Two-Tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan [1]

Peter Sanders and Jochen Speck

Universität Karlsruhe, D-76128 Karlsruhe, Germany

and

Jesper Larsson Träff *

NEC Laboratories Europe, NEC Europe Ltd., Rathausallee 10, D-53757 Sankt Augustin, Germany

Abstract

We present a new, simple algorithmic idea for the collective communication operations broadcast, reduction, and scan (prefix sums). The algorithms concurrently communicate over two binary trees which both span the entire network. By careful layout and communication scheduling, each tree communicates as efficiently as a single tree with exclusive use of the network. Our algorithms thus achieve up to *twice* the bandwidth of most previous algorithms. In particular, our approach beats all previous algorithms for reduction and scan. Experiments on clusters with Myrinet and InfiniBand interconnect show significant reductions in running time for all three operations sometimes even close to the best possible factor of two.

1 Introduction

Parallel programs for distributed memory machines can be structured as sequential computations plus calls to a small number of (collective) communication primitives. Part of the success of communication libraries such as MPI

* Corresponding Author: Phone +49 2241 92 52 60, Fax: +49 2241 92 52 99
Email addresses: `sanders@ira.uka.de` (Peter Sanders), `jspeck@ira.uka.de` (Jochen Speck), `traff@it.neclab.eu` (Jesper Larsson Träff).

[1] This paper is a substantially expanded version of the conference presentation: P. Sanders, J. Speck, J. L. Träff, "Full Bandwidth Broadcast, Reduction and Scan with only two Trees", EuroPVM/MPI 2007, Lecture Notes in Computer Science, Volume 4757, pages 17–26, 2007.

(the *Message Passing Interface*, see [21]) is that this approach gives a clear division of labor: the programmer thinks in terms of these primitives and the library is responsible for implementing them efficiently. Hence, there has been intensive research on optimal algorithms especially for basic, collective communication primitives that involve sets of processors [1,3,4,11,17,20,22,24,25]. Even constant factors matter. This paper is concerned with the somewhat surprising observation which for a standard cost model and three of the most important primitives—broadcast [2], reduction [3] (accumulate), and scan [4] (prefix sum)— a simple algorithmic approach closes the gap between previously known approaches and the lower bound of the execution time given by the (bidirectional) communication bandwidth of the machine [18]. For broadcast this has previously been achieved also by other algorithms [2,13,24], but these are typically more complicated and do not extend to the reduction and parallel prefix operations. We believe the results achieved for reduction and parallel prefix to be the theoretically currently best known. The algorithms are simple to implement, and have been implemented within the framework of NEC proprietary MPI libraries[16].

Overview

In Section 2 we review related work and basic ideas relevant for the subsequent discussion. In particular, our algorithms adopt the widely used approach to split the message into k packets and sending them along a binary tree in a pipelined fashion. The central new idea behind our *2Tree algorithms* is to use *two* such binary tree at once, each handling half of the message. Section 3 explains how these trees are constructed and how communication can be scheduled so that the both trees can work at the same bandwidth one would get with a single tree with complete use of the network. This is possible because we can pair leaves of one tree with interior nodes of the other tree. When a processing element (PE) sends to its 'second' child in one tree, it can simultaneously receive from its parent in the other tree. Note that this covers the entire communication requirement of a leaf. Scheduling the communication optimally is possible by modeling the communication requirements as a bipartite graph.

Section 4 adapts the 2Tree idea to three different collective communication operations. For a broadcast, the PE sending the message is not directly integrated into the two trees but it alternates between sending packets to each of the roots of the two trees. Reduction is basically a broadcast with inverted

[2] One PE sends a message to all other PEs.
[3] The sum $\bigoplus_{i<p} M_i$ (or any other associative operation) of values on each PE is computed.
[4] On PE j, $\bigoplus_{i\leq j} M_i$ is computed.

direction of communication (plus the appropriate arithmetical operations on the data). For scanning, both trees (which have different root PEs) work independently. Otherwise, the necessary communications resemble a reduction followed by a broadcast.

Important refinements of the 2Tree idea are discussed in Section 5. Perhaps the most surprising result is a simple yet nontrivial algorithm for computing the communication schedule of each PE in time $\mathcal{O}(\log p)$ without any communication between the PEs. We also outline how to adapt the 2Tree algorithms to the simplex model and to networks with small bisection bandwidth. In Section 6 we report implementation results which indicate that our new algorithms outperform all previous algorithms in some situations. Section 7 summarizes the results and outlines possible future research.

2 Preliminaries and Previous Work

Let p denote the number of processing elements (PEs) and n the size of the messages being processed. The PEs are numbered successively from 0 to $p-1$. Each PE knows p and its own number. We assume a simple, linear communication cost model, and let $\alpha + \beta m$ be the cost for a communication involving m data elements. We use the single-ported, *full-duplex* variant of this model where a PE can simultaneously send data to one PE and receive data from a possibly different PE (sometimes called the *simultaneous send-receive model* [2]). Section 5.2 explains how to adapt this to the *simplex* (=half-duplex) model where each PE can either send or receive.

A *broadcast* sends a message from a specified root PE to all other PEs. It is instructive to compare the complexity of broadcasting algorithms with two simple lower bounds: $\alpha \log p$ is a lower bound (in this paper $\log p$ stands for $\log_2 p$) because with each communication, the number of PEs knowing anything about the input can at most double. Another lower bound is βn because all the data has to be transmitted at some point. *Binomial tree broadcasting* broadcasts the message as a whole in $\log p$ steps. This achieves nearly full bandwidth for $\beta n \ll \alpha$ but is suboptimal by a factor approaching $\log p$ for $\beta n \gg \alpha$.

For very large messages, most algorithms split the message into k pieces, transmitting a piece of size n/k in time $\alpha + \beta \frac{n}{k}$ in each step. To avoid tedious rounding issues, from now on we assume that k divides n. Sometimes, we also write $\log p$ where $\lceil \log p \rceil$ would be correct, etc. The tuning parameter k can be optimized using calculus. Arranging the PEs into a linear pipeline yields execution time $\approx (p+k)(\alpha + \beta \frac{n}{k})$. This is near optimal for $\beta n \gg \alpha p$ but suboptimal up to a factor $\frac{p}{\log p}$ for short messages. A reasonable compromise is to use pipelining with a binary tree. Using a complete binary tree we get execution time

$$\mathbf{2}(\log p + k)(\alpha + \beta\frac{n}{k}) \ . \tag{1}$$

For both small and large message lengths, this is about a factor of two away from the lower bounds.

Fractional tree broadcasting (and commutative reduction) [17] interpolates between a binary tree and a pipeline and is faster than either of these algorithms but slower than [1,11,20,24] (see below). The resulting communication graphs are "almost" trees and thus can be embedded into networks with low bisection bandwidth.

