Laboratoire de l'Informatique du Parallélisme

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

Broadcast Trees for Heterogeneous Platforms

Olivier Beaumont,
Loris Marchal,
Yves Robert

November 2004

Research Report N° 2004-46

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`

Broadcast Trees for Heterogeneous Platforms

Olivier Beaumont, Loris Marchal, Yves Robert

November 2004

Abstract

In this paper, we deal with broadcasting on heterogeneous platforms. Typically, the message to be broadcast is split into several slices, which are sent by the source processor in a pipeline fashion. A spanning tree is used to implement this operation, and the objective is to find the tree which maximizes the throughput, i.e. the average number of slices sent by the source processor every time-unit. We introduce several heuristics to solve this problem. The good news is that the best heuristics perform quite efficiently, reaching more than 70% of the absolute optimal throughput, thereby providing a simple yet efficient approach to achieve very good performance for broadcasting on heterogeneous platforms.

Keywords: Collective communications, steady-state, heterogeneous platforms, modeling

Résumé

Dans ce rapport, nous considérons le problème de la diffusion de messages sur plates-formes hétérogènes. En particulier, nous considérons le cas où le message à envoyer, de grande taille, est fractionné en plusieurs morceaux qui sont envoyés par le processeur à la source de la diffusion de façon successive et pipelinée. On utilise un arbre de diffusion pour implémenter cette opération, et le but est de trouver l'arbre de diffusion qui permet d'obtenir le meilleur débit. Nous introduisons plusieurs heuristiques pour résoudre ce problème. La meilleure de ces heuristiques atteint un débit de plus de 70% de l'optimal, et fournit ainsi une approche simple mais efficace pour obtenir de bonnes performances pour la diffusion sur plates-formes hétérogènes.

Mots-clés: Communications collectives, régime permanent, plates-formes hétérogènes, modélisation

STA	Single Tree, Atomic	Only for small messages
STP	Single Tree, Pipelined	Allows pipelining along the tree
MTP	Multiple Tree, Pipelined	Powerful, but difficult to design/implement

Table 1: Summary of the three approaches for broadcasting.

1 Introduction

Broadcasting in computer networks is the focus of a vast literature. The one-to-all broadcast, or single-node broadcast [23], is the most primary collective communication pattern: initially, only the source processor has the data that needs to be broadcast; at the end, there is a copy of the original data residing at each processor. Parallel applications and algorithms often require to send identical data to all other processors, in order to disseminate global information (typically, input data such as the problem size or application parameters). Numerous broadcast algorithms have been designed for parallel machines such as meshes, hypercubes, and variants (see among others [18, 39, 36, 22, 38]). The *MPI_Bcast* routine [33] is widely used, and particular care has been given to its efficient implementation on a large variety of platforms [17].

For short-size broadcasts, a single message is sent by the source processor, and forwarded across the network. A spanning tree is used to implement this operation. However, finding the best spanning tree, i.e. the tree which minimizes the total execution time of the broadcast, is a difficult problem; it turns out NP-complete even for the uttermost basic telephone model (problem ND49 in [14]).

For broadcasting larger messages, pipelining strategies are mandatory to optimize the total execution time. At the application level, the source message is split into a number of slices, which are routed in a pipelined fashion from the source processor to all other nodes[1]. There is more freedom here: either we decide to route all the slices along the same spanning tree, or we use several spanning trees simultaneously. In the latter case, each tree is used to broadcast a distinct fraction of the total message. Each message fraction will itself be divided into slices, to be sent in a pipeline fashion along the corresponding tree. Of course the implementation becomes more complex: the communications along the different trees must be orchestrated so as to take resource conflicts into account (e.g., if two trees share a physical link, the slices from both trees have to share the available bandwidth).

We summarize the three previous approaches, labeled STA, STP and MTP, in Table 1. Numerous heuristics are available in the literature for the STA problem, such as *Fastest Node First* [1] and *Fastest Edge First* [9]. For the STA problem, there is a single message to broadcast, and the objective is to find a tree that minimizes the total execution time (the makespan). For the STP problem, there is a large number of message slices to broadcast, and the objective is to find a tree that maximizes the throughput, i.e. the average number of slices sent by the source processor every time-unit. The problem of throughput maximization can be viewed as a relaxation of the problem of makespan minimization, because the initialization and clean-up phases are ignored. Still, the problem of throughput maximization remains NP-hard (see problem ND1 in [14] and the reduction in [6]). To the best of our knowledge, little work has been conducted on the design and experimental evaluation of polynomial heuristics

[1]Note that a message slice can itself be further divided into packets by the system or network layer, but (i) this is transparent to the user and (ii) this is accounted for by using an affine cost model for communications, see Section 2.

for the STP problem. One major goal of this paper is to fill the void.

At first sight, the MTP problem looks more complicated than the STP problem: finding a set of trees that can be used simultaneously, without link conflict, to maximize the total throughput that can be achieved, seems more difficult than finding a single tree. Surprisingly, the optimal solution to the MTP problem can be computed in in polynomial time [5, 6]. However, the latter result is mostly of theoretical interest, because it requires a very complicated algorithm to extract the set of spanning trees that achieves the optimal throughput. Even after deriving the set of trees, it would be quite time-consuming to implement the fractioning of the original message, and the reconstruction of the solution at every node. However, the first step of the approach is both simple and fast: it consists in computing the optimal throughput that can be achieved (only the value of the throughput, not the trees needed to achieve it) by solving a linear program over the rationals, using standard tools such as Maple [10] or MuPad [35]. This is of great practical interest, because the knowledge of the optimal throughput enables to quantify the absolute performance of the STP heuristics, and to assess how far they are from the optimal[2]. The good news is that the best STP heuristics reach around 70% of the optimal, thereby providing a simple yet efficient way to achieve very good performance for broadcasting on heterogeneous platforms.

