Efficient and Scalable Hardware-Based Multicast in Fat-Tree Networks

Salvador Coll, *Member*, *IEEE*, Francisco J. Mora, *Senior Member*, *IEEE*, Jose Duato, *Member*, *IEEE*, and Fabrizio Petrini, *Member*, *IEEE*

Abstract—This article presents an efficient and scalable mechanism to overcome the limitations of collective communication in switched interconnection networks in the presence of faults. Considering that current trends in supercomputing are moving toward massively parallel computers, with many thousands of components, reliability becomes a challenge. In such scenario, fat-tree networks that provide hardware support for collective communication suffer from serious performance degradation due to the presence of, even, a single faulty node. This paper describes a new mechanism to provide high-performance collective communication in such situations. The feasibility of the proposed technique is formally demonstrated. We present the design of a new hardware-based routing algorithm for multicast, that is at the base of our proposal. The proposed mechanism is implemented and experimentally evaluated. Our experimental results show that hardware-based multicast trees provide an efficient and scalable solution for collective communication in fat-tree networks, significantly outperforming traditional solutions.

Index Terms—Multicast, data communications, interprocessor communications, network communication, network problems, trees.

1 Introduction

During the last decade, trends in supercomputing have consolidated toward the use of parallel processing to solve computationally intensive problems. As a result, massive parallel machines have emerged as the most widely used high-performance computing platforms. These machines are characterized by the distribution of memory among an ensemble of, currently, thousands of computing nodes; in the future, many more. Since memory is physically distributed, nodes communicate by sending messages through a network. Communication operations may be either *point-to-point*, which involve a single source and a single destination, or *collective* [1], in which more than two processes participate.

An efficient and scalable implementation of collective communication is critical to large-scale machines, since it has a significant impact on the performance and scalability of both applications and system software.

1.1 Application Performance

Recent analytical [2] and experimental [3] studies conducted on ASC-class (Advanced Simulation and Computing Program) machines have shown that scientific codes spend a considerable part of their runtime executing collective communication, in some cases up to 70 percent. Proprietary meteorological codes studied at Cray Research were found to run 7 percent slower when, instead of using the T3E's barrier hardware, they utilized a 15 μsec slower all-software barrier [4]. As an example of the importance of collective communication, BlueGene/L implements two dedicated networks to support this type of operations [5].

1.2 System Software Efficient Support

Another issue that has often been neglected is the performance and scalability of the system software on large-scale machines. Here, system software refers to all software running on a machine other than user applications.

As we demonstrated in [6], [7], and [8], high-performance and scalable resource management (one of the most important system-level operations) can be achieved by using a small set of collective communication operations as building blocks, (e.g., scheduling two orders of magnitude faster than the best previously reported results).

1.3 Impact of Reliability Issues

The large total component count of massively parallel systems makes any assumption of complete reliability entirely unrealistic: though the mean time between failure (MTBF) for the individual components and the physical connections between them may be very high, the large component count in the system will inevitably lead to frequent failures. With the MTBF for today's 10-20 Tflops machines on the order of 10-40 hours [9] and application running times on the order of days or weeks, multiple failures can be expected during a single execution of a single application.

1.4 Problem Statement

In the supercomputing scenario, the Quadrics interconnection network (QsNet) holds a prevalent position due to its

- *S. Coll and F.J. Mora are with the Departamento de Ingenieria Electronica, Universidad Politecnica de Valencia, Camino de Vera s/n, E-46022 Valencia, Spain. E-mail: {scoll, fjmora}@eln.upv.es.*
- *J. Duato is with the Departamento de Informatica de Sistemas y Computadores (DISCA), Universidad Politecnica de Valencia, Edificio 1G, Camino de Vera s/n, E-46022 Valencia, Spain. E-mail: jduato@disca.upv.es.*
- *F. Petrini is with the Cell Solutions Department, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598. E-mail: fpetrin@us.ibm.com.*

Manuscript received 16 July 2008; revised 25 Sept. 2008; accepted 29 Sept. 2008; published online 7 Oct. 2008.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0268.
Digital Object Identifier no. 10.1109/TPDS.2008.228.

outstanding point-to-point [10] and collective communication [11], [12], [13]. From the point of view of the collective communication operations, QsNet completely fulfills applications and system software requirements, as shown by us in [12] and [14], thanks to a number of special hardware features. As an example, our experiments, conducted on one section of the ASC Q machine with 1,024 nodes (4,096 processors) [12], have shown that collective operations are achieved with a latency comparable to a single point-to-point communication in most commodity networks (e.g., the network is capable of synchronizing 1,024 nodes in less than 10 μs).