If p is a power of two, there is an elegant optimal broadcasting algorithm [11] based on $\log p$ edge disjoint spanning binomial trees (ESBT) in a hypercube. However, this algorithm does not work for other values of p and cannot be generalized for scanning or noncommutative reduction. Since it fully exploits the hypercube network, its embedding into lower bandwidth networks like meshes leads to high edge contention. Recently, a surprisingly simple extension of this algorithm to arbitrary values of p was given [10]. Highly complicated broadcast algorithms for general p were developed earlier [1,20,24]. Compared to ESBT these algorithms have the additional disadvantage that they need time polynomial in p (rather than logarithmic) to initialize the communication topology. [5] Hence, for large machines these algorithms are only useful if the same communication structure is used many times.

For broadcast the idea of using two trees to improve bandwidth was previously introduced in [7], but the need for coloring was not realized (due to the TCP/IP setting of this work).

At the cost of a factor two in the amount of communicated data, the overheads for message startups and PE synchronization implicit in pipelined algorithms can be avoided by reducing broadcasting to a scatter operation followed by an allgather operation [3]. Hence, this algorithm is good for medium sized messages and situations where the processors get significantly desynchronized by events outside the control of the communication algorithms.

In [5] a single tree is used, each half of which broadcasts only half the message. An exchange step at the end completes the broadcast operation. The performance of this algorithm lies about midway between the plain binary tree algorithm and our algorithm.

A tree based broadcasting algorithm can be transformed into a reduction algorithm by reversing the direction of communication and adding received

[5] [1,20] do not even provide an explicit initialization algorithm.

data wherever the broadcasting algorithm forwards data to several recipients. However, for noncommutative operations this transformation only works if the PEs form an in-order numbering of the tree. This is a problem for algorithms that use multiple trees with inconsistent numbering.

A scan operation can be subdivided into an up-phase that is analogous to a reduction and a down-phase analogous to a broadcast (except for additional arithmetical operations performed. Each of these phases can be pipelined [14,19]. In [19] it is additionally observed that by overlapping these two phases, communication requirement can be further reduced. Disregarding internal computations, the time complexity is $\approx (3k + 4\log p)(\alpha + \beta\frac{n}{k})$.

3 Two pipelined binary trees instead of one

To explain the new algorithm we consider first the broadcast operation. The problem with pipelined binary tree broadcasting is that the bidirectional communication is only exploited in every second communication step for the interior node processors and not at all for the root and leaf processors. In particular, the leaves are only receiving blocks of data. We propose to use two [6] binary trees simultaneously in order to achieve a more balanced use of the communication links. The two trees are constructed in such a way that the interior nodes of one tree correspond to leaf nodes of the other. This allows us to take full advantage of the bidirectional communication capabilities. In each communication step, a processor receives a block from its parent in one of the two trees and sends the previous block to one of its children in the tree in which it is an interior node. To make this work efficiently, the task is to devise a construction of the two trees together with a schedule which determines for each time step from which parent a block is received and to which child the previous block is sent so that each communication step consists of at most one send and at most one receive operation for each processor. We note that no explicit global synchronization is necessary; point-to-point communication suffices to implicitly synchronize the processors to the extent necessary.

Let $h = \lceil \log(p + 2) \rceil$. We construct two binary trees T_1 and T_2 of height [7] $h - 1$ with the following properties: PEs are assigned to the nodes of these trees such that they form an in-order numbering of both T_1 and T_2. T_2 is dual to T_1 in the sense that its nonleaf nodes are the leaves of T_1 and vice versa.

[6] Note that the 'two' is not just some first step in a family of more and more sophisticated algorithms with more and more trees. We show that two trees are enough to eliminated any imbalance of the load on the PEs thus resulting in an algorithm that is optimal up to terms sublinear in the message length.

[7] The *height* of a tree is the number of edges on the longest root–leaf path.

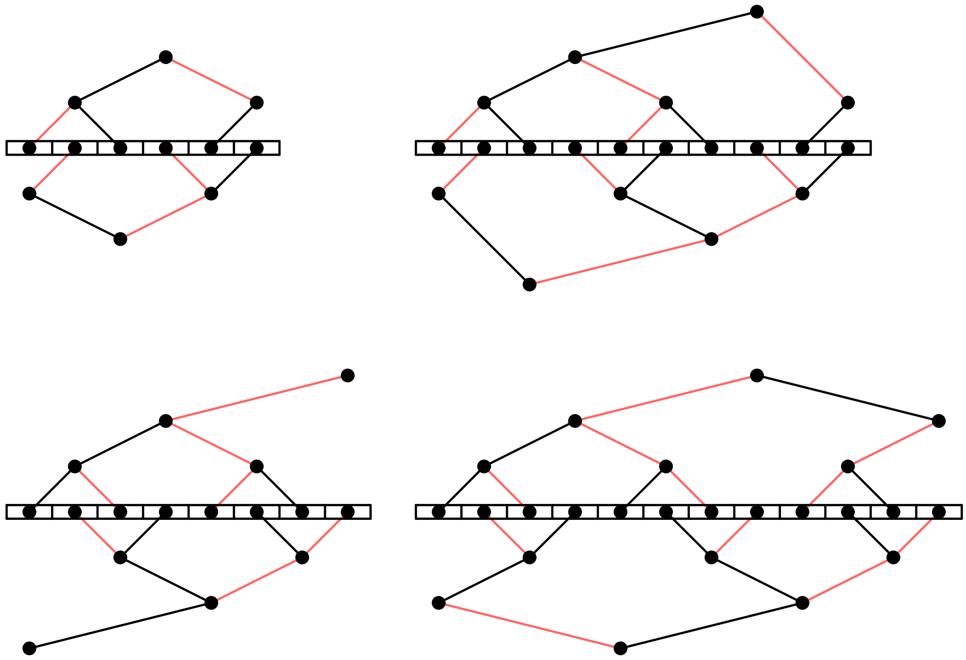

Fig. 1. Colorings for 2Tree collectives and $p \in \{6, 8, 10, 12\}$ using *mirroring* for the tree construction.

Tree T_1 is constructed as follows: If $p = 2^h - 2$, T_1 is a complete binary tree of height $h - 1$ except that the rightmost leaf is missing. Otherwise, T_1 consists of a complete binary tree of height $h-2$ covering PEs $0..2^{h-1}-2$, a recursively constructed tree covering PEs $2^{h-1}..p$, and a root at PE $2^{h-1}-1$ whose children are the roots of the left and the right subtree. Note that in T_1, PE 0 is always a leaf and PE $p - 1$ never has two children.

There are two ways to construct T_2 which both have advantages and disadvantages. With *shifting*, T_2 is basically a copy of T_1 shifted by one position to the left except that the leaf now at position -1 is dropped and PE $p - 1$ becomes a child of PE $p-2$. With *mirroring*, T_2 is the mirror image of T_1. This only works for even p, but the additional symmetry simplifies certain tasks. Figure 1 gives examples. For odd p, we can add PE $p - 1$ as a new root for both T_1 and T_2. For broadcasting and commutative reduction we can add PE $p-1$ as the right child of the rightmost PE $(p-1)$ in T_1 and as the left child of the leftmost PE (1) in T_2. The latter trick can also be used to increase by one the maximum number of PEs in a tree of given height.