The rest of the paper is organized as follows. First, Section 2 reviews several platform models, and provides the framework that will be used throughout the paper. In Section 3, we introduce a first set of heuristics for the STP problem (i.e. using a single spanning tree for pipelining the broadcast message). These heuristics are derived from classical graph algorithms. Next, in Section 4, we briefly recall the linear program computing the optimal solution for the MTP problem. We use the solution of this linear program to construct a second set of heuristics for the STP problem. We report some experimental data to compare all these heuristics in Section 5. We briefly survey related work in Section 6. Finally, we state some concluding remarks in Section 7.

2 Models and framework

In this section, we review several platform models to implement a broadcast operation. The models mainly differ by the capability (or not) for a node to perform several communications simultaneously.

2.1 Models

The target architectural platform is represented by a directed graph $\mathcal{P} = (V, E)$. Note that this graph may well include cycles and multiple paths. For the sake of generality, we assume that the graph is directed, so that all links are unidirectional (but using two opposite edges would model a bidirectional link).

Consider two adjacent processors P_u and P_v in the graph (hence the link $e_{u,v} : P_u \rightarrow P_v$ belongs to the set of physical links E). Assume that P_u sends a message of size L to P_v. There are several models in the literature, which we summarize through the general scenario depicted in Figure 1.

Here are some notations to analyze the communication from P_u to P_v:

[2]Recall that computing the optimal solution for the STP problem requires exponential time, unless P=NP. This explains the detour via MTP.

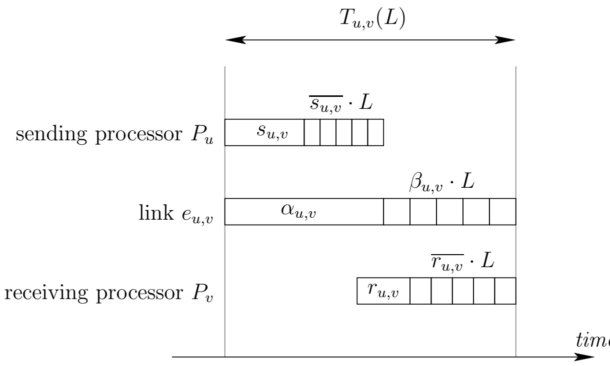


Figure 1: Sending a message of size L along the link $e_{u,v} : P_u \to P_v$

- The time where P_u is busy sending the message is expressed as an affine function $send_{u,v}(L) = s_{u,v} + L \cdot \overline{s_{u,v}}$. The start-up time $s_{u,v}$ corresponds to the software and hardware overhead paid to initiate the communication. The link capacity $\overline{s_{u,v}}$ corresponds to the inverse of the transfer rate which can be achieved (say, from main memory of P_u to a network card able to buffer the message).

- Similarly, the time where P_v is busy receiving the message is expressed as an affine function of the message length L, namely $recv_{u,v}(L) = r_{u,v} + L \cdot \overline{r_{u,v}}$.

- The total time for the communication, which corresponds to the total occupation time of the link $e_{u,v} : P_u \to P_v$, is also expressed as an affine function $T_{u,v}(L) = \alpha_{u,v} + L \cdot \beta_{u,v}$. The parameters $\alpha_{u,v}$ and $\beta_{u,v}$ correspond respectively to the start-up cost and to the inverse of the link bandwidth.

Simpler models do not make the distinction between the three occupation times $send_{u,v}(L)$ (emission by P_u), $T_{u,v}(L)$ (occupation of $e_{u,v} : P_u \to P_v$) and $recv_{u,v}(L)$ (reception by P_v). Such models use $s_{u,v} = r_{u,v} = \alpha_{u,v}$ and $\overline{s_{u,v}} = \overline{r_{u,v}} = \beta_{u,v}$. This amounts to assume that the sender P_u and the receiver P_v are blocked throughout the communication. In particular, P_u cannot send any message to another neighbor P_w during $T_{u,v}(L)$ time-steps. However, some system/platform combinations may allow P_u to proceed to another send operation before the entire message has been received by P_v. To account for this situation, more complex models would use different functions for $send_{u,v}(L)$ and $T_{u,v}(L)$, with the obvious condition that $send_{u,v}(L) \leqslant T_{u,v}(L)$ for all message sizes L (this implies $s_{u,v} \leqslant \alpha_{u,v}$ and $\overline{s_{u,v}} \leqslant \beta_{u,v}$). Similarly, P_v may be involved only at the end of the communication, during a time period $recv_{u,v}(L)$ smaller than $T_{u,v}(L)$.

Here is a summary of the general framework: assume that P_u initiates a communication of size L to P_v at time-step $t = 0$:

- Link $e_{u,v} : P_u \to P_v$ is busy from $t = 0$ to $t = T_{u,v}(L) = \alpha_{u,v} + L \cdot \beta_{u,v}$

- Processor P_u is busy from $t = 0$ to $t = send_{u,v}(L) = s_{u,v} + L \cdot \overline{s_{u,v}}$, where $s_{u,v} \leqslant \alpha_{u,v}$ and $\overline{s_{u,v}} \leqslant \beta_{u,v}$

- Processor P_v is busy from $t = T_{u,v}(L) - recv_{u,v}(L)$ to $t = T_{u,v}(L)$, where $recv_{u,v}(L) = r_{u,v} + L \cdot \overline{r_{u,v}}$, $r_{u,v} \leqslant \alpha_{u,v}$ and $\overline{r_{u,v}} \leqslant \beta_{u,v}$

In the following, we review some models that have been introduced in the literature. *Multi-port* models allow parallel sends (and parallel receive) while *One-port* models assume that a sending processor is blocked throughout the communication.

