Full Bandwidth Broadcast, Reduction and Scan with only two Trees

Peter Sanders[1], Jochen Speck[1], and Jesper Larsson Träff[2]

[1] Universität Karlsruhe
Am Fasanengarten 5, D-76131 Karlsruhe, Germany
`sanders@ira.uka.de`
[2] C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10, D-53757 Sankt Augustin, Germany
`traff@ccrl-nece.de`

Abstract. We present a new, simple algorithmic idea for exploiting the potential for bidirectional communication present in many modern interconnects for the collective MPI operations broadcast, reduction and scan. Our algorithms achieve up to *twice* the bandwidth of most previous and commonly used algorithms. In particular, our algorithms for reduction and scan are the currently best known. Experiments on clusters with Myrinet and InfiniBand interconnects show significant reductions in running time for broadcast and reduction, for reduction even close to the best possible factor of two.

1 Introduction

The *Message Passing Interface* (MPI) [12] offers a set of *collective communication and computation operations* that are eminently useful for expressing parallel computations. It is therefore important that MPI libraries implement these operations as efficiently as possible for the intended target architectures. Hence, algorithms that can fully exploit the given communication capabilities are needed. Consequently, recent years has seen a lot of activity on algorithms and implementations for MPI collectives for different communication architectures, see [3, 8, 10, 13] to mention but a few.

In this paper we give new algorithms with implementations for three important MPI collectives, namely `MPI_Bcast` (broadcast), `MPI_Reduce` (reduction to root) and `MPI_Scan`/`MPI_Exscan` (parallel prefix). The new algorithms are able to fully exploit bidirectional communication capabilities as offered by modern interconnects like InfiniBand, Myrinet, Quadrics, and the NEC IXS, and thus in contrast to many commonly used algorithms for these operations have the potential of achieving the full bandwidth offered by such interconnects. For broadcast this has previously been achieved also by other algorithms [1, 6, 16], but these are typically more complicated and do not extend to the reduction and parallel prefix operations. We believe that the results achieved for reduction and parallel prefix are the currently best known. The algorithms are still simple to implement, and have been implemented within the framework of NEC proprietary MPI libraries.

2 Two pipelined binary trees instead of one

To explain the new algorithm we focus on the `MPI_Bcast` operation, and illustrate the improvement achieved by comparing to a *linear pipeline* and a *pipelined binary tree*. We let p denote the number of processors which are numbered from 0 to $p-1$, and m the size of the data to be broadcast from a given root processor r. As in `MPI_Bcast` we assume that all processors know p, r and m. We assume that communication is homogeneous, one-ported, and bidirectional in the sense that each processor can at the same time send a message to a processor and receive a message from another, possibly different processor.

Assume first that $p = 2^h - 1$ for some tree height $h > 1$. A binary tree broadcast algorithm uses a balanced, ordered binary tree rooted at processor r. To broadcast, the root sends its data to its left and right child processors, and upon receiving data each processor which is not a leaf sends data to its left and right child processors in that order. As can be seen the rightmost leaf receives data at communication step $2h$. Assuming that the time to transfer data of size m is $\alpha + \beta m$, the total broadcast time is $2h(\alpha + \beta m)$. The binary tree algorithm can be pipelined, and sending instead the m data as N blocks of size m/N yields a broadcast time (for the rightmost leaf) of $2(h+N-1)(\alpha+\beta m/N)$ (even exploiting bidirectional communication: each interior processor receives a new block from its parent at the same time as it sends a block to its right child). Balancing the terms $N\alpha$ and $(h-1)\beta m/N$ yields a best broadcast time of $2(h-1)\alpha + 4\sqrt{(h-1)\alpha}\sqrt{\beta m} + \mathbf{2}\beta\mathbf{m}$. This is roughly a factor of two from the lower bound of $\min\{\alpha h, \beta\mathbf{m}\}$.