Although the performance and scalability features of the Quadrics interconnect collective communication are outstanding, it presents an important problem. The hardware mechanisms that are at the base of the collective communication primitives rely on the nodes taking part in the collective operation being physically consecutive. A single gap in the nodes allocated for the collective operation, for instance due to a faulty node or a job scheduling decision, makes the hardware support completely unusable. Slower traditional point-to-point tree-based [15] collective communication primitives are used in that case. According to our experiments, the performance penalty is significant: two times slower barrier, and eight times slower broadcast on a 32-node cluster [13]. This effect increases with the network size, since the hardware mechanisms provide very good scalability while the point-to-point tree-based primitives show a logarithmic performance degradation.

The above limitation becomes particularly relevant since a situation where the nodes allocated to a job are not consecutive, due to the presence of faults, is very likely to happen. In practice, the hardware collective communication mechanisms cannot be used most of the time. This has important implications: on the one hand, the applications performance can be significantly degraded; on the other hand, current state-of-the-art techniques developed to improve scalability, and performance of the system software may provide little benefits or even become not implementable.

In this paper, we show that it is possible to significantly enhance collective communication performance in the Quadrics network in those cases where the hardware support is not directly usable. In particular, in those cases where a few faulty nodes make the set of destinations of a collective operation to be not physically consecutive. We introduce an innovative collective communication mechanism for switched networks, called Hardware-Based Multicast Tree, that provides performance improvements over the existing alternatives while preserving the application source code, operating system internals and network architecture.

The rest of this paper is organized as follows: Section 2 briefly introduces the Quadrics interconnection network. Section 3 describes the hardware-based multicast tree developed by us, while Section 4 presents the algorithm to calculate optimal trees. Section 5 evaluates the performance of the proposed algorithm implementations to obtain multicast trees, while the evaluation results for hardware-based multicast trees are presented in Section 6. Finally, conclusions are drawn in Section 7.

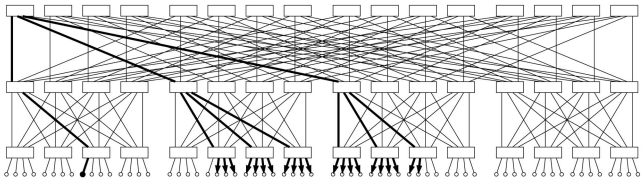


Fig. 1. Multicast in a 64-node fat-tree.

2 The Quadrics Interconnection Network

The Quadrics network is a butterfly bidirectional multistage interconnection network based on 4×4 switches, which can be viewed as a quaternary fat-tree. A quaternary fat-tree belongs to the more general class of the k-ary n-trees [16] (Fig. 1 shows a 4-ary 3-tree). Some of the most important aspects of this network are the integration of the local memory (either in the NIC or in the host) into a distributed virtual shared memory, the support for zero-copy remote DMA transactions, and the hardware support for collective communication.

The transmission of each packet is pipelined into the network using wormhole switching. Minimal routing between any pair of nodes is accomplished by sending the message to one of the nearest common ancestor switches and from there to the destination [17]. In this way, each packet experiences two routing phases: an adaptive ascending phase (in forward direction) to get to a nearest common ancestor, where the switches forward the packet through the least loaded link, and a deterministic descending phase (in backward direction) to the destination.

Hardware support of multicast communication requires increased functionality within the routers: the interpretation of multicast addresses and forwarding of messages onto multiple outgoing channels [18], [19]. The hardware multicast capability of the Quadrics network allows packets to be sent to multiple destinations. The switches can forward a packet to several output ports, with the only restriction that these ports must be contiguous. In this way, a group of adjacent nodes can be reached by using a single hardware multicast transaction.

The operation of the hardware-based multicast is outlined in Fig. 1. During the ascending phase, the nearest common ancestor switch for the source node and the destination group is reached. After that, during the descending routing phase, the packet spans the appropriate links to reach all the destination nodes.

3 Hardware-Based Multicast Trees

As stated in Section 1, the Quadrics network hardware support for collective communication can only be used when all the destination nodes are contiguous. Otherwise, a software-based multicast algorithm based on a balanced tree is used [13], where, when the number of nodes in the destination set is high, multicast latency and bandwidth

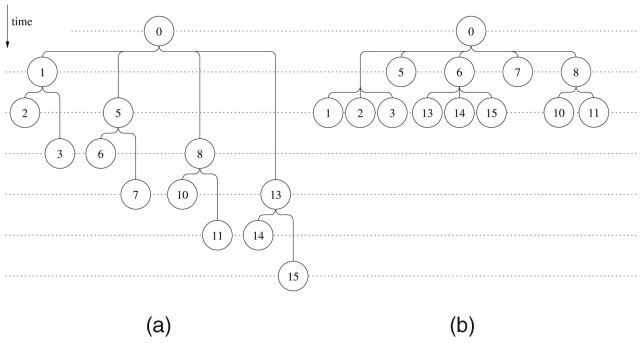


Fig. 2. (a) SW and (b) HW-based multicast trees.