In order to simultaneously use both trees for collective communication, we need the following lemma:

Lemma 1 *The edges of T_1 and T_2 can be colored with colors 0 and 1 such that*

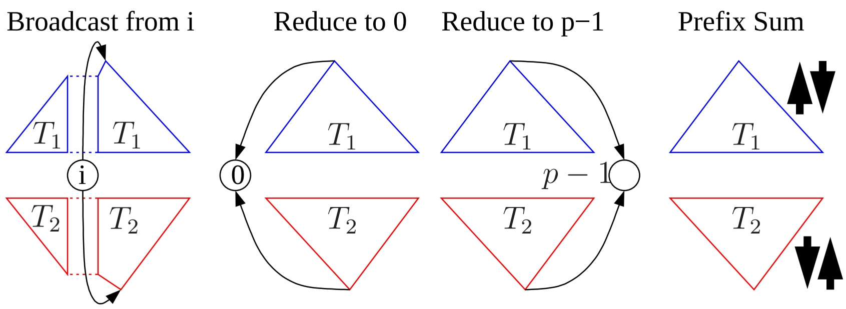

Fig. 2. Data flow in 2Tree collective communication routines.

(1) no PE is connected to its parent nodes in T_1 and T_2 using edges of the same color and
(2) no PE is connected to its children nodes in T_1 or T_2 using edges of the same color.

PROOF. Consider the bipartite graph $B = (\{s_0, \ldots, s_{p-1}\} \cup \{r_1, \ldots, r_{p-1}\}, E)$ where $\{s_i, r_j\} \in E$ if and only if j is a successor of i in T_1 or T_2. This graph models the sender role of PE i with node s_i and the receiver role of PE i with node r_i. By definition of T_1 and T_2, B has maximum degree two, i.e., B is a collection of paths and *even* cycles. Hence, the edges of B can be two-colored by just traversing these paths and cycles.

This lemma already implies a linear work algorithm for finding a coloring. Although this algorithm reduces to list ranking and hence is parallelizable [9], the coloring procedure might be a bottleneck for a practical parallel implementation. Therefore, in Section 5.1 we explain how each PE can independently compute and color its incident edges in time $\mathcal{O}(\log p)$ for the case of mirroring.

4 Collectives

The following theorems state bounds for 2Tree algorithms that can be used to implement the MPI operations `MPI_Bcast`, `MPI_Reduce`, `MPI_Scan` and `MPI_Exscan`.

4.1 Broadcast

Assume PE i wants to send a message to everybody. See also Figure 2. We build the trees T_1 and T_2 for the remaining PEs (renumbering them $0..p-2$). The tree edges are directed from the root towards the leaves. T_1 broadcasts the first half of the message using pipelined binary tree broadcasting. Concurrently, T_2

broadcasts the second half using the same algorithm. PE i takes turns dealing out one of k pieces of the left and right halves to the roots of T_1 and T_2 respectively. The coloring of the edges determines when an edge is used. In time step j, all PEs communicate along edges with color $j \bmod 2$, i.e., they will simultaneously receive from the parent in one tree and send to one child in the other tree.

We will now give an analysis showing that the total communication time is about

$$\beta m + 2\alpha \log p + \sqrt{8\alpha\beta m \log p} \tag{2}$$

where we ignore a few rounding issues to keep the formulas simple. We add footnotes that explain what has to be done to get a theorem with a similar albeit much more complicated formula. Suppose the message is split into $2k$ pieces so that a communication step takes time $\alpha + \beta\frac{n}{2k}$. Let $h \approx \log p$ denote the height of the resulting communication structure with root at PE i and two trees below it. [8] After $2h$ steps, the first data packet has reached every node in both trees. Afterwards, every node receives another packet in every step until it received all the data. Hence, the total number of steps [9] is $\le 2h + 2k$ resulting in a total execution time of

$$\le (2h + 2k)(\alpha + \beta\frac{n}{2k}) .$$

We can now choose an optimal $k = k^* = \sqrt{\frac{\beta m h}{2\alpha}}$ using calculus resulting in the above bound. [10]

Note that when $\beta n \gg \alpha \log p$, the execution time bound (2) approaches the lower bound βn. This is a factor of two better than pipelined binary tree broadcasting and up to a factor $\Theta(p/\log p)$ better than a linear pipeline. The α-dependent term is a factor two larger than [1,20,24]. However, for long messages this is only a small disadvantage often outweighed by a much faster initialization algorithm and possibly easier embeddabilty (see Section 7).

4.2 Reduction

A reduction computes $\bigoplus_{i<p} M_i$ where M_i is a vector of length n originally available at PE i. At the end, the result has to be available at a specified root PE. The operation $\bigoplus$ is associative but not necessarily commutative (e.g. the

[8] More precisely, we have $h = \lceil \log(p+1) \rceil$ or $h = 1 + \lceil \log p \rceil$ depending on the exact algorithm for constructing the trees and whether p is odd (see Section 3).

[9] The actual number is one less since we have counted one packet twice.

[10] For a tight bound we have to add a small constant since we have to round to the next integer from k^*.

MPI specification [21] requires support for noncommutative operations). The 2Tree-reduction algorithm assumes that the root PE has index 0 or $p-1$ (see also Figure 2). [11] In this case, reduction is the mirror-image of broadcasting. Assume PE i wants to compute the sum over all messages. We build the trees T_1 and T_2 for the remaining PEs. The tree edges are directed from the leaves towards the roots. T_1 sums the first half of the message for PEs with index from $\{0, \ldots, p-1\} \setminus \{i\}$ and T_2 sums the second half. PE i takes turns receiving pieces of summed data from the roots of T_1 and T_2 and adds its own data. Disregarding the cost of arithmetic operations, we have the same execution time bound 2 as for broadcasting.

If the operator $\oplus$ is commutative, the result can be achieved for any root r by appropriately renumbering the PEs.