2.2 Multi-port models

Banikazemi et al [2] propose a model which is very close to the general model presented above. They use affine functions to model the occupation time of the processors and of the communication link. The only minor difference is that they assume that the time intervals where P_u is busy sending (of duration $send_{u,v}(L)$) and where P_v is busy receiving (of duration $recv_{u,v}(L)$) do not overlap, so that they write

$$T_{u,v}(L) = send_{u,v}(L) + link_{u,v}(L) + recv_{u,v}(L).$$

In [2] a methodology is proposed to instantiate the six parameters of the affine functions $send_{u,v}(L)$, $link_{u,v}(L)$ and $recv_{u,v}(L)$ on a heterogeneous platform. The authors point out that these parameters actually differ for each processor pair and depend upon the CPU speeds.

A simplified version of the general model has been proposed by Bar-Noy et al [3]. In this variant, the time during which an emitting processor P_u is blocked does not depend upon the receiver P_v (and similarly the blocking time in reception does not depend upon the sender. In addition, only fixed-size messages are considered in [3], so that this model writes

$$T_{u,v} = send_u + link_{u,v} + recv_v \tag{1}$$

The models of Banikazemi et al [2] and Bar-Noy et al [3] are called *multi-port* because they allow a sending processor to initiate another communication while a previous one is still on-going on the network. However, both models insist that there is an overhead time to pay before being engaged in another operation, so there are not allowing for fully simultaneous communications.

2.3 One-port models

These models come in two variants. In the unidirectional variant, a processor cannot be involved in more than one communication at a given time-step, either a send or a receive. In the bidirectional model, a processor can send and receive in parallel, but at most to a given neighbor in each direction. In both variants, if P_u sends a message to P_v, both P_u and P_v are blocked throughout the communication: with previous notations $s_{u,v} = r_{u,v} = \alpha_{u,v}$ and $\overline{s_{u,v}} = \overline{r_{u,v}} = \beta_{u,v}$.

The bidirectional one-port model is used by Bhat et al [8, 9] for fixed-size messages. They advocate its use because "current hardware and software do not easily enable multiple messages to be transmitted simultaneously". Even if non-blocking multi-threaded communication libraries allow for initiating multiple send and receive operations, they claim that all these operations "are eventually serialized by the single hardware port to the network". Experimental evidence of this fact has recently been reported by Saif and Parashar [32], who report that asynchronous MPI sends get serialized as soon as message sizes exceed a few megabytes. Their result hold for two popular MPI implementations, MPICH on Linux clusters and IBM MPI on the SP2.

The one-port model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. It generalizes a simpler model studied by Banikazemi et al. [1]

Liu [25] and Khuller and Kim [21]. In this simpler model, the communication time $T_{u,v}(L)$ only depends on the sender, not on the receiver: in other words, the communication speed from a processor to all its neighbors is the same.

2.4 Framework

Let $\mathcal{P} = (V, E)$ be the platform graph, and $p = |V|$ be the number of nodes. The source node P_{source} initially holds all the data to be broadcast. All the other nodes P_u, $1 \leqslant u \leqslant p, u \neq s$, are destination nodes which must receive all the data sent by P_{source}. We assume that the total size of the data to be broadcast is large, say from a few megabytes to larger values.

As discussed in Section 1, a natural strategy is to pipeline the broadcast along a single spanning tree. At the application level, the large message will be split into several slices which will be broadcast consecutively, in a pipeline fashion, so that first slices will reach the leaves of the tree while the source node is still emitting the last slices. The usefulness of pipelining a large number of slices has been demonstrated by van de Geijn et al. [37, 4] when communicating over LANs et WANs. Including such pipelining strategies in MPICH-G2 if the focus of on-going work [20].

We let L denote the size of a slice, which should be set at the application level. When the value of L has been fixed, we have a series of same-size messages to be broadcast consecutively by P_{source}. The objective is to find a spanning tree with good *throughput*, where the throughput is defined as the average number of message slices sent by P_{source} every time-unit. Because the number of slices is assumed to be large, we can safely neglect the initialization and clean-up phases: as soon as the first slices are circulated along the tree, every node operates in steady-state.

Following the previous discussion, we mainly use the (bidirectional) one-port model to implement the broadcast operation. However, we will extend some of the heuristics to the multi-port model.

3 Platform-based heuristics

In this section, we describe several heuristics for the STP problem: given a platform graph and a source processor, we aim at finding a "good" broadcast tree, that is a tree where messages can be sent in a pipelined fashion with a good throughput.

3.1 One-port model

We use here the (bidirectional) one-port model, where a processor can be involved in only one sending (and one receiving) operation. We consider the pipelined broadcast of a message divided into slices of same size L. For every link $e_{u,v} : P_u \rightarrow P_v$, we write:

$$send_{u,v}(L) = recv_{u,v}(L) = T_{u,v}(L) = T_{u,v}.$$

The edges of platform graph $\mathcal{P} = (V, E)$ are weighted by the time needed to send a message of size L: the weight of $e_{u,v} : P_u \rightarrow P_v$ is $T_{u,v}$. We design four heuristics. The first two heuristics start from the platform graph and delete edges until the resulting graph is a tree, while the third heuristic grows a spanning tree rooted at the source processor. The fourth one, based on MPI policy to broadcast a message, is included for sake of comparison with existing broadcast techniques.

3.1.1 Simple Platform Pruning

The idea of this simple heuristic is to prune the platform graph, deleting edges with maximum weight, until we obtain a tree spanning all the nodes: see Algorithm 1.

```
SIMPLE-PLATFORM-PRUNING(𝒫, P_source)
  TreeEdges ← all edges of E
  while |TreeEdges| > n − 1 do
    L ← edges of TreeEdges sorted by non-increasing weight T_{u,v}
    for each edge e ∈ L do
      if the graph (V, TreeEdges\{e}) is still connected then
        TreeEdges ← TreeEdges\{e}
  return (V, TreeEdges)
```

Algorithm 1: The simple platform pruning algorithm

3.1.2 Refined Platform Pruning

If we look carefully at the previous heuristic, we realize that there is no reason to discard all edges with large weight: if a node has a many children in the tree (say, 10), with all its outgoing edges of medium weight (e.g. 2), it will spend 20 time-units to broadcast each message slice. On the contrary, a node linked to a single child in a tree by an edge of larger weight (say 15) will only need 15 time-units broadcast each message slice. The throughput of each node is inversely proportional to its weighted outgoing degree, i.e. the sum of the weights of its outgoing edges in the tree. A more accurate metric in the heuristic would be the weighted out-degree of a node in the tree rather than the maximum weight of all edges.