Using instead a linear pipeline in which processor i receives a new block from processor $i-1$ and a the same time sends the previous blocks on to processor $i+1$, a broadcast time (for the last processor) of $(p-2)\alpha + 2\sqrt{(h-1)\alpha}\sqrt{\beta m} + \beta\mathbf{m}$ can be achieved. This algorithm has asymptotically optimal broadcast time βm but at the cost of a large latency term $(p-2)\alpha$ and is therefore interesting mostly when m is large compared to p.

We are interested in algorithms that combine the low latency of the pipelined binary tree and the optimal time achieved by the linear pipeline, while retaining as far as possible the implementation simplicity of these algorithms (as well as other properties such as low demands on bisection bandwidth, embeddability in non-homogeneous networks).

The problem with the pipelined binary tree algorithm 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 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[3] from which parent a block is received and to which child the previous block is sent so that each communication step consists in at most one send and at most one receive operation for each processor.

For now we (w.l.o.g) assume that $r = p - 1$ and construct the two binary trees T_1 and T_2 over processors $0 \ldots p-2$ as follows. Let 2^h be the smallest power of two larger or equal to $p - 1$ and let $P = 2^h - 1$.

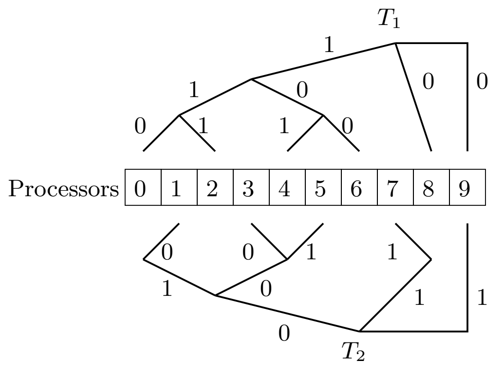

Fig. 1. The two inorder binary trees T_1 (top to bottom) and T_2 (bottom to top) with broadcast root $r = 9$ for $p = 10$. A 0/1 coloring is also shown.

1. Construct an inorder numbered balanced tree of size P, such that that the leaves have even numbers $0, 2, \ldots P - 2$.
2. If $P > p - 1$ eliminate nodes starting from the highest numbered node u as follows: if u is a leaf simply remove it; otherwise set the parent of u's (only) child to be the parent of u and remove u. This tree is T_1.
3. Construct an inorder numbered balanced tree of size P.
4. Shift the tree one position to the left, e.g. rename node u to $u - 1$. Remove the leftmost leaf -1. In this tree leaves have odd numbers $1, 3, \ldots$.
5. If $P - 1 > p - 1$ eliminate nodes from the right as in Step 2. This tree is T_2.

To complete the construction for the broadcast algorithm, the roots of the two trees are connected to the broadcast root node $p - 1$. An example of the construction for $p = 10$ is shown in Figure 1. The construction obviously has the desired property that leaves of T_1 are interior nodes of T_2 and vice versa.

To use the two trees for broadcast, the idea is to pipeline half the data through T_1, and the other half of the data through T_2. With bidirectional communication capabilities, the two pipelines can run concurrently, yielding a broadcast time of

[3] note that no explicit global synchronization is necessary. Point-to-point communication suffices to implicitly synchronize all processors to the extent necessary.

$2[2(h-1)\alpha + 4\sqrt{(h-1)\alpha}\sqrt{\beta m/2} + 2\beta m/2] = 4(h-1)\alpha + 4\sqrt{(h-1)\alpha}\sqrt{\beta m/2} + \beta\mathbf{m}$. This is half the time of the pipelined binary tree at the expense of only twice the (still) logarithmic latency.

The left to right order of the children is not sufficient to avoid that a node in the same communication step receives a block from both of its parents (which would compromise the analysis and slow down the algorithm). We need an even-odd (0/1) coloring of the parent to child edges without such conflicts. Even (0) edges are then used in even communication steps, odd (1) edges in odd steps. More precisely 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*

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 $\{s_i, r_j\} \in E$ iff j is a successor of i in T_1 or T_2. This graph models the sender role of processor i with node s_i and the receiver role of processor 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.