4.3 Scan

A *scan* computes the prefix sums $\bigoplus_{i \le j} M_i$ for PE j using an associative operation $\oplus$ as already described in Section 4.2. In the 2Tree-scan algorithm, the two trees operate completely independently—each on one half of the input message. For one tree, the algorithm is the scan algorithm described in [19] adapted to the full-duplex model. To make the paper self contained, we summarize the algorithm:

The in-order numbering has the property that PEs in a subtree $T(j)$ rooted at j have consecutive numbers in the interval $[\ell, \ldots, j, \ldots, r]$ where ℓ and r denote the first and last PE in $T(j)$, respectively. The algorithm has two phases. In the *up-phase*, PE j first receives the partial result $\oplus_{i=\ell}^{j-1} M_i$ from its left child and adds x_j to get $\oplus_{i=\ell}^{j} M_i$. This value is stored for the down-phase. PE j then receives the partial result $\oplus_{i=j+1}^{r} M_i$ from its right child and computes the partial result $\oplus_{i=\ell}^{r} M_i$. PE j sends this value upward without keeping it. In the *down-phase*, PE j receives the partial result $\oplus_{i=0}^{\ell-1} M_i$ from its parent. This is first sent down to the left child and then added to the stored partial result $\oplus_{i=\ell}^{j} M_i$ to form the final result $\oplus_{i=0}^{j} M_i$ for j. This final result is sent down to the right child.

With obvious modifications, the general description covers also nodes that need not participate in all of these communications: Leaves have no children. Some nodes may only have a leftmost child. Nodes on the path between root and leftmost leaf do not receive data from their parent in the down-phase. Nodes on the path between rightmost child and root do not send data to their parent in the up-phase. The data flow is summarized in Figure 4.

[11] Otherwise, $2\beta m$ is a lower bound for the execution time of reduction anyway since PE i must receive n elements both from the left and from the right.

The total execution time of 2Tree-scan is about twice the time bound 2 needed for a broadcast.

5 Refinements

5.1 Fast Coloring

In this section we use mirroring for tree construction and thus assume that p is even. The incoming edge of the root of T_1 has color 1. It suffices to explain how to color the edges incident to PEs that are inner nodes in T_1—leaves in T_1 are inner nodes in T_2, and this case is symmetric in the mirroring scheme. In this section (and only here) it is convenient to assume that PEs are numbered from 1 to p. In this case, the height of node i in T_1 is the number of trailing zeros in i. With these conventions, we only have to deal with even PE indices i (PEs with odd i are leaves of T_1). Furthermore, it is enough to know how to compute the color c of the incoming edge of PE i in T_1. The color of the other incoming edge (when i is a leaf in T_2) will be $1 - c$ and the color of the edges leaving i in T_1 can be found by computing the colors of the edges entering the children of i.

Figure 3 gives pseudocode solving the remaining coloring problem. Function inEdgeColor is initially called with the trivial lower bound $h = 1$ for the height of an inner node. The function first finds the exact height h of node i by counting additional trailing zeroes. Then the id i' of the parent of i is computed. The desired color only depends on the color of the parent i' (which is of course also an inner node of T_1), the value of $p \bmod 4$ (2 or 0), and on the relative position of i and i'. The appendix proves that the simple formula given in Figure 3 yields the correct result. There are at most $\log p$ levels of recursion and $\log p$ total iterations of the loop incrementing h over *all* iterations taken together. Thus the total execution time is $\mathcal{O}(\log p)$. This establishes the following main theorem:

Theorem 2 *In the mirroring scheme, the edges of each node can be colored in time* $\mathcal{O}(\log p)$*.*

5.2 The Simplex Model

In the less powerful simplex communication model, where the PEs can only either send or receive at any given time, we can use a trick already used in [17]: PEs are teamed up into *couples* consisting of a *receiver* and a *sender*.

Function inEdgeColor(p, i, h)

  if i is the root of T_1 **then return** 1

  while i bitand $2^h = 0$ **do** h++ −− compute height

  $$i' := \begin{cases} i - 2^h & \text{if } 2^{h+1}\,\mathsf{bitand}\,i = 1 \lor i + 2^h > p \\ i + 2^h & \text{else} \end{cases} \quad \text{−− compute parent of } i$$

  return $\mathsf{inEdgeColor}(p, i', h)\ \mathsf{xor}\ (p/2 \bmod 2)\ \mathsf{xor}\ [i' > i]$−− explained in appendix

Fig. 3. Pseudocode for coloring the incoming edge in T_1 of an even PE i (assuming PE numbers $1..p$). Parameter h is a lower bound for the height of node i in tree T_1. $[P]$ converts the predicate P to a value 0 or 1.

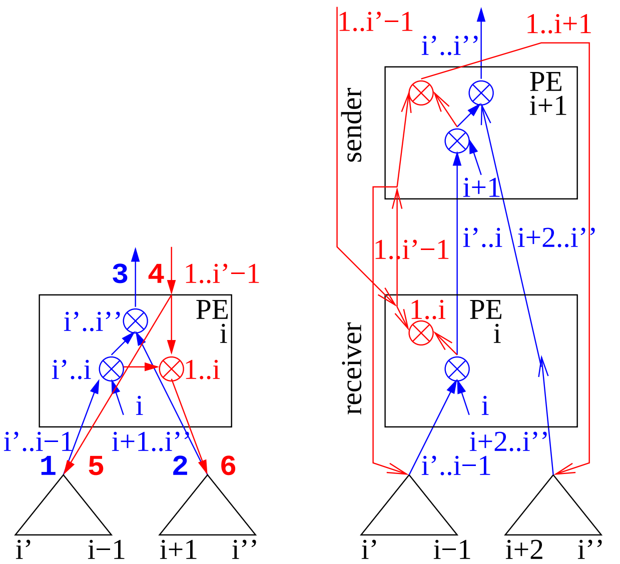

Fig. 4. Data flow for pipelined scanning in duplex (left) and simplex model (right).

In time step $2j$, a couple imitates step j of one PE of the duplex algorithm. In step $2j + 1$, the receiver forwards the data it received to the sender. For reduction and scan, the appropriate arithmetic operations have to be added. For example, Figure 4 illustrates the data flow for scan.

For broadcast and reduction, we can handle odd p by noting that the common root of both trees only needs to send or receive respectively. For scanning, we can add PE $p - 1$ as a new degree one root for both T_1 and T_2. As the rightmost processor in both trees, it never needs to send any data.

Our algorithms are also an improvement in the simplex model. For example, while a pipelined broadcast over a single binary tree needs time $\geq 3\beta n$, with two trees the term depending linearly on n becomes $2\beta n$ which is optimal.

6 Experimental results

The 2Tree algorithms for `MPI_Bcast`, `MPI_Reduce` and `MPI_Scan` have been implemented within the framework of proprietary NEC MPI libraries (see, e.g., [16]). Experiments comparing the bandwidth achieved with the new algorithms to other, commonly used algorithms for these MPI collectives have been conducted on a small AMD Athlon based cluster with Myrinet 2000 interconnect, and a larger Intel Xeon based InfiniBand cluster. Bandwidth is computed as data size m divided by the time to complete for the slowest process. Completion time is the smallest measured time (for the slowest process) over a small number of repetitions. We give only results for the case with one MPI process per node, thus the number of processors p equals the number of nodes of the cluster.