We can adapt the previous heuristic to this metric by maintaining the current weighted out-degree of each node u, denoted as $OutDegree(u)$, and trying to delete an edge from the node which maximizes this metric. This is summarized in Algorithm 2.

```
REFINED-PLATFORM-PRUNING(𝒫, P_source)
 1: TreeEdges ← all edges of E
 2: for each u ∈ V do
 3:   OutDegree(u) ← Σ_{v, (u,v)∈E} T_{u,v}
 4: while |TreeEdges| > n − 1 do
 5:   SortedNodes ← nodes sorted by non-increasing value of OutDegree(u)
 6:   for u ∈ SortedNodes do
 7:     L ← edges sorted by decreasing weight T_{u,v}
 8:     for each edge e = (u, v) ∈ L do
 9:       if the graph (V, TreeEdges\{e}) is still connected then
10:         TreeEdges ← TreeEdges\{e}
11:         OutDegree(u) ← OutDegree(u) − T_{u,v}
12:         goto 4
13: return (V, TreeEdges)
```

Algorithm 2: The refined platform pruning algorithm.

3.1.3 Growing a Minimum Weighted Out-Degree Tree

This heuristic is derived from Prim's algorithm [11] for building a minimum cost spanning tree. The usual metric for the cost of the tree is the sum of all its edges. However, as discussed in the previous heuristic, this is not the good metric for our problem. Instead, we are interested in minimizing the maximum weighted out-degree of each node in the tree. We can adapt Prim's algorithm as shown in Algorithm 3.

When we add a new edge (u, v) in the tree, we update the cost of edges (u, w), for all neighbors w of u not already in the tree. The cost of an edge (u, w) (with P_u in the tree and P_w not yet in the tree) is defined as the sum of the weights of the current tree edges outgoing from P_u. By selecting the edge with minimum cost, we add the edge which increases as little as possible the maximum weighted out-degree of any node in the tree.

```
GROWING-MINIMUM-WEIGHTED-OUT-DEGREE-TREE(𝒫, P_source)
  TreeEdges ← ∅
  TreeVertices ← {P_source}
  for each edge e = (u, v) do
    cost(u, v) ← T_{u,v}
  while TreeVertices ≠ V do
    choose the link (u, v) such that u ∈ TreeVertices, v ∉ TreeVertices and (u, v)
    has minimum value cost(u, v)
    TreeVertices ← TreeVertices ∪ {v}
    TreeEdges ← TreeEdges ∪ {(u, v)}
    for each edge (u, w) ∉ TreeEdges do
      cost(u, w) ← cost(u, w) + cost(u, v)
  return (TreeVertices, TreeEdges)
```

Algorithm 3: The growing tree algorithm.

3.1.4 Binomial tree heuristic

For the sake of comparison with existing strategies for the STP problem, we introduce another heuristic using a binomial tree. This heuristic is based on the classical MPI implementation of the broadcast [33], which constructs a binomial spanning tree based on the index of each processor, without any topological information. We assume here that the source has index 0, and we compute a binomial tree for the first 2^m nodes of the platform (where $m = \lfloor \log_2 |V| \rfloor$). Each of the remaining nodes x receives the message from one of the previous nodes $(x - 2^m)$ in the last stage of the tree construction. When adding a transfer from node u to node v, edge (u, v) may not exist; in this case, we schedule the transfer through the shortest path from u to v. This heuristic is described in Algorithm 4 (we suppose that SHORTEST-PATH(u,v) returns the edges of the shortest path from u to v).

3.2 Multi-port

The Growing-Minimum-Weighted-Out-degree-Tree heuristic can be adapted to the multi-port model. Recall that in this model, the occupation of the sending processor is less than the total occupation of the communication link. According to Equation 1, the time to transfer

BINOMIAL-TREE($\mathcal{P}, P_{source}$)
  $\mathit{TreeEdges} \leftarrow \emptyset$
  $m = \lfloor \log_2 |V| \rfloor$
  for $p = 0 \ldots m-1$ **do**
    for $X = 0 \ldots 2^p - 1$ **do**
      $\mathit{TreeEdges} \leftarrow \mathit{TreeEdges} \cup \text{SHORTEST-PATH}(X \cdot 2^{m-p}, X \cdot 2^{m-p} + 2^{m-p-1})$
  for $u = 2^m \ldots |V| - 1$ **do**
    $\mathit{TreeEdges} \leftarrow \mathit{TreeEdges} \cup \text{SHORTEST-PATH}(u, u - 2^m)$
  return $(V, \mathit{TreeEdges})$

Algorithm 4: The binomial tree algorithm.

a message from P_u to P_v is $T_{u,v} = send_u + link_{u,v} + recv_v$. If P_u initiates several sends, the first and third term $send_u$ and $recv_v$ have to be serialized, while the second term $link_{u,v}$ may be parallelized. This is represented in Figure 2.