Computing the coloring as described in the proof can be done in $O(p)$ steps. In the appendix we outline how (with another construction of the two trees) each processor i can compute all relevant information (neighbors in the trees, colors of incident edges) in time $O(\log p)$ given only p and i.

2.1 Broadcast

If the broadcast root is not $r = p - 1$ as in the above, we simply renumber each processor i to instead play the role of processor $(i - (r + 1)) \bmod p$. To be useful in MPI libraries the coloring should be done at communicator construction time so that the coloring time is amortized over all subsequent broadcast operations on the communicator.

For broadcast the idea of using two trees to improve bandwidth was previously introduced in [5], but the need for coloring was not realized (due to the TCP/IP setting of this work).

2.2 Reduction

Let $\oplus$ denote an associative (possibly commutative) binary operation, and let processor i have a data vector m_i of size m. The reduction to root operation `MPI_Reduce` computes $\oplus_{i=0}^{p-1} m_i$ and stores the result at the root processor r. Assuming $r = 0$ or $r = p-1$, the two tree construction can be used for reduction to root by reversing the flow of data from the leaves towards the root in both of the trees. Since the trees are inorder numbered the reduction order can be maintained such that only associativity of $\oplus$ is exploited. We have the following Theorem.

Theorem 1. *For $r = 0$ or $r = p - 1$ reduction to root of data m_i (each of size m) can be done in communication time $4(h-1)\alpha + 4\sqrt{(h-1)\alpha}\sqrt{\beta m/2} + \beta \mathtt{m}$. The amount of data reduced per processor is $2m$.*

If the operator $\oplus$ is commutative the result can be achieved for any root r.

2.3 Scan

The (inclusive) parallel prefix operation `MPI_Scan` computes $\oplus_{i=0}^{j} m_i$ for each processor j. Using an inorder binary tree parallel prefixes can be computed by first performing an *up-phase* in which each interior node computes a partial sum $\oplus_{i=\ell}^{r}$ for left- and rightmost leaves ℓ and r, followed by a *down-phase* in which prefixes of the form $\oplus_{i=0}^{\ell-1}$ are sent down the tree and allow each processor to finish computing its parallel prefix. Both phases can be pipelined, and for each N blocks can be completed in $2h - 1 + 2(N - 1)$ communication steps. This is explained in more detail in [7, 11]. With the two tree construction two up-phases and two down-phases can take place simultaneously. This halves the number of steps for N blocks for each phase. The total number of steps required for the parallel prefix computation is therefore $4h - 2 + 2(N - 2)$. This is roughly two thirds the $4h - 2 + 3(N - 1)$ steps required by the *doubly pipelined parallel prefix algorithm* described in [11].

Theorem 2. *The parallel prefix operation on data m_i can be done in communication time $2((2h-3) + 2\sqrt{(2h-3)\alpha}\sqrt{\beta m/2} + \beta m$. The amount of data reduced per processor is (at most) $3m$.*

3 Experimental results

The two tree algorithms for `MPI_Bcast` and `MPI_Reduce` have been implemented within proprietary NEC MPI implementations. Experiments comparing the bandwidth achieved with the new algorithms to other, commonly used broadcast and reduction algorithms 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.

3.1 Broadcast

We compare to the following algorithms:

- *Circulant graph* first presented in [16]. This algorithm has asymptotically optimal completion time, and only half the latency of the two tree algorithm presented here, but is significantly more complex and requires a more involved precomputation than the simple coloring needed for the two tree algorithm.
- *Scatter-allgather* for broadcast [2] as developed in [14]. We also contrast to the implementation of this algorithm in MVAPICH.
- Simple *binomial tree* as in the original MPICH implementation [4].
- Pipelined binary tree
- Linear pipeline

