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

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

Abstract

In this paper, we consider the communications involved by the execution of a complex application, deployed on a heterogeneous "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 a 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 reduce 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.

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. Here are the definitions of these operations:

Scatter One processor P_{source} has to send a distinct message to each target processor $P_{t_0}, \ldots, P_{t_N}$.

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 $\{P_{t_0}, \ldots, P_{t_N}\}$.

Reduce Each processor P_i among the set $P_{r_0}, \ldots, P_{r_N}$ 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 latter 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

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)
IEEE Computer Society
0-7695-2132-0/04/$17.00 (C) 2004 IEEE

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. Finally, we state some concluding remarks in Section 5.

Due to lack of space, we do not survey related work: instead, we refer to the extended version [7] of this paper.

2. Framework

We adopt a model of heterogeneity close to the one developed by Bhat, Raghavendra and Prasanna [3]. 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 a unit-size message along the edge.

Among different scenarios found in the literature, 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 $t + m \times c(i, j)$: (i) processor P_i cannot initiate another send operation (but it can perform a receive operation and an independent computation), (ii) processor P_j cannot initiate another receive operation (but it can perform a send operation and an independent computation), (iii) 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, 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:

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

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

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:

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

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

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. This idea is expressed by the following constraint:

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

Moreover, let us define the throughput at processor P_k as 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:

$$\forall P_k, k \in T, \sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = \text{TP} \quad (6)$$

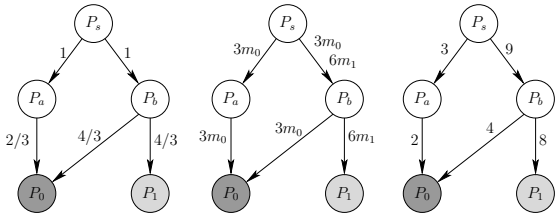
We can summarize the previous constraints in a linear program:

Steady-State Scatter Problem on a Graph
SSSP(G)
Maximize TP,
subject to
$\forall P_i, \forall P_j, 0 \leqslant s(P_i \to P_j) \leqslant 1$
$\forall P_i, \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1$
$\forall P_i, \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1$
$\forall P_i, P_j, s(P_i \to P_j) = \sum_{m_k} send(P_i \to P_j, m_k) \times c(i,j)$
$\forall P_i, \forall m_k, k \neq i, \sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_k)$
$= \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_k)$
$\forall P_k, k \in T \sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = \text{TP}$

This linear program can be solved in polynomial time by using tools like `lpsolve`[2], Maple [4] or MuPaD [6]. 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 4.5 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 1. Figure 1(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.



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

Figure 1. 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.

Figures 1(b) and 1(c) show the results of the linear program: on Figure 1(b) we represent the number of messages of each type going through the network, whereas Figure 1(c) describes the occupation of each edge.

The throughput achieved with this solution is 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.

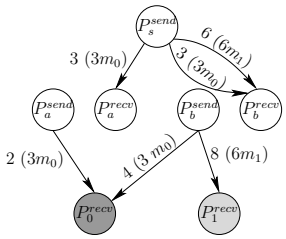


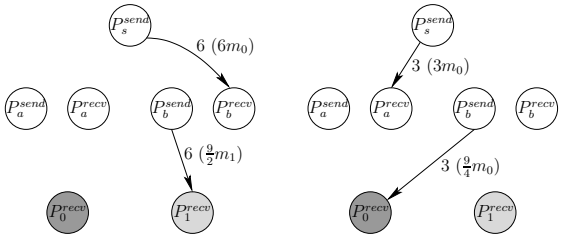
Figure 2. Bipartite Graph

3.3. Building a schedule

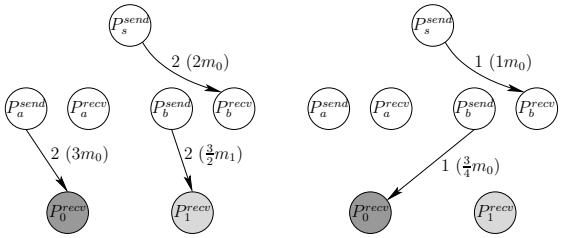
Once the linear program is solved, we get the period T of the schedule and the integer number of messages going through each link. We still need to exhibit a schedule of the message transfers where emissions (resp. receptions) never overlap on one node. This is done using a weighted-matching algorithm, as explained in [1]. 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.