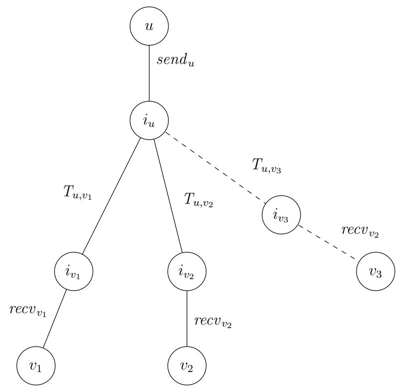


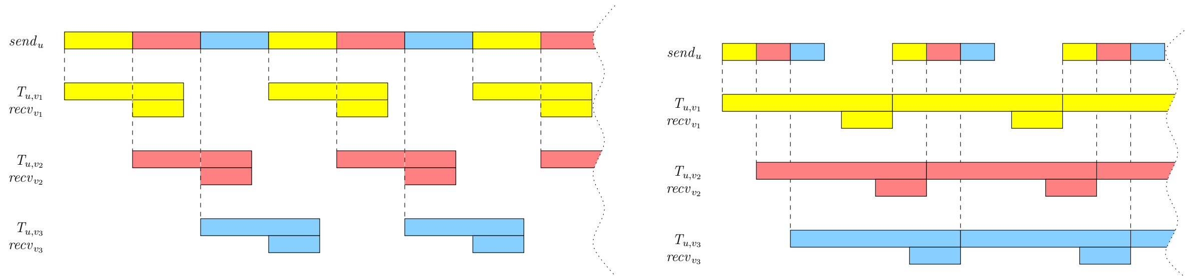
Figure 2: Tree under construction under the multiport model

In Figure 2, nodes $i_u, i_{v_1}, i_{v_2}, i_{v_3}$ represent network interfaces of nodes u, v_1, v_2 and v_3. In this example, we assume that the edges (u, v_1) and (u, v_2) are already included in the broadcast tree, and we study the impact of adding edge (u, v_3) in the tree. If we add this edge, P_u can send a new message slice to its three children in the tree every T_{period}, where $T_{period} = \max(3 \times r_u, T_{u,v_1}, T_{u,v_2}, T_{u,v_3})$. See Figure 3 for a graphical explanation of this formula.

In general, the throughput that a node P_u can achieve is the inverse of T_{period}, where

$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$

where $\delta_{out}(P_u)$ is the out-degree of u (the number of its children in the tree). Therefore, adapting the previous heuristic to the multi-port model simply amounts to change the cost of adding a new edge. This is described in Algorithm 5.



(a) Case where throughput is bounded by the serialized $send_u$.

(b) Case where throughput is bounded by the longest link occupation $T_{u,v}$.

Figure 3: Examples for the two cases determining T_{period}

MULTIPORT-GROWING-MINIMUM-WEIGHTED-OUT-DEGREE-TREE($\mathcal{P}, P_{source}$)

  $TreeEdges \leftarrow \emptyset$
  $TreeVertices \leftarrow \{P_{source}\}$
  for each $u \in V$ **do**
    $children(u) \leftarrow \emptyset$
  for all edge $e = (u, v)$ **do**
    $cost(u, v) \leftarrow T_{u,v}$
  while $TreeVertices \neq V$ **do**
    choose the link (u, v) such that $u \in TreeVertices$, $v \notin TreeVertices$ and (u, v) has minimum value $cost(u, v)$
    $TreeVertices \leftarrow TreeVertices \cup \{v\}$
    $TreeEdges \leftarrow TreeEdges \cup \{(u, v)\}$
    $children(u) \leftarrow children(u) \cup \{v\}$
    for each edge $(u, w) \notin TreeEdges$ **do**
      $cost(u, v) \leftarrow \max\left(\left|children[u]\right| * send_u \;,\; \max_{w' \in children[u]} (T_{u,w'})\right)$
  return $(TreeVertices, TreeEdges)$

Algorithm 5: The multiport growing tree algorithm

4 LP-based heuristics

4.1 Linear program

As pointed out in Section 1, the optimal throughput for the MTP problem is equal to the solution of the following linear program over the rationals (which we solve with standard tools such as Maple [10] or MuPad [35]:

$$
\begin{array}{l}
\textsc{Steady-State Broadcast Problem on a Graph SSB(G)}\\
\textsc{Maximize } \mathrm{TP},\\
\textsc{subject to}\\
\left\{\begin{array}{lll}
(a) & \forall u, & \sum_{P_v \in \mathcal{N}^{out}(P_{source})} x_u^{source,v} = \mathrm{TP}\\
(b) & \forall u,\ Pu \neq P_{source}, & \sum_{P_v \in \mathcal{N}^{in}(P_u)} x_u^{v,u} = \mathrm{TP}\\
(c) & \forall u,v,\ P_v \neq P_{source},\ P_v \neq P_u, & \sum_{P_w \in \mathcal{N}^{in}(P_v)} x_u^{w,v} = \sum_{P_w \in \mathcal{N}^{out}(P_v)} x_u^{v,w}\\
(d) & \forall e_{u,v} : P_u \to P_v, & n_{u,v} = \max_w x_w^{u,v}\\
(e) & \forall e_{u,v} : P_u \to P_v, & t_{u,v} = n_{u,v} \cdot T_{u,v}\\
(f) & \forall u, & t_u^{(in)} = \sum_{P_v \in \mathcal{N}^{in}(P_u)} t_{v,u}\\
(g) & \forall u, & t_u^{(out)} = \sum_{P_v \in \mathcal{N}^{out}(P_u)} t_{u,v}\\
(h) & \forall u,v, & t_{u,v} \leqslant 1\\
(i) & \forall u, & t_u^{(in)} \leqslant 1\\
(j) & \forall u, & t_u^{(out)} \leqslant 1
\end{array}\right.
\end{array}
\qquad (2)
$$

In the linear program (2), the objective function is the throughput TP. We let $\mathcal{N}^{out}(P_u)$ denote the set of the output neighbors of P_u, i.e. the set of nodes P_v such that $e_{u,v} : P_u \to P_v \in E$; similarly, $\mathcal{N}^{in}(P_v)$ is the set of the input neighbors of P_u, i.e. nodes P_v such that $e_{u,v} \in E$. Also, $x_w^{u,v}$ denotes the fractional number of message slices which are sent by P_{source} to P_w every time-unit, and that transit on the edge $e_{u,v} : P_u \to P_v$. Of course each processor will receive the same set of messages in the end, but the same message may well be forwarded along different trees to different destinations, hence the need for "tracking" the circulation of messages along each edge.

