 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*, pages 131–140, Atlanta, GA, 1999.

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

[18] P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[19] P. Liu and T.-H. Sheng. Broadcast scheduling optimization for heterogeneous cluster systems. In *SPAA'2000, 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 129–136. ACM Press, 2000.

[20] P. Liu and D.-W. Wang. Reduction optimization in heterogeneous cluster environments. In *14th International Parallel and Distributed Processing Symposium (IPDPS'2000)*. IEEE Computer Society Press, 2000.

[21] B. Lowekamp and 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, 1996.

[22] F. Ooshita, S. Matsumae, and 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, 2002.

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

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

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