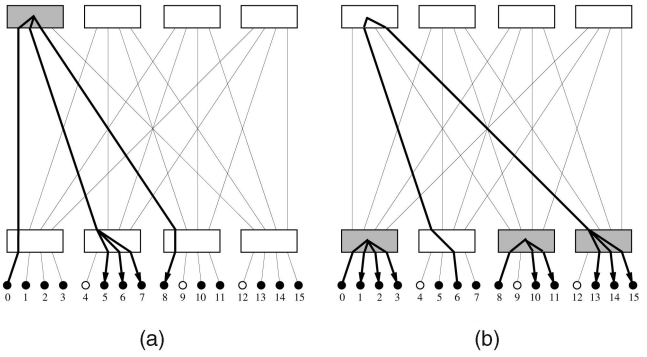


Fig. 3. Hardware-based multicasts.

consumption significantly increase with respect to the pure hardware solution.

Our alternative, Hardware-Based Multicast Trees, uses a multicast tree to distribute the data among all the destinations, but with the particular issue that each message is a hardware-based multicast. In this way, the source node performs a multicast DMA to a subset of contiguous destinations, each of which forwards the message to another subset. Eventually, all destinations will receive the message. This mechanism will be shown to provide significant performance improvements when the number of destinations is high and there are few gaps in the group allocation (e.g., due to some faulty nodes). The latency will be reduced while the available bandwidth will be increased because unicast messages are replaced with multicast messages, thus reducing the depth of the tree.

Example 1. Consider a multicast transaction on a 16-node network (4-ary 2-tree) with node 0 being the source ($S = 0$), and the set of destinations $D = \{1, 2, 3, 5, 6, 7, 8, 10, 11, 13, 14, 15\}$. Fig. 2 shows a diagram comparing the behavior of the original software-based tree (Fig. 2a), used by the Quadrics system software, and the proposed strategy (Fig. 2b). In the software tree, once each node receives the message from its parent, it forwards one copy to each one of its children. In the hardware tree, the source node sends a hardware multicast to one subgroup of adjacent nodes ([5, 8]) and, after that, each node with a copy of the message can forward the message to another subgroup of adjacent nodes. In this case, the tree requires only two steps and many less messages circulating through the network.[1]

The particular multicast transactions used in the example have been selected to provide the minimum tree depth, while fulfilling the default QsNet routing. These multicasts are detailed in Fig. 3a, for the first step, and Fig. 3b, for the second step, where the switches that initiate packet replication are highlighted.

As can be seen, a key factor to achieve the best possible performance is the parallel transmission of as many multicast transactions as possible during a tree step. This requirement has two significant consequences: first, deadlock-freedom must be guaranteed, since a situation where multiple multicast packets proceed in parallel may lead to deadlock [15]; second, for multicasts to be routed in parallel, without serialization at any switch or link, routing collisions must be avoided. Finally, given a multicast problem, defined by a source node and a set of destinations, the sequence of steps that allows a message to be delivered in the shortest time requires careful calculation.

3.1 Avoiding Deadlocks

A communication example which involves two multicast groups with a deadlocked situation is presented in Fig. 4. The gray and black nodes represent two groups with partial overlap of destinations, the leftmost node in each group being the source. In the depicted situation, none of the transactions can proceed since both of them have been allocated only a fraction of the output ports they require, at the bottom switches, to reach the destinations.

In order to guarantee deadlock-freedom, the QsNet routing mechanism establishes a limitation that is based on the concept of *Broadcast Tree* and *Root Switch*.

Definition 1. *A* Broadcast Tree, *in a fat-tree network topology, is composed of the links and switches that provide a descending path between the top leftmost switch in the network and all the nodes. Fig. 5 highlights the default broadcast tree for a 64-node network.*

Definition 2. *A* Root Switch *for any given group of nodes refers to the nearest common ancestor switch which belongs to the broadcast tree for that group of nodes. Fig. 5 shows the root switch (filled) for the group of nodes highlighted in black (same as the group used in the example in Fig. 1).*

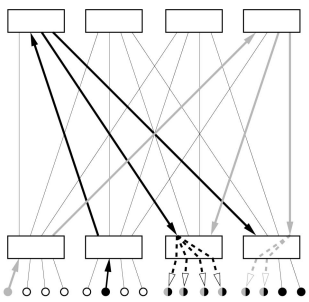


Fig. 4. Deadlocked situation.

1. This example assumes the latency for a multicast is equal to that of a point-to-point message. This will be justified in Section 6.