The first constraint (a) states that the total number of messages destined to P_u and which are sent from P_{source} every time-unit is indeed TP. Similarly, constraint (b) expresses that the total number of messages which are actually received by P_u every time-unit is also equal to TP. Constraint (c) states a conservation law at any intermediate processor P_v distinct from P_{source} and from P_u: the number of messages destined to P_u which arrive at P_v each time-unit is the same as the number of same type messages that go out of P_v. This conservation law is only valid in steady-state operation, it does not apply to the initialization and clean-up phases.

The following set of constraints is related to link occupation. First, $n_{u,v}$ denotes the total number of messages that transit on the edge $e_{u,v} : P_u \to P_v$. We know that for each w, there are $x_w^{u,v}$ messages sent to P_w which do transit on this edge. The main difficulty is that the sets of messages transiting on the edge and sent to different P_w's may partly overlap. If they were all disjoint, we would write $n_{u,v} = \sum_w x_w^{u,v}$ (this would hold true for a scatter operation). In fact, it turns out that is it possible to design a schedule such that $n_{u,v} = \max_w x_w^{u,v}$, which is precisely constraint (d). This means that is is possible to orchestrate the communications such that given any two message sets circulating on the same edge and destined to different processors, one is a subset of the other. The proof of this result is quite involved, see [6]. The next constraints are easier to derive. Constraint (e) expresses the time to circulate all messages on the edge $e_{u,v}$. The next two constraints correspond to the one-port model: constraint (f) ensures that all incoming communications are sequentialized, and constraint (g) is the counterpart for outgoing communications. The last three constraints simply state that any set of sequential communications lasts no more than one time-unit.

We do not use the complicated algorithm described in [5, 6] to design an actual schedule that achieves the optimal throughput TP through the parallel propagation of message slices along several spanning trees. We only compute the optimal solution of the linear program (2), and we retain the values of TP, the optimal throughput, and of $n_{u,v}$, the number of message slices that circulate along each edge $e_{u,v}$.

4.2 Heuristic for the one-port model

We assign the weight $n_{u,v}$ to each edge $e_{u,v}$ of the platform graph $G = (V, E)$, and we use these weights to design two heuristics for the STP problem. We call *communication graph* the platform graph weighted as just defined. The first heuristic starts from the whole communication graph and removes the edges carrying the smallest number of messages, until having reduced the graph to a spanning tree. The second heuristic works bottom-up, and grows a spanning tree starting from the source processor, using the the most "useful" edges.

4.2.1 Communication graph pruning

In this heuristic, we delete the edges which preserve the connectivity of the graph and have minimum weight, i.e. edges carrying the fewest messages in the solution returned by the linear program (2). This is described in Algorithm 6.

```
LP-PRUNE(𝒫, P_source)
   solve linear program (2), and compute n_{u,v}, the number of messages
   sent through edge (u,v) during one time-unit
   TreeEdges ← all edges of E
   while |TreeEdges| > n − 1 do
      L ← edges (u,v) sorted by non-increasing value of n_{u,v}
      for each edge e ∈ L do
         if the graph (V, TreeEdges\{e}) is still connected then
            TreeEdges ← TreeEdges\{e}
   return (V, TreeEdges)
```

Algorithm 6: The communication graph pruning algorithm

4.2.2 Growing a spanning tree over the communication graph

In this heuristic, we consider again the communication graph given by the solution of the linear program (2). We grow a spanning tree, starting from the source processor, and selecting edges with maximum weight, i.e. edges carrying the maximum number of messages in the solution of the linear program. This heuristic is described in Algorithm 7.

LP-GROW-TREE($\mathcal{P}, P_{source}$)
  solve linear program (2), and compute $n_{u,v}$, the number of messages sent through edge (u, v) during one time-unit
  $TreeEdges \leftarrow \emptyset$
  $TreeVertices \leftarrow \{P_{source}\}$
  while $TreeVertices \neq V$ **do**
    choose the link (u, v) such that $u \in TreeVertices$, $v \notin TreeVertices$ and (u, v) has maximum value $n_{u,v}$
    $TreeVertices \leftarrow TreeVertices \cup \{v\}$
    $TreeEdges \leftarrow TreeEdges \cup \{(u, v)\}$
  return $(TreeVertices, TreeEdges)$

Algorithm 7: The growing tree algorithm based on the communication graph.

5 Experiments

In this section, we describe the experiments conducted to assess the performance of all the previous heuristics. We perform experiments through simulation, so as to test our heuristics on a wide range of heterogeneous platforms.

5.1 Platforms

We use two types of platforms: first we randomly generate platform graphs, using the parameters described in Table 2. In Table 2, the *density* is the probability of the existence of an edge between two nodes.

number of nodes :	10, 20,. . . , 50
density :	0.04, 0.08,. . . , 0.20
$T_{u,v}$:	Gaussian distribution (mean=100MB/s, deviation=20MB/s)
$send_{u,v}$:	$0.80 \cdot \min_{w,(u,w)\in E} \{T_{u,w}\}$ (depends only on sending node u)

Table 2: Parameters for random generation of platform. For each set of parameters, we generate 10 different configurations.

Next, to perform simulations on more realistic platforms, we use platforms generated by TIERS, a popular generator of network topologies [19]. Using TIERS, we generate 100 platforms with 30 nodes, and 100 platforms with 65 nodes. These platforms have a density between 0.05 and 0.15, depending on the number of nodes. We use the same distribution for the values of $T_{u,v}$ as for random platforms.