6.1 Broadcast

We compare the implementation based on the 2Tree algorithm to the following algorithms:

- *Circulant graph* first presented in [24]. This algorithm has asymptotically optimal completion time, and only half the latency of the 2Tree algorithm presented here, but is significantly more complex and requires a more involved precomputation than the simple coloring needed for the 2Tree algorithm.
- *Scatter-allgather* for broadcast [3] as developed and implemented in [23]. On the Infiniband cluster we compare to the implementation of the scatter-allgather algorithm in the MVAPICH implementation [22].
- Simple *binomial tree* as in the original MPICH implementation [6].
- Pipelined binary tree.
- Linear pipeline.

Bandwidth results for the two systems are shown in Figure 5 and 6. On both systems the 2Tree algorithm asymptotically achieves the same bandwidth as the optimal circulant graph algorithm, but can of course not compete for small problems where the circulant graph algorithm degenerates into a binomial tree which has only half the latency of the binary tree. Even for large m (up to 16MBytes) both algorithms fare better than the linear pipeline, although none of the algorithms have reached their full bandwidth on the InfiniBand cluster. On the Myrinet cluster the algorithms achieve more than 1.5 times the bandwidth of the scatter-allgather and pipelined binary tree algorithms. For the Myrinet cluster where we also compared to the simple binomial tree, a factor 3 higher bandwidth is achieved for 28 processors.

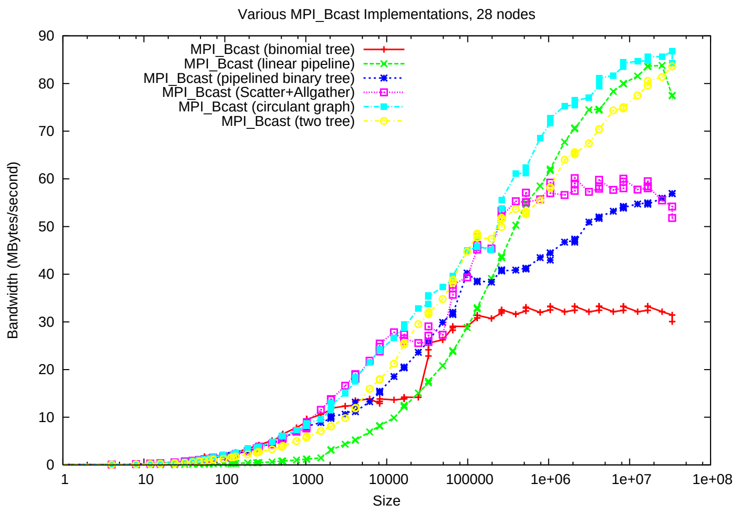

Fig. 5. Broadcast algorithms on the AMD/Myrinet cluster, 28 nodes.

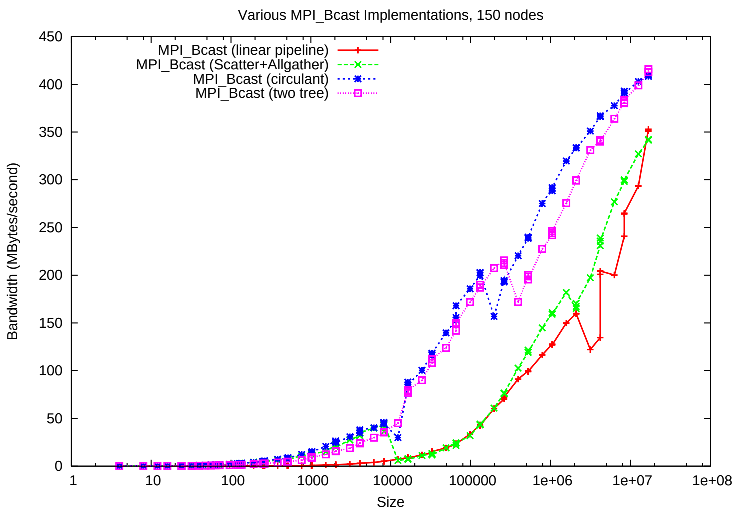

Fig. 6. Broadcast algorithms on the Xeon/InfiniBand cluster, 150 nodes.

The 2Tree broadcast algorithm is a serious candidate for improving the broadcast bandwidth for large problems on bidirectional networks. It is arguably simpler to implement than the optimal circulant graph algorithm [24], but have to be combined with a binomial tree algorithm for small to medium sized problems. Being a pipelined algorithm with small blocks of size $\Theta(\sqrt{m})$ it is also well suited to implementation on SMP clusters [24].

6.2 Reduction

We compare the 2Tree based reduction implementation to the following algorithms:

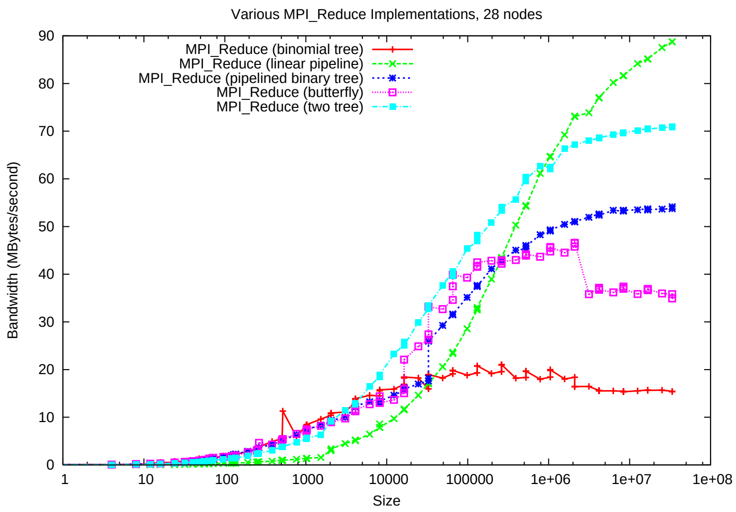

Fig. 7. Reduction algorithms on the AMD/Myrinet cluster, 28 nodes.

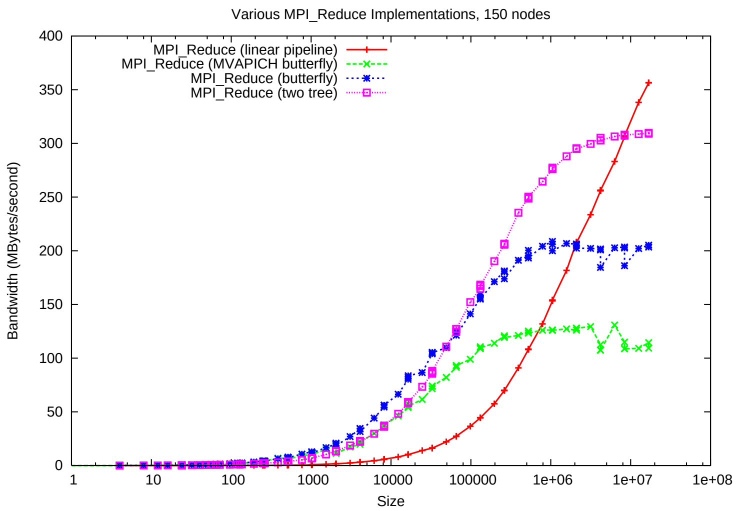

Fig. 8. Reduction algorithms on the Xeon/InfiniBand cluster, 150 nodes.