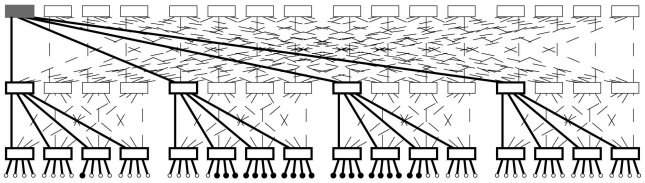


Fig. 5. Broadcast tree and root switch example.

The deadlock avoidance routing limitation for collective communication in the QsNet is defined in Proposition 1.

Proposition 1. *Multicast deadlock-free routing is guaranteed if a multicast packet can only take paths included in the broadcast tree, both in the ascending and the descending routing phase.*

Proof. This proof is based on the fact that the switches serialize incoming packets (both point-to-point and multicast) if there is some overlap among the requested output ports; otherwise, they can proceed in parallel. As multicast packets always follow the broadcast tree, simultaneous multicasts are serialized at or before the root switch and, therefore, deadlocks are avoided. □

According to that, the QsNet multicast routing is formalized in Algorithm 1, where $rs(g)$ denotes the root switch of a group of destinations, g, and $rs(s, g)$ denotes the root switch of a source node, s, and a group of destinations, g.

Algorithm 1. QsNet multicast routing
Input: source node s and group of adjacent destinations g
Procedure:

1) Send the message following the broadcast tree in the ascending direction until the root switch (rs) of the source node and the destination group $rs(s, g)$ is reached.
2) At this point a turnaround routing step is performed.
3) If $rs(s, g) \neq rs(g)$ then send the message down the broadcast tree to $rs(g)$.
4) Forward the message down the broadcast subtree of group g spanning the appropriate set of adjacent links to reach the destinations.

This routing mechanism establishes a path between the source and the destinations which is a generalization of the turnaround routing path [15] used in bidirectional multistage interconnection networks.

As an original contribution of this work, it is possible to define a more flexible routing mechanism for hardware-based multicasts on the Quadrics network, while still guaranteeing deadlock-freedom. We propose a new routing restriction for hardware-based multicasts.

Proposition 2. *Multicast deadlock-free routing is guaranteed if a multicast packet can only take paths included in the broadcast tree, once the replication to several output ports at the destinations root switch starts.*

Proof. According to Proposition 2, the route followed by a multicast packet to reach the root switch of the destinations is not restricted exclusively to the broadcast tree [20], [21], as required by Proposition 1. A multicast packet can be routed as a unicast packet while it has not reached the root switch, and can follow paths not included in the broadcast tree. Deadlocks are not possible in that part of the path because the unicast routing algorithm in QsNet is deadlock-free. During the rest of the path to the destinations, the packet is replicated to multiple output ports. All the multicast messages destined to the same multicast subtree will use the same root switch, even if the corresponding destination sets are disjoint. If there is some overlap among the requested output ports, multicast packets will be sequenced through that root switch, thus avoiding deadlocks. If there is not, they will be able to proceed in parallel. To sum up, deadlocks are not possible during the unicast part of the route because the unicast routing is deadlock-free, and deadlocks are not possible during the multicast part of the route because multicasts in conflict will be serialized through the overlapping root switch. □

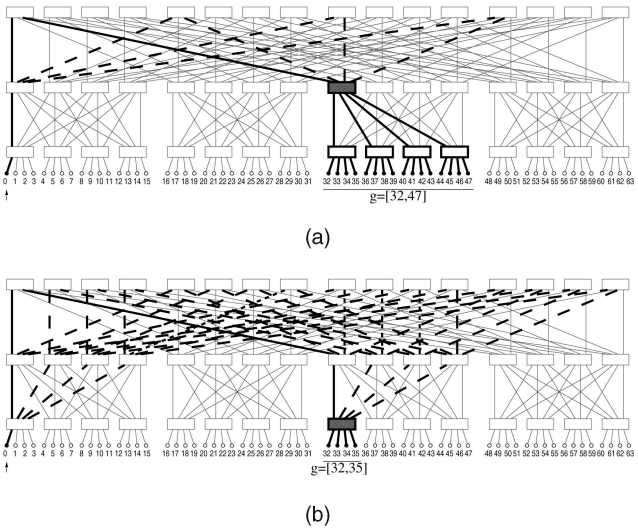


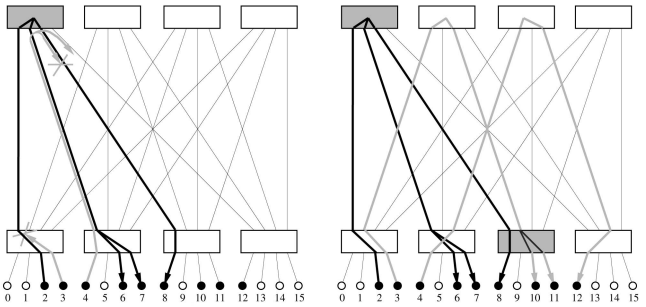
Fig. 6. Original and improved multicast routing.