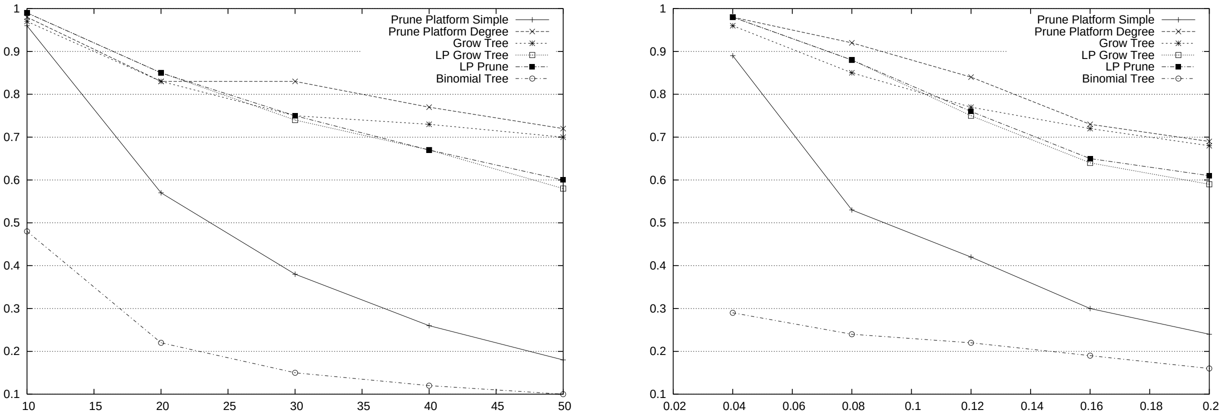
For both random and Tiers platforms, we conduct some experiments under the multi-port model. In that case, the value of $send_{u,v}$ is set to 80% of the shortest link occupation when sending a message to one neighbor. This percentage is somewhat arbitrary, but our simulations show that the results do not strongly depend upon this parameter.

5.2 Results

On each platform configuration, we compute the throughput of each heuristic, and compare it to the optimal throughput of the MTP problem under the one-port model, obtained as the solution of the linear program described in Section 4. So what is called "relative performance" in the following results is the throughput of a given heuristic compared to the best throughput that can be achieved using several broadcast trees.

5.2.1 Random platforms, one-port

In Figure 4(a), we plot the performance of the different heuristics for several platform sizes. We point out that for a small number of nodes, our heuristics are able to reach a throughput very close to the optimal. For larger platforms, the "advanced heuristics" (i.e. Topo-Prune-Degree, Topo-Grow-Tree, LP-Prune and LP-Grow-Tree) are able to reach 60% of the optimal throughput with several trees. The heuristics Topo-Prune-Degree and Topo-Grow-Tree are even within 70% of the optimal. The simple pruning heuristic (Topo-Prune-Simple) behaves well for a small number of nodes, but is not scalable to larger platforms: its throughput falls down to 20% of the optimal. Last, the Binomial-Tree heuristic gives very poor results, which was expected because it does not take topological information into account.



(a) Performance as a function of number of nodes

(b) Performance as a function of density

Figure 4: Performance of the different heuristics in the one-port model. The Y axis is the relative average performance compared to the optimal solution for the MTP problem.

Figure 4(b) shows the relative performances of our heuristics as a function of the density of the underlying platform. Intuitively, a higher density allows for more freedom in the routing, hence more gain in using several trees in parallel. However, our refined heuristics are still within 70% of the optimal throughput.

5.2.2 Random platforms, multi-port

Figure 5 shows the performance of the Multi-Port Growing Tree heuristic described in Section 3.2, compared to the Binomial-Tree heuristic. Again, we compute the ratio of the throughput of these heuristics over the optimal throughput given by the linear program of Section 4. At first sight it may seem surprising to achieve ratios larger than 1 (hence better than the "optimal"). However, recall that the linear program gives an optimal throughput under the one-port assumption, while the throughput of the heuristics are computed using the multi-port model, which allows some overlapping of consecutive sends by a given processor We still choose to plot this ratio because: (i) we do not know how to compute the optimal throughput under the multi-port model; and (ii) we believe that it is interesting to compare all heuristics over the same basis, here the solution of the linear program, giving a good idea of what can be achieved on the platform.

We notice that the performance of the Binomial-Tree heuristic is better that previously, and this is because the multi-port model is less constrained, allowing for multiple communications to go through one node without severely decreasing the throughput. However, the adapted Multi-Port Growing Tree heuristic gives much better results. We also present the performance of the heuristics based on linear programming under this model, which are close to the performance of the adapted Growing Tree heuristic. Finally, note that other heuristics, such as Topo-Prune-Degree, can be adapted to the multi-port model, and give good results too: the latter heuristic is labeled Multiport-Prune-Degree in Figure 5).

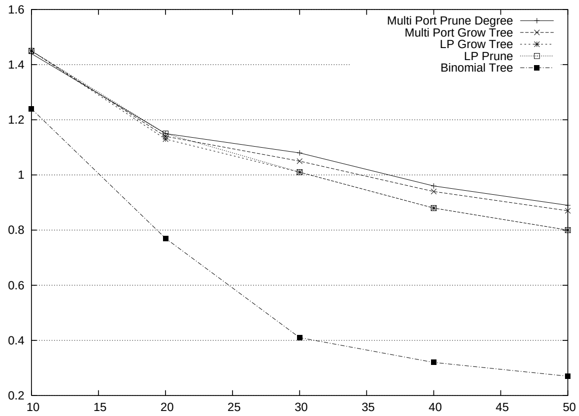


Figure 5: Performance of the different heuristics in the multi-port model. The X axis is the number of nodes in randomly generated platform. The Y axis is the relative average performance compared to the optimal solution for the MTP problem on the same platform (but under the one-port model).