- Improved *butterfly* [15] and the *butterfly* as implemented in MVAPICH [22].
- Straight-forward *Binomial tree.*
- *Pipelined binary tree.*
- *Linear pipeline.*

Bandwidth results for the two systems are shown in Figure 7 and 8. The 2Tree algorithm achieves about a factor 1.5 higher bandwidth than the second best algorithm which is either the pipelined binary tree (on the Myrinet cluster) or the butterfly (on the InfiniBand cluster). On both systems the linear pipeline achieves an even higher bandwidth, though, but problem sizes have to be larger than 1MByte (for $p = 28$ on the Myrinet cluster), or 16Mbyte (for $p = 150$ on the InfiniBand cluster), respectively. For smaller problems the linear pipeline is inferior and should not be used. On the InfiniBand cluster there is a

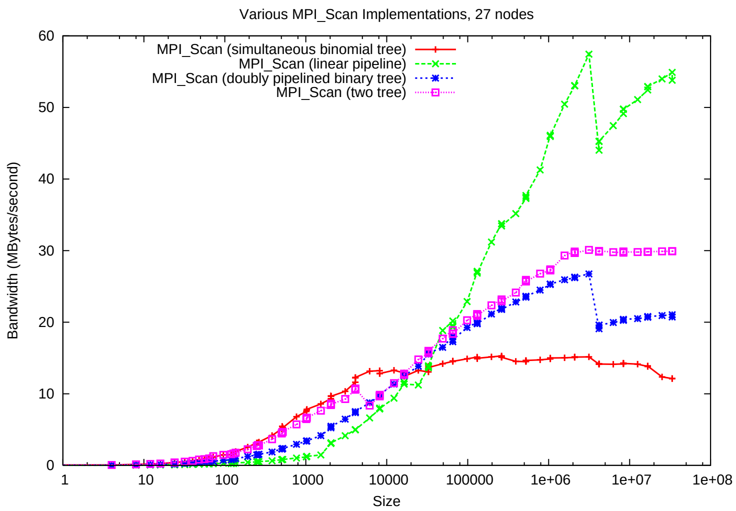

Fig. 9. Scan algorithms on the AMD/Myrinet cluster, 27 nodes.

considerable difference of almost a factor 2 between the two implementations of the butterfly algorithm (with the implementation of [15] being the faster), which is surprising. The sudden drop in bandwidth for the butterfly algorithm on the Myrinet cluster is due to a protocol change in the underlying point-to-point communication, but for this algorithm it is difficult to avoid getting into the less suitable protocol domain. The pipelined algorithms give full flexibility in the choice of block sizes and such effects can thus be better countered.

6.3 Scan

Bandwidth results for parallel prefix are given in Figure 9 for the Myrinet cluster. The 2Tree algorithm is compared to the following algorithms:

- *Simultaneous binomial trees* [8].
- *Doubly pipelined binary tree* [19].
- Linear pipeline.

For large data the 2Tree algorithm achieves almost three times the bandwidth of the simultaneous binomial tree algorithm, and about one third better bandwidth than the doubly pipelined binary tree algorithm which (probably due to a suboptimal pipeline block parameter choice) exhibits a considerable drop in bandwidth around 4MBytes. For very large data, 2Tree based prefix algorithm cannot compete with the linear pipeline. But this is to be expected, since the linear pipeline has asymptotic bandwidth of only βm, compared to $2\beta m$ for the 2Tree algorithm. The linear pipeline thus is the algorithm of choice for $m \gg p$.

PEs	100	1 000	10 000	100 000
Time [μs]	99.15	1399.43	20042.28	48803.58

Table 1
Computation times for the block schedule needed for the circulant graph broadcast algorithm [24] on a 2.1GHz AMD Athlon processor.

6.4 Coloring

We have implemented the logarithmic time coloring algorithm from Section 5.1. Even its constant factors are very good. Even for 100 000 PEs, computing all the information needed for one PE never takes more than 1.5 μs. Note that this is less than the startup overhead for a single message on most machines, i.e., the coloring cost is negligible. In contrast, one of the best currently available implementations of a full bandwidth broadcast algorithm [24] needs considerable time for its $\mathcal{O}(p \log p)$ time block schedule computation. Table 1 shows some execution times. Precomputation is not needed in the recent full bandwidth braadcast algorithm in [10].

7 Summary

We find it astonishing that it has not been exploited before that we can communicate with two trees at once for the cost of only one tree. Since both trees can have the same in-order numbering, this scheme is general enough to support not only broadcasting but also noncommutative reduction and scanning. The resulting algorithms are simple, practical and almost optimal for data sizes m with $m\beta \gg \alpha \log p$.

For operations using a single tree, the term $2\log p$ can be reduced to $\log_\Phi p$ by using a *Fibonacci tree* rather than a complete binary tree ($\Phi = \frac{1+\sqrt{5}}{2}$ denotes the golden ratio) [17]. The idea is to use a "skewed" tree with the property that during a broadcast, all leaves receive their first data packet (almost) simultaneously. Tim Kieritz [12] has adapted this idea to 2Tree-broadcasting. Note that this is nontrivial since our versions of the algorithms are very much tied to the structure of complete binary trees. Indeed, it is yet unclear how to generalize his idea to noncommutative reduction and scanning since he does not use inorder numbering of the nodes.

It would also be interesting to find a fast distributed algorithm for coloring when T_1 and T_2 are constructed using shifting since this scheme seems better for odd p and networks with low bisection bandwidth.

References

[1] A. Bar-Noy and S. Kipnis. Broadcasting multiple messages in simultaneous send/receive systems. In *5th IEEE Symp. Parallel, Distributed Processing*, pages 344–347, 1993.

[2] A. Bar-Noy, S. Kipnis, and B. Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *Discrete Applied Mathematics*, 100(1–2):1–15, 2000.

[3] M. Barnett, S. Gupta, D. G. Payne, L. Schuler, R. van de Geijn, and J. Watts. Building a high-performance collective communication library. In *Supercomputing'94*, pages 107–116, 1994.

[4] E. W. Chan, M. F. Heimlich, A. Purkayastha, and R. A. van de Geijn. On optimizing collective communication. In *Cluster 2004*, 2004.

[5] J. P.-G. et al. Performance analysis of MPI collective operations. *Cluster Computing*, 10(2):127–143, 2007.

[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable imlementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[7] H. H. Happe and B. Vinter. Improving TCP/IP multicasting with message segmentation. In *Communicating Process Architectures (CPA 2005)*, 2005.

[8] W. D. Hillis and J. Guy L. Steele. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986.