As a consequence, we propose and formalize a new routing algorithm for hardware-based multicasts, defined by Algorithm 2.

Algorithm 2. Improved QsNet multicast routing
Input: source node s and group of adjacent destinations g
Procedure:

1) Send the message following the broadcast tree in the ascending direction until a switch at the same level l of $rs(g)$ is reached.
2) If $rs(s, g) \neq rs(g)$ then adaptively route the message to a switch at the same level as $rs(s, g)$.
3) At this point a turnaround routing step is performed.
4) If $rs(s, g) \neq rs(g)$ then send the message down to $rs(g)$.
5) Forward the message down the broadcast subtree of group g spanning the appropriate set of adjacent links to reach the destinations.

A direct result is that the proposed routing algorithm provides, in those situations where the source node and the destinations belong to different subtrees, multiple alternative paths (not always disjoint) that will balance load across channels. In addition, the increased path diversity makes possible that multiple multicast transactions proceed in parallel with no serialization in the broadcast tree. Fig. 6



(a) (b)

Fig. 7. Partitioning the multicast destination set. (a) Subset {10, 11, 12} cannot be reached. (b) Subset {10, 11, 12} can be reached.

shows two examples with the available multicast routes when using the original routing (continuous lines) and when using the improved routing (dashed lines).

From the examples shown it can be easily derived that the number of alternative paths provided by our routing algorithm, for a k-ary n-tree, is k^d, where d is the difference between the root switch level of the destinations and the root switch level of the destinations and the source. The increased number of alternative paths provided by Algorithm 2 not only reduces contention of multicasts with other traffic, but it also facilitates multiple nonoverlapping multicasts to proceed in parallel.

3.2 Avoiding Serialization

As it has been described, a step of the hardware-based multicast tree consists, in general, of a set of parallel hardware multicasts sent by a number of nodes to disjoint sets of contiguous destinations. For those multicasts to be routed in parallel, and avoid serialization, routing collisions among the transactions must be avoided.

In Example 1, the multicast destination set has been initially partitioned into the minimal number of subsets of consecutive nodes, and this partitioning has been maintained during the multicast tree. The result is a two-step multicast tree, which is optimal in terms of latency (a nonconsecutive set of destinations cannot be reached in less than two transmission steps). However, in a general case, it is not guaranteed that the initial partitioning will be collision-free once the multicast tree starts.

Example 2. Consider a hardware-based multicast tree step where nodes $S = \{2, 3, 4\}$ already have a copy of the message and the set of pending destinations is $D = \{6, 7, 8, 10, 11, 12\}$. As shown in Fig. 7a, if, for instance, node 2 sends a multicast message to subset {6, 7, 8}, node 3 cannot reach subset {10, 11, 12} since there is an overlap in the ascending route with node 2 message. Same situation happens with node 4, with an overlap on the descending route. The destination subsets in D cannot be simultaneously reached since there is an overlap in the required paths. This situation would require an additional tree step to complete the multicast. However, an alternative technique can be used. If subset {10, 11, 12} is split into two smaller subsets, {10, 11} and {12}, all the destinations can be reached during the same tree step (shown in Fig. 7b). In this case, the overlaps have been removed by splitting one of the original subsets of adjacent nodes into smaller ones. In this example, two different types of overlaps or potential collisions can be identified: the overlap on the route from nodes 2 and 3 to subset {10, 11, 12} that arises during the ascending routing phase and the overlap on the route from nodes 2 and 4 to subset {10, 11, 12} that appears during the descending routing phase. The first overlap will be referred to as *forward routing overlap* and the second one as *backward routing overlap*.

According to the proposed strategy, on a hardware-based multicast tree, the destination set is initially partitioned into the minimal number of subsets such that each subset consists of consecutive nodes. However, as has been shown in Example 2, this is not enough because when nodes that already receive a copy of the multicast message during previous tree steps try to forward it to other sets of consecutive nodes, the corresponding routes may overlap, thus requiring additional tree steps to reach all the destinations. Example 2 illustrates that appropriately splitting subsets of adjacent nodes into smaller subsets provides a mechanism to remove overlaps. As a consequence, the partitioning of the initial destination set into an overlap-free partition during each tree step is an important issue to be considered since this may provide lower depth trees, and achieve faster solutions. A detailed formal analysis of k-ary n-trees and routing overlaps, and the methodology defined to appropriately partitioning the destination set to remove the overlaps is included in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.228.

4 Optimal Hardware-Based Multicast Trees

In the context of multicast trees, the solution is a sequence of decisions that defines which groups of adjacent nodes are reached from which nodes, at each tree step (an example is shown in Fig. 2b). In order to find the shortest depth multicast tree, a backtracking approach that tries all the possible solutions to the problem has been designed and implemented. An additional refinement introduced in the proposed algorithm is that after a first solution to the problem is found, only partial solutions that improve the current best solution are explored. In this way, the depth of the search tree is reduced.

The proposed backtracking algorithm (Algorithm 3) is based on the sequential checking of all possible combinations of groups of adjacent nodes reachable in parallel from the nodes that received the message at a previous tree step. In this way, on the first iteration, given that only the source node has a copy of the message to be sent, the algorithm selects one group of adjacent nodes as destination for the first step of the tree. No restriction applies in this case since any group of adjacent nodes can be reached in a single step using a hardware-base multicast. For the second step of the tree, there must be, at least, one additional group of adjacent nodes. Otherwise, the solution to the multicast problem is trivial since all the destinations could be reached with a single step. At this point, the algorithm must verify if the pending destinations are reachable in parallel from the original source

node and the group of destinations selected for the first step of the tree. Theorem 3 and Corollary 8 provide the basic mechanisms to verify the parallel reachability of a set of groups of adjacent destinations from an arbitrary set of sources. If any routing overlap, either backward or forward, is detected, it is removed using the techniques defined by Corollaries 4 and 6. If all the destinations can be reached with only two tree steps, that is an optimal solution since a nonadjacent set of destinations cannot be reached in less than two steps, using hardware-based multicasts. In that case, the algorithm can finish with no additional exploration since there are no solutions with shorter tree depth. On the contrary, if additional steps are required, the algorithm continues till a solution is found. When a solution improving the first one and requiring more than two steps is found, the algorithm backtracks two steps in the search tree and continues from there looking for better solutions. Backtracking a single step in the tree, when a solution is found, is useless since all the solutions from that point will have higher than or equal depth.

A further refinement of this algorithm is obtaining a good first solution. In this way, the space of solutions to be explored is significantly reduced. The heuristic used to find a good solution for the first iteration of the algorithm consists of sorting the original list of groups of adjacent nodes in decreasing order of sizes (groups with equal sizes are ordered in increasing order of root switch level). This heuristic is described in detail in Section 4.2, since it was originally devised for the greedy algorithm. The intuition behind this heuristic is that by selecting larger groups first, the multicast tree depth should be decreased, thus providing better solutions.

Algorithm 3. Optimal hardware-based tree preparation
Input:
source nodes list ($srcs$),
destination nodes list ($dests$),
sublist of destination nodes to be tested ($dests\_to\_try$)
Output:
partial solution being obtained (sol),
number of steps of the partial solution ($steps$),
best found solution ($best\_sol$),
number of steps of the best found solution ($best\_steps$)
Note:
the destination nodes list is sorted before the initial call according to number of nodes and root switch level
Procedure optimal_tree
copy srcs, dests, dests_to_try lists to local variables;
while (!$last\_combination$ && ($steps < best\_steps - 1$))
$dests\_to\_try$ = next combination of $|srcs|$ groups out of $dests\_to\_try$;
update $last\_combination$;
if backward_routing_overlap($dests\_to\_try$) **then**
/* generate two alternative lists where the overlap is removed */
backward_routing_overlap_resolution($dests\_to\_try$, $left\_try$, $right\_try$);
/* check two alternative branches of the solutions tree */
optimal_tree($srcs$, $dests$, $right\_try$, sol, $steps$, $best\_sol$, $best\_steps$);
optimal_tree($srcs$, $dests$, $left\_try$, sol, $steps$, $best\_sol$, $best\_steps$);
else
if (forward_routing_overlap($srcs$, $dests\_to\_try$, $limited\_level$)) **then**
for each group in $dests\_to\_try$ with level $\geq limited\_level$
/* generate an alternative list where the overlap is likely to be removed */
$dests\_to\_try$ = split_a_group($dests\_to\_try, group$);
optimal_tree($srcs$, $dests$, $dests\_to\_try$, sol, $steps$, $best\_sol$, $best\_steps$);
endfor
else
/* groups in dests_to_try are reachable in parallel, account for a new tree step */
$steps$++;
/* update dests and srcs with reached groups */
update_dests($dests$, $dests\_to\_try$);
update_srcs($srcs$, $dests\_to\_try$);
update_solution(sol);
if ($dests \neq \emptyset$) **then** /* still pending destinations */
optimal_tree($srcs$, $dests$, $dests$, sol, $steps$, $best\_sol$, $best\_steps$);
else
if ($steps < best\_steps$) **then**
/* a better solution is found */
update_solution($best\_sol$, sol);
endif
endif
endif
endif
endwhile
/* explore an alternative branch of the solutions tree */
$steps$- -;
copy $srcs$, $dests$, $dests\_to\_try$ lists to local variables;
end

4.1 Routing Overlaps

As stated in Algorithm 3 description above, the parallel reachability of a given set of destination groups from a given set of source nodes must be checked. For doing so, two conditions are verified: *backward routing overlap* (Definition 12) and *forward routing overlap* (Definition 13). The overlap detection mechanisms implemented in the algorithm are based on Theorem 3 and Corollary 8. Combinations of groups with overlaps are not part of valid multicast trees. Nevertheless, overlaps can be removed by splitting some groups into smaller groups in such a way that the selected combination of groups (or part of it, if not enough source nodes are available at a particular tree step) can be reached.