5.2.3 Tiers platforms, one-port

Table 3 gives the results of all the heuristics for Tiers generated platforms, under the one-port model. The results are congruent with those on randomly generated platforms, although the LP-based heuristics (LP-Prune and LP-Grow-Tree) give a slightly better results on large Tiers platforms.

6 Related work

As pointed out in the introduction, most papers dealing with broadcasting on heterogeneous platforms restrict to the STA problem, i.e. they build a single spanning tree, without pipelining.

As mentioned in Section 2.3, Banikazemi et al. [1] have considered a simple model in which the heterogeneity among processors is characterized by the speed of the sending processors.

number	Platform based heuristics			LP based heuristics		Binomial
of nodes	Prune Simple	Refined Prune	Grow Tree	Grow Tree	Prune Max	Tree
30	46% (±0.12%)	82% (±10%)	75% (±10%)	82% (±11%)	82% (±11%)	11% (±2%)
65	30% (±10%)	73% (±13%)	71% (±12%)	73% (±11%)	74% (±11%)	5% (±1%)

Table 3: Performance of the one-port heuristics on two types of platforms generated by Tiers (average value, (±deviation%))

The authors argue that this simple model of heterogeneity can well describe the different communication delays in a heterogeneous cluster. They introduce the Fastest Node First (FNF) heuristic: to construct a good broadcast tree, it is better to put fastest processors (processors that have the smallest sending time) at the top of tree. Some theoretical results (NP-completeness and approximation algorithms) have been developed for the problem of broadcasting a message in this model: see [15, 21, 25, 26, 31].

The (bidirectional) one-port model; which we extensively used in this paper, has been introduced by Bhat et al. [8, 7]. In [8], some heuristics are proposed for the STA problem (broadcast and multicast) using this model.

Sun et al [34] investigate clusters of SMPs connected by one-port switches, and they introduce several heuristic for the STA problem on such hierarchical platforms. Other collective communications, such as multicast, scatter, all-to-all, gossiping, and gather (or reduce) have been studied in the context of heterogeneous platforms: see [16, 27, 24, 28, 29, 30] among others.

7 Conclusion

In this paper, we have considered the problem of broadcasting large messages on heterogeneous platforms. The broadcast may be performed either using a single tree and sending the whole message at once (the STA approach), or using a single tree and sending the message in a pipeline fashion (the STP approach), or using several broadcast trees and sending the message in a pipeline fashion (the MTP approach). Surprisingly, the former two problems are NP-Complete, whereas the latter can be solved in polynomial time.
Nevertheless, the use of a single tree has many advantages. In particular, there is no need of complex synchronization to handle conflicts that may arise on communication resources; also, a communication scheme using a single broadcast tree may well be more robust to small changes in link performances.

We have derived several heuristics for the STP approach, and compared them, through extensive simulations, against the optimal solution of the MTP problem. The results presented in this paper show that for realistic platforms such as those generated by Tiers [19], there is little difference in the throughput achieved when using a single or several broadcast trees. Our results also prove that it is mandatory to take into account the actual topology and performances of the network to derive efficient implementations. Estimates of the transfer speeds can be acquired by querying grid information services [12, 40, 13], or by directly observing the performance being delivered by the communication links. Whenever available on the target heterogeneous platform, plugging this information into our heuristics is very likely to provide significant improvement over current MPI implementations.

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. 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] A. Bar-Noy, S. Guha, J. S. Naor, and B. Schieber. Message multicasting in heterogeneous networks. *SIAM Journal on Computing*, 30(2):347–358, 2000.

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

[5] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining broadcasts on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer Society Press, 2004.

[6] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining broadcasts on heterogeneous platforms. *IEEE Trans. Parallel Distributed Systems*, 2004, to appear. Available as LIP Research Report 2003-34.

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

[8] P. Bhat, C. Raghavendra, and V. 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] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

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

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

[12] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[13] The Ganglia Project. http://ganglia.sourceforge.net.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

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

[16] 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.

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

[18] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, 1989.

[19] K. Calvert and M. Doar and E.W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[20] N. Karonis, 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.

[21] S. Khuller and Y. Kim. On broadcasting in heterogenous networks. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1011–1020. Society for Industrial and Applied Mathematics, 2004.

[22] H. Ko, S. Latifi, and P. Srimani. Near-optimal broadcast in all-port wormhole-routed hypercubes using error-correcting codes. *IEEE Trans. Parallel and Distributed Systems*, 11(3):247–260, 2000.

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

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] Z. B. Miled, J. A. Fortes, R. Eigenmann, and V. Taylor. On the implementation of broadcast, scatter and gather in a heterogeneous architecture. In *Thirty-First Annual Hawaii International Conference on System Sciences (volume 3)*. IEEE Computer Society Press, 1998.

[30] 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.

[31] F. Ooshita, S. Matsumae, T. Masuzawa, and N. Tokura. Scheduling for broadcast operation in heterogeneous parallel computing environments. *Systems and Computers in Japan*, 35(5):44–54, 2004.

[32] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.

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

[34] Y. Sun, D. Bader, X. Lin, and Y. Ling. Broadcast on clusters of SMPs with optimal concurrency. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*. CSREA Press, 2002.

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

[36] Y.-C. Tseng, S.-Y. Wang, and C.-W. Ho. Efficient broadcasting in wormhole-routed multicomputers: a network-partitioning approach. *IEEE Trans. Parallel and Distributed Systems*, 10(1):44–61, 1999.

[37] R. van de Geijn. On global combine operations. *J. Parallel and Distributed Computing*, 22(2):324–328, 1995.

[38] S.-Y. Wang and Y.-C. Tseng. Algebraic foundations and broadcasting algorithms for wormhole-routed all-port tori. *IEEE Trans. Computers*, 49(3):246–258, 2000.

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

[40] R. Wolski, N. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(10):757–768, 1999.