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



(a) Matching 1 (b) Matching 2



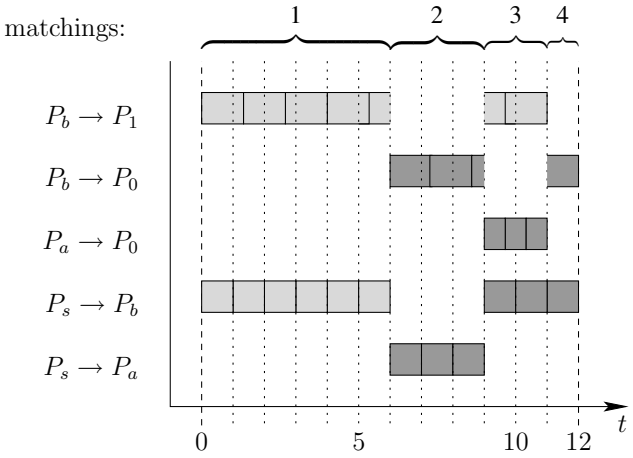
(c) Matching 3 (d) Matching 4

Figure 3. Decomposition of the bipartite graph 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.

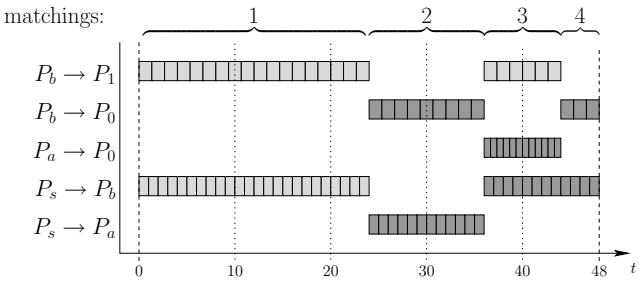
- for each transfer $send(P_i \rightarrow P_j, m_k)$, insert an edge between P_i^{send} and P_j^{recv} labeled with the time needed by the transfer: $send(P_i \rightarrow P_j, m_k) \times c(i,j)$.

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 [8, vol.A chapter 20] provides in polynomial time a polynomial number of matchings, which 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 2. It can be decomposed into four matchings, represented on Figures 3(a) to 3(d).

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 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 Figure 4(a).



(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.

3.4. Asymptotic optimality

In this section, we state that the previous periodic schedule is asymptotically optimal: basically, no scheduling algorithm (even non periodic) can execute more scatter operations in a given time-frame than ours, up to a constant number of operations. This section contains the formal statement of this result, whose proof is available in [7].

Given a platform graph $G = (V, E, c)$, a source processor P_{source} holding an infinite num-

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

ber 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 $\mathrm{SSSP(G)}$ of Section 3.1 applied to this platform graph G. We have the following result:

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

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. This algorithm is fully described in [7]. Let $steady(G, K)$ denote the number of reduction operations performed in time K with this algorithm. We state that $steady(G, K) \geqslant K \times \mathrm{P} - A$, where A does not depend on K, which leads to following result:

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

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

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

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 $P_{r_0}, \ldots, P_{r_N}$ own a value $v_0, \ldots, v_N$. The goal is to compute the reduction of these values: $v = v_0 \oplus \cdots \oplus v_N$, 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 [5] 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 $v_k, \ldots, v_m$:

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

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 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

We call *reduction tree* a tree representing how a reduction is done: the nodes represent the task (either computation or communication) located on the

[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.

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

resources (node or link), and there is an edge $n_i \to n_j$ is the result of task n_i is used as a data of task n_j. Figure 5 presents the example of a simple reduction tree.

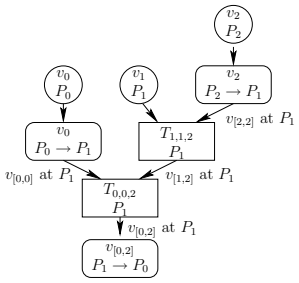


Figure 5. Simple example of a reduction tree

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_{r_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 $\{v_0^t, \ldots, v_N^t\}$ 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 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 as follows:

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

- $size(v_{[k,l]})$ 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:

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

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

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

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:

$\forall P_i, \forall v_{[k,m]}$ with (if there exists i' such that $P_{r_{i'}} = P_i$, $k \neq i'$ or $m \neq i'$) and ($P_{target} \neq P_i$ or $k \neq 0$ or $m \neq N$)

$$\sum_{P_j,(j,i)\in E} send(P_j \to P_i, v_{[k,m]}) + \sum_{k\leqslant l<m} cons(P_i, T_{k,l,m}) = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, v_{[k,m]}) + \sum_{n>m} cons(P_i, T_{k,m,n}) + \sum_{n<k} cons(P_i, T_{n,k-1,m}) \tag{10}$$

Note that this equation is not verified for the message $v_{[i',i']}$ on processor $P_{r_{i'}}$ (we assume we have an unlimited number of such messages). It is is also not verified for the final complete message $v = v_{[0,N]}$ 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:

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

If we summarize all these constraints, we come to a linear program, called STEADY-STATE REDUCE PROBLEM ON A GRAPH(G), whose objective function is to maximize TP under the constraints described in Equations 1, 2, 3, 7, 8, 9, 10 and 11.

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.

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

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

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. 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 $v_0^t, \ldots, v_{N-1}^t$. A given tree $\mathcal{T}$ might be used by many time-stamps t.

We can get a description of a schedule more compact than the extensive description of all trees using a family of trees weighted by the throughput of each tree. In [7], we provide an algorithm which extracts from the solution a set of weighted trees such that: 1) the weights are integers, 2) the weighted sum of the trees is equal to the original solution, and 3) the number of trees (and their description) is polynomial in the size of the topology graph G. Once this decomposition obtained, we use the same approach as for the scatter operation, based on a weighted-matching algorithm, to build a valid schedule. 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:
  (throughput of the tree) $\times$ $size(v_{[k,m]}) \times c(i,j)$.

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.

4.4. Asymptotic optimality

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

[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.

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

4.5. 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 and its decomposition into a set of weighted reduction trees $\{\mathcal{T}, weight(\mathcal{T})\}$. We compute the following values:

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

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 new throughput $\mathrm{TP}^* = \frac{1}{T_{fixed}} \times \sum_{\mathcal{T} \in \mathrm{Trees}} r(\mathcal{T})$ of the approximated solution and the original throughput TP:

$$\begin{aligned}
&\mathrm{TP} - \mathrm{TP}^* \\
&\quad = \mathrm{TP} - \textstyle\sum_{\mathcal{T} \in \mathrm{Trees}} \frac{1}{T_{fixed}} \times \left\lfloor \frac{weight(\mathcal{T})}{T} \times T_{fixed} \right\rfloor \\
&\quad \leqslant \mathrm{TP} - \textstyle\sum_{\mathcal{T} \in \mathrm{Trees}} \frac{1}{T_{fixed}} \times \left(\frac{weight(\mathcal{T})}{T} \times T_{fixed} - 1 \right) \\
&\quad \leqslant \frac{card(\mathrm{Trees})}{T_{fixed}}
\end{aligned}$$

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

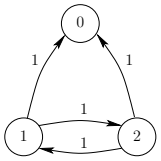
Proposition 4. *We can derive a steady-state operation for periods of arbitrary length, whose throughput converges to the optimal solution as the period size increases.*

4.6. Example and experimental results

In this section, we work out a complete example. Figure 6(a) shows 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. The solution of the linear program is described on Figure 6(b), and mapped on the topology graph on Figure 6(c). The exhaustive description of

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

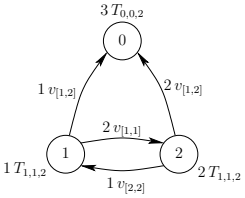
a valid schedule using the values given in 6(c). Three reductions are performed every three time-units. The values reduced are labeled with their time-stamp (upper index). Figure 6(d) shows the schedule of one period. This schedule can be pipelined on a period of three time-units, leading to a throughput of one reduce operation per time-unit.



(a) Topology

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

(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

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

In the extended version of the paper [7], we also present an example where the topology graph is obtained from a topology generator. We notice that in the obtained schedule, one reduction tree is not sufficient to come with the optimal throughput.

5. 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 processors whose rank is lower than its own rank.

References

[1] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Optimal algorithms for the pipelined scheduling of task graphs on heterogeneous systems. Technical Report RR-2003-29, LIP, ENS Lyon, France, April 2003.

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

[3] 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.

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

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

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

[7] A. Legrand, L. Marchal, and 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.

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

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