In the case of overlaps, the algorithm could have tried all the possible combinations of groups split into all the possible ways. But this would have increased the computational cost of the backtracking algorithm to even higher levels. An approach that takes into account the specific properties of the overlaps has been developed instead. For this reason, the routing overlaps have been analyzed in the

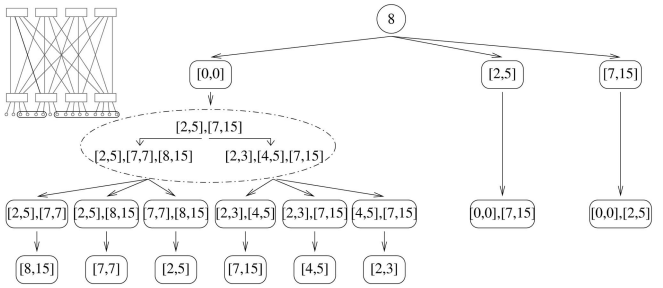


Fig. 8. Example of the backtracking algorithm operation with backward routing overlap.

Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.228, where Corollaries 4 and 6 have been proposed for removing backward and forward routing overlaps, respectively.

Algorithm 3 applies the routing overlap detection and removal mechanisms in a predefined order. First, if a backward routing overlap is detected for a particular set of groups, those groups are partitioned according to Corollary 4 and the overlap is removed. Later, if a forward routing overlap is detected, the destination groups are partitioned based on Corollary 6 to remove the overlap. As the backward routing overlap takes place in a later stage in packet routing, when the overlapping groups are split the resulting groups are bigger than the groups that would be obtained if the forward routing overlap was removed first. As the goal is finding the minimum steps multicast tree, reaching larger groups as soon as possible will provide better solutions.

When a backward routing overlap is detected, the algorithm proceeds in the following way. The combination of preselected groups (which is implemented using a list) is checked for overlaps among groups. When the first overlap is found, two lists of preselected groups are generated by splitting some groups (based on Corollary 4). Those two lists provide two alternative branches of the search tree. The remaining groups in the list are not rearranged at this point, if additional overlaps exist, they will be detected later. After that, two recursive calls are made with the two newly generated lists (that is, two branches of the solution tree are explored) and the procedure is executed until an overlap-free list of groups is selected. This technique guarantees that a backward routing overlap-free combination from the list of pending destinations, of size the number of source nodes, will be selected.

Example 3. Fig. 8 shows the solutions tree explored by the algorithm on a 16-node network with the following conditions: source node 8; original groups of adjacent destination nodes {[0, 0], [2, 5], [7, 15]}. The circle represents the source node, rounded boxes denote group/s of adjacent nodes that can be reached at a given multicast tree step, and arrows represent algorithm decisions. Any path between the tree root to a leaf represents a solution to the hardware-based multicast tree problem. For instance, following the rightmost branches, the solution is: during the first step of the tree, node 8 sends the packet to group [7, 15] and during the second step, two nodes in {[8, 8], [7, 15]} send two copies of the packet to groups [0, 0] and [2, 5], respectively. As can be seen, the number of rounded boxes between the root node and a leaf is equal to the tree depth, which is equivalent to the number of steps of the multicast tree (two in the commented example).

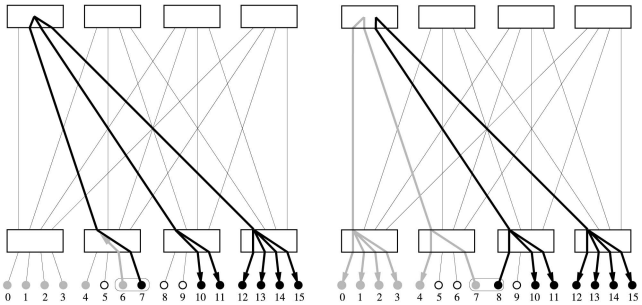
The dotted ellipse highlights the point in a partial solution where a backward routing overlap is detected. Group [2, 5] and group [7, 15] share a link in their broadcast trees at switch level 1, as it is indicated in the network representation on the top left corner of the figure (the overlapping groups are highlighted). The overlap resolution mechanism performs two different rearrangements of the preselected groups ([2, 5], [7, 15]), as defined by Corollary 4: first, group [7, 15] is split into groups [7, 7] and [8, 15]; second, group [2, 5] is split into groups [2, 3], [4, 5]. From this point, two different solution subtrees are explored, as indicated in the figure, by performing two recursive calls to the backtracking procedure with the new two lists ([2, 5], [7, 7], [8, 15] and [2, 3], [4, 5], [7, 15]). As can be seen, by splitting the overlapping groups in this way, two alternative backward routing overlap-free group arrangements are explored. In this example, both of them provide the same tree depth but this might not be the case in a more complex scenario.