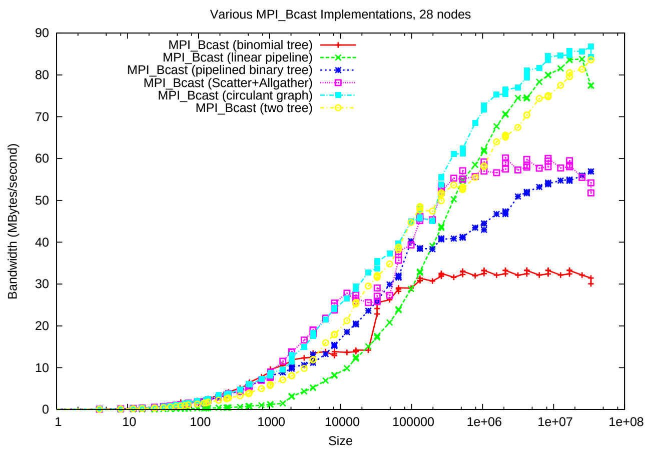

Fig. 2. Broadcast algorithms on the AMD/Myrinet cluster, 28 nodes.

Bandwidth results for the two systems are shown in Figure 2 and Figure 3. On both systems the two tree 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. 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.

The two tree 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 [16], but have to be combined with a binomial tree algorithm for small to medium sized prob-

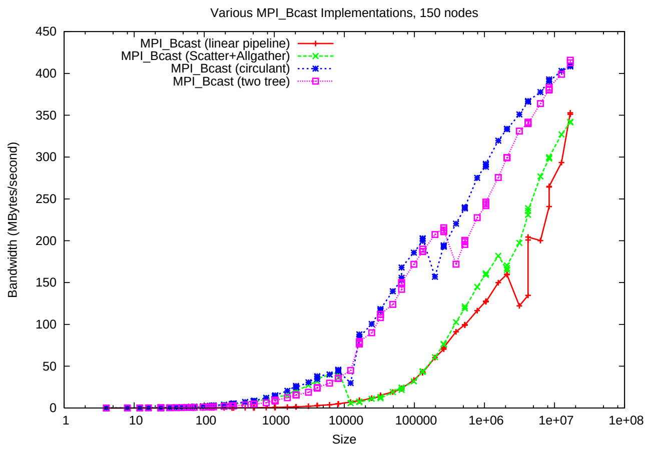

Fig. 3. Broadcast algorithms on the Xeon/InfiniBand cluster, 150 nodes.

lems. Being a pipelined algorithm with small blocks of size $\Theta(\sqrt{m})$ it is also well suited to implementation on SMP clusters [15].

3.2 Reduction

We compare to the following algorithms:

- Improved *butterfly* [9]
- *Binomial tree*
- *Pipelined binary tree*
- *Linear pipeline*

Bandwidth results for the two systems are shown in Figure 4 and Figure 5. The two tree 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 512Mbyte (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 considerable difference of almost a factor 2 between the two implementations of the butterfly algorithm (with the implementation of [9] being the faster). 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 better be countered.

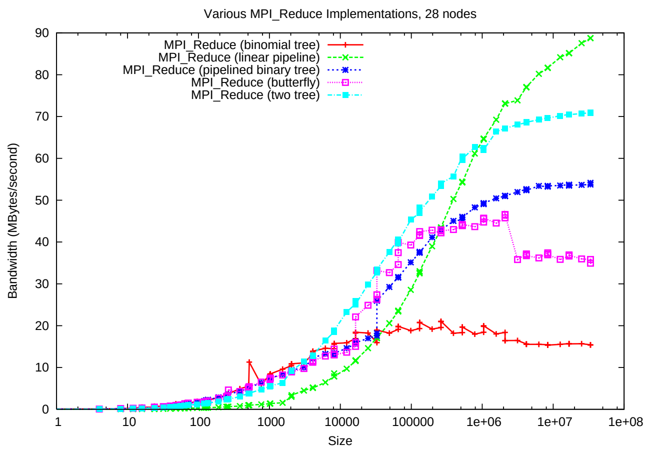

Fig. 4. Reduction algorithms on the AMD/Myrinet cluster, 28 nodes.

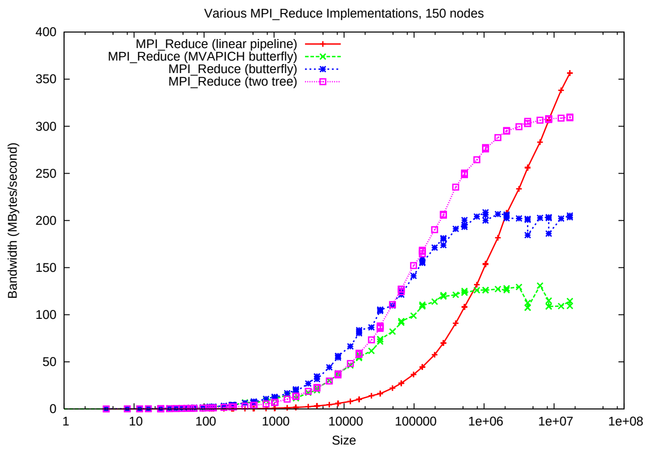

Fig. 5. Reduction algorithms on the Xeon/InfiniBand cluster, 150 nodes.

3.3 Coloring

Table 1 compares the preprocessing times for a simple linear time implementation of two tree coloring and the $O(p \log p)$ time algorithm for computing the block schedule required for the circulant graph algorithm from [16]. The coloring algorithm is always faster than the block scheduling algorithm. It is to be expected that with the logarithmic time coloring algorithm, the speed difference would become quite dramatic for large p. However, for currently used machine sizes, both scheduling times are not a big issue when they are only needed once for each communicator.

Processors	Two tree 0/1-coloring	Circulant graph block schedule
100	71.75	99.15
1000	443.37	1399.43
10000	13651.42	20042.28
100000	209919.33	248803.58
1000000	1990228.74	3074909.75

Table 1. Computing times in microseconds on an AMD 2.1GHz Athlon processor for the precomputation of double-tree coloring and block schedule.

4 Conclusion

We presented a new, simple algorithmic idea for broadcast, reduction and parallel prefix operations as found in MPI. The theoretical result and achieved performance for `MPI_Bcast` is similar to that achieved by other, recent, but more complicated algorithms, whereas the results for reduction to root and parallel prefix are presumably the best currently known. We expect the same for an implementation of the parallel algorithm.

References

1. 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.
2. 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.
3. E. W. Chan, M. F. Heimlich, A. Purkayastha, and R. A. van de Geijn. On optimizing collective communication. In *IEEE International Conference on Cluster Computing (CLUSTER 2004)*, 2004.
4. 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.
5. H. H. Happe and B. Vinter. Improving TCP/IP multicasting with message segmentation. In *Communicating Process Architectures (CPA 2005)*, 2005.

6. O.-H. Kwon and K.-Y. Chwa. Multiple message broadcasting in communication networks. *Networks*, 26:253–261, 1995.
7. 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.
8. J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra. Performance analysis of MPI collective operations. In *International Parallel and Distributed Processing Symposium (IPDPS 2005), Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO)*, 2005.
9. 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, 2004.
10. 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.
11. 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 *Lecture Notes in Computer Science*, pages 49–57. Springer, 2006.
12. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core. MIT Press, second edition, 1998.
13. 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.
14. 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, 2004.
15. J. L. Träff and A. Ripke. An optimal broadcast algorithm adapted to SMP-clusters. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 12th European PVM/MPI Users' Group Meeting*, volume 3666 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2005.
16. J. L. Träff and A. Ripke. Optimal broadcast for fully connected networks. In *High Performance Computing and Communications (HPCC'05)*, volume 3726 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2005.

Logarithmic time Scheduling Algorithm

The algorithm below assumes that p is even and T_2 is the mirror image of T_1. The tree construction is a straight forward recursive algorithm. In the full paper we prove the (non obvious) correctness of the simple coloring algorithm below.

```
**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
    $i' := \begin{cases} i - 2^h \text{ if } 2^{h+1} \text{ bitand } i = 1 \lor i + 2^h > p \\ i + 2^h \text{ otherwise} \end{cases}$      -- compute parent of $i$
    **Return** inEdgeColor$(p, i', h)$ xor $(p/2 \bmod 2)$ xor $[i' > i]$
```