[9] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[10] B. Jia. Process cooperation in multiple message broadcast. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 14th European PVM/MPI Users' Group Meeting*, volume 4757 of *Lecture Notes in Computer Science*, pages 27–35. Springer-Verlag, 2007.

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

[12] T. Kieritz. Doppelte fibonacchi-bäume. unpublished manuscript, Universität Karlsruhe, 2008.

[13] O.-H. Kwon and K.-Y. Chwa. Multiple message broadcasting in communication networks. *Networks*, 26:253–261, 1995.

[14] E. W. Mayr and C. G. Plaxton. Pipelined parallel prefix computations, and sorting on a pipelined hypercube. *Journal of Parallel and Distributed Computing*, 17:374–380, 1993.

[15] R. Rabenseifner and J. L. Träff. More efficient reduction algorithms for message-passing parallel systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 36–46. Springer-Verlag, 2004.

[16] H. Ritzdorf and J. L. Träff. Collective operations in NEC's high-performance MPI libraries. In *International Parallel and Distributed Processing Symposium (IPDPS 2006)*, page 100, 2006.

[17] P. Sanders and J. Sibeyn. A bandwidth latency tradeoff for broadcast and reduction. In *6th Euro-Par*, number 1900 in LNCS, pages 918–926, 2000.

[18] P. Sanders, J. Speck, and J. L. Träff. Full bandwidth broadcast, reduction and scan with only two trees. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 14th European PVM/MPI Users' Group Meeting*, volume 4757 of *Lecture Notes in Computer Science*, pages 17–26. Springer-Verlag, 2007.

[19] P. Sanders and J. L. Träff. Parallel prefix (scan) algorithms for MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *LNCS*, pages 49–57. Springer, 2006.

[20] Santos. Optimal and near-optimal algorithms for k-item broadcast. *JPDC: Journal of Parallel and Distributed Computing*, 57:121–139, 1999.

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

[22] R. Thakur, W. D. Gropp, and R. Rabenseifner. Improving the performance of collective operations in MPICH. *International Journal on High Performance Computing Applications*, 19:49–66, 2004.

[23] J. L. Träff. A simple work-optimal broadcast algorithm for message-passing parallel systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 173–180. Springer-Verlag, 2004.

[24] J. L. Träff and A. Ripke. Optimal broadcast for fully connected processor-node networks. *Journal of Parallel and Distributed Computing*, 2008. To appear.

[25] R. van de Geijn. On global combine operations. *Journal of Parallel and Distributed Computing*, 22:324–328, 1994.

A Correctness of Fast Coloring

In this appendix we give the correctness proof for the fast, local coloring algorithm of Section 5.1. Recall that we use the mirroring construction, and that we have to deal only with the case of p even. The task is to color the incoming edge of any inner node of T_1 from its parent. We know from the tree construction that there is always one inner node with exactly one child. This node is the rightmost node in the upper tree and the leftmost node in the lower tree. Also the sibling of an inner node is always an inner node.

In the following, "u corresponds to ℓ" means that u and ℓ belong to the same processor. Let the processors be numbered from left to right starting with 0. Let $h(v)$ be the height of a node v in the respective tree (leaves have height 0). For a node v let $i(v)$ be the PE to which v belongs. In order to prove the correctness of the algorithm given in subsection 5.1, we first introduce some things needed in the proof.

A.1 Partitioning the problem using an auxiliary graph

Let $\tilde{V_1}$ be the nodes of T_1 and $\tilde{V_2}$ be the nodes of T_2. Now we construct a new, auxiliary graph $G = (V, E)$ with $V = \tilde{V_1} \dot{\cup} \tilde{V_2}$. Let E_v be the set of "vertical" edges between the corresponding nodes in T_1 and T_2 or

$$E_v = \{\{u, \ell\} | i(u) = i(\ell),\ u \in \tilde{V_1},\ \ell \in \tilde{V_2}\}$$

and let E_h be the set of "horizontal" edges between siblings in T_1 and T_2 or

$$E_h = \{\{u_1, u_2\} | u_1, u_2 \text{ are siblings in } T_1\} \dot{\cup} \{\{\ell_1, \ell_2\} | \ell_1, \ell_2 \text{ are siblings in } T_2\}$$

then $E = E_v \dot{\cup} E_h$. Set

$$V_1 = \{ \text{ inner nodes in the upper tree } \} \cup \{ \text{ leaves in the lower tree } \}$$

and $V_2 = V - V_1$. In the following pictures edges in G will be blue and edges in T_1 and T_2 will be black.

E_v connects only inner nodes in T_1 with leaves in T_2 and inner nodes in T_2 with leaves in T_1. Hence, E_v does not connect any node in V_1 with a node in V_2. Because of the tree construction, a sibling of an inner node is always an inner node and a sibling of a leaf is always a leaf. Therefore, E_h does not connect any node in V_1 with a node in V_2. Overall, we see that no node in V_1 is adjacent to a node in V_2. It is also clear that there are exactly four nodes with degree one in G (the roots of the trees and the two other nodes that have no siblings), and that all other nodes have degree two (with one incident edge in

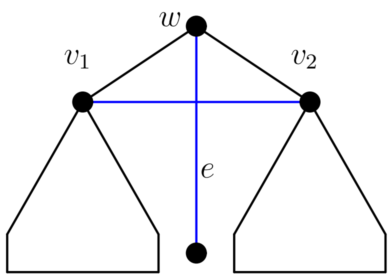

Fig. A.1. The edge e in the proof of lemma 3.

E_h and one in E_v). Therefore it is clear that the set of connected components of G consists of two paths and a certain number of cycles. We first prove:

Lemma 3 *The number of cycles is zero.*

PROOF. Assume the contrary, i.e., G contains a cycle c. W.l.o.g. the nodes of c lie in V_1. Let v_1 be the node with the largest height in T_1 that is in c. Of course the sibling v_2 of v_1 is in c too. Let w be the parent node of v_1 and v_2. The vertical edge e incident to w crosses the edge between v_1 and v_2. But e does not cross any other edge between nodes in V_1, because v_1 and v_2 are the only siblings which are inner nodes of the upper tree with less height than w, and lie on different sides of w, and edges between leaves of the lower tree are never crossed by an edge in E_v. So only one edge of the cycle crosses e, but we need two paths from v_1 to v_2 to close the cycle. Thus the other path must lie above w, but this leads to a contradiction. Hence, there are no cycles in G.

Altogether we have shown that G_1 and G_2 are paths. In this section we will refer to G_1 as a directed path starting in the root of T_1 and ending in the node in V_1 that has no sibling. Obviously the path G_1 visits all nodes in V_1.