When a given set of potential destinations is backward overlap free, forward routing overlaps are checked. Forward routing overlaps are detected using Lemma 7. When this overlap is detected, the algorithm selects all the groups with root switch level lower than or equal to the level where the overlap is detected. Then, each one of the groups is partitioned to produce subgroups with the root switch at an immediate lower level. Each partitioning opens a different branch in the search tree. After that, a new call to the backtracking procedure is performed. If no additional overlaps are detected the algorithm proceeds normally. This procedure is repeated for all the groups with root switches at the level where the limitation is detected and lower, one by one. This is done in this way to generate all the possible group arrangements that remove the detected overlap. If the resulting list of destinations has additional overlaps, they will be detected and removed on a later recursive call.

4.2 Greedy Algorithm

This section describes a greedy algorithm to calculate hardware-based multicast trees that has been developed to provide an implementation with practical computational cost. The structure of this algorithm is based on the backtracking approach presented in previous section. The main difference is that no additional solutions are explored after the first solution is found. This solution will be demonstrated to be a good solution to the problem of finding the optimal hardware multicast tree.

The key factor of the greedy algorithm is a heuristic for choosing the group (or set of groups) to be reached during each step of the multicast tree. At any step of the hardware multicast tree, the optimal subset of groups of destination nodes to be selected is not known. Nevertheless, two properties of any given destination group give an indication of the potential capabilities of that group to reach additional destinations during subsequent multicast tree steps:



(a) (b)

Fig. 9. Hardware-based multicast capabilities of two subgroups composed of two nodes.

- *The number of nodes in the group*. The larger the groups, the more destinations they are capable to reach in parallel. This indicates the maximum number of multicasts that might be transmitted in parallel in the next step by the corresponding group. This maximum number could be reached if there was no routing overlap.
- *The level of the root switch for the group*. This parameter gives an indication of the routing alternatives from that group to a given root switch (and, hence, to groups of adjacent nodes). This is related to the concept of routing capabilities presented in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.228, and is further explained below using the example in Fig. 9.

Fig. 9 compares the routing capabilities of two groups composed of two nodes in a 16-node system. Fig. 9a shows a 2-node group (group [6, 7]) with the root switch located at level 1. This configuration allows reaching only one group with the root switch at level 0 and another group with the root switch at level 1 (two groups with the root switch at level 1 are reachable too). Fig. 9b shows a 2-node group with the root switch at level 0 (group [7, 8]). In this case, two groups with their root switch at level 0 are reachable in parallel, provided they do not show backward routing overlap. All the combinations of two groups with root switches at higher levels are possible as well, that is: one group with the root switch at level 0 and one group with the root switch at level 1, or two groups with the root switch at level 1. As can be seen, the routing capabilities of the group in Fig. 9b include those provided by the group in Fig. 9a.

In general, in order to compare the capabilities of two disjoint groups of adjacent nodes to reach additional destinations, several situations are possible:

- One of the groups has lower root switch level and more nodes. That group is the best one because its routing capabilities include and overcome all the others.
- Two groups have the same root switch level. In this case, it is clear that the best group is the one with more nodes.
- Two groups have the same amount of nodes and different root switch level. The best one (in terms of multicast capabilities) will be the group with the lower root switch level.
- In all the other cases, the best group depends on the remaining destinations.

TABLE 1
Group with Better Capabilities to Reach Destinations in Parallel

	$L_1 = L_2$	$L_1 < L_2$	$L_1 > L_2$
$\|g_1\| = \|g_2\|$	g_1, g_2	g_1	g_2
$\|g_1\| > \|g_2\|$	g_1	g_1	?
$\|g_1\| < \|g_2\|$	g_2	?	g_2

Table 1 summarizes the possibilities presented above, indicating which group (g_i) is the best one according to the root switch level $(L_i = rsl(g_i))$ and the cardinality.

Taking into account the above considerations, the heuristic applied in the proposed algorithm consists of sorting the original list of destination groups in decreasing order of nodes and increasing order of root switch level. In this way, the algorithm, whose structure is based on the backtracking implementation, selects groups of destinations in the established order. If any routing overlap problem arises, it is solved in the same way used for the backtracking approach, with the difference that only the first alternative is explored.

The computational cost of the greedy algorithm is proportional to the multicast tree depth, and thus