A.2 The algorithm

The key to our algorithm is the following *coloring lemma* that explains how the path G_1 controls the coloring:

Lemma 4 *For coloring the incoming edges of two sibling inner nodes v_l and v_r in the upper tree T_1, it suffices to know their order in G_1. In particular, if the first node in G_1 gets color 1, the first node among v_l and v_r gets color 0 and the second one gets color 1. Similarly, consider the incoming edge e_v of a node v in T_1 and its corresponding node ℓ_v in T_2. Edge e_v gets color 1 if v precedes ℓ_v in G_1 and e_v gets color 0 if v succeeds ℓ_v.*

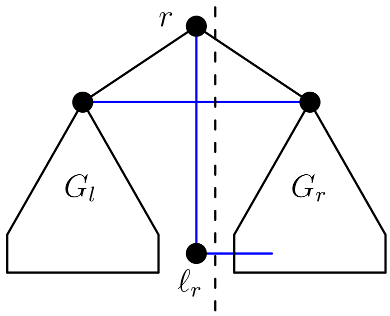

Fig. A.2. Case 1: partitioning of G_1 for $p \equiv 2 \bmod 4$.

PROOF. We first look at the path G_1 starting in the root node r of T_1. Vertical edges of G_1 connect an inner node of T_1 with a leaf in T_2. Because both nodes belong to the same PE, the colors of the incoming tree edges have to be different. For a horizontal edge between two siblings we have a similar constraint—because both nodes are children of the same parent node, the incoming tree edges of the siblings need different colors. Therefore, the incoming tree edges belonging to the nodes in G_1 receive alternating colors. We can write $G_1 = \langle r = u_1, \ell_1, \ell_2, u_2, u_3, \ell_3, \ell_4, \ldots\rangle$ where u stands for nodes in upper tree and ℓ for nodes in the lower tree. The claim of the lemma now follows by alternately assigning 1s and 0s to the nodes in this sequence.

We now do a case analysis that will allow us to find all colors we need. The sibling $\tilde{\ell}$ of a leaf node ℓ always has a PE number of the form $i(\tilde{\ell}) = i(\ell) \pm 2$. A node v of the upper tree has a PE number of the form $i(v) = 2^{h(v)} - 1 + k \cdot 2^{h(v)+1}$ for some $k \in \mathbb{N}_0$. In particular, if $h(v) > 1$ then $i(v) \equiv 3 \bmod 4$.

We have two main cases.

Case 1, $p \equiv 2 \bmod 4$: The lower tree T_2 has a leaf ℓ with $i(\ell) = 1$ which has no sibling. This leaf is also the last node of G_1. All other leaves in T_2 come in pairs such that for the left sibling ℓ_l of each pair we have $i(\ell_l) \equiv 3 \bmod 4$. Thus inner nodes of T_1 with $h(v) > 1$ correspond to left leaves in a pair of leaves of T_2.

Now consider the root r of the upper tree, and let ℓ_r be the corresponding leaf in the lower tree. W.l.o.g. $h(r) > 1$. Let G_l and G_r denote the subgraphs induced by the nodes v of G_1 with $i(v) < i(r)$ and $i(v) > i(r)$ respectively. Figure A.2 depicts this situation.

The two horizontal edges in Figure A.2 are the only edges of G_1 that are not within either G_l or G_r. Furthermore, G_l and G_r are paths—G_r is the first part of path G_1 (except for the nodes r and ℓ_r) and G_l is a postfix of G_1. The nodes from T_1 in G_r and G_l define subtrees T_l and T_r of T_1 respectively. Slightly

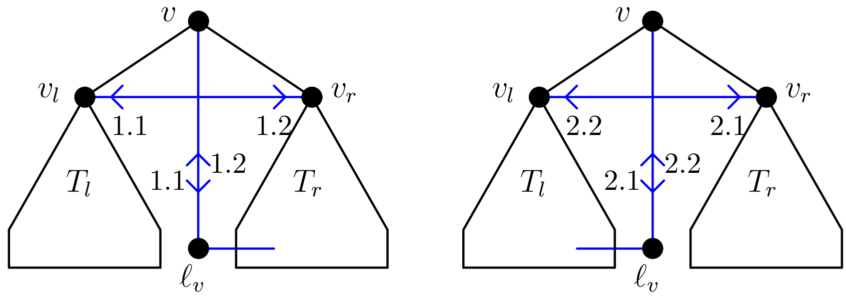

Fig. A.3. The edge directions in the different cases.

generalizing the above approach, we can recursively continue to partition G_1 into smaller and smaller paths corresponding to subtrees in T_1 as long as the root nodes have height larger than one. Consider a node v of T_1 with $h(v) > 1$ and the corresponding leaf ℓ_v of T_2. Let us look at the edge $\{v, \ell_v\}$ in the path. Let T_l be the left subtree of v with root v_l, and let T_r be the right subtree of v with root v_r (see Figure A.3 right).

Case 1.1, v precedes ℓ_v on the path: The first node of G_r on the path is the sibling of ℓ_v and the last node within G_r is v_r; the immediate successor is v_l. Hence, by Lemma 4, the color of the incoming tree edge of v_l is 1, the color of the incoming tree edge of v_r is 0.

Case 1.2, ℓ_v precedes v on the path: Node v_r is an immediate successor of v_l which is the last node on the path within G_l. The last node within G_r is the sibling of ℓ_v. By Lemma 4, the color of the incoming tree edge of v_l is 0, the color of the incoming tree edge of v_r is 1.

Case 2, $p \equiv 0 \bmod 4$: The lower tree has no leaf which has no sibling. Hence, all leaves in T_2 come in pairs such that for the right sibling ℓ_r of each pair we have $i(\ell_r) \equiv 3 \bmod 4$. Thus inner nodes v with $h(v) > 1$ of T_1 correspond always to the right leaf of a pair of leaves of T_2. Analogous to Case 1, a subtree of T_1 always corresponds to a path in G_1. (Although the right subtree can contain zero nodes in this case.) As before, for a node v with $h(v) > 1$ as a root of a subtree, with corresponding leaf ℓ_v, let us look at the edge $\{v, \ell_v\}$ in the path. Let T_l be the left subtree with root v_l, and let T_r be the right subtree of v with root v_r (see Figure A.3 left). Cases 2.1 and 2.2 now are exactly analogous to Cases 1.1 and 1.2 respectively—just exchange the roles of the left and right subtrees of v.

Table A.1 summarizes the cases. We observe, that the columns v_l and v_r are a function of the columns v and $p/2 \bmod 2$. Moreover, we can decide whether a node is a left or right child of its parent by comparing the corresponding PE numbers. Now we can verify the coloring function in the last line of Algorithm 3 by inspecting Table A.1.

Table A.1
Case analysis for coloring incoming edges of inner nodes of T_1. In each case, the color for column v is an immediate consequence of the second part of Lemma 4.

Case	color v_l	color v_r	color v	$p/2 \bmod 2$
1.1	1	0	1	1
1.2	0	1	0	1
2.1	0	1	1	0
2.2	1	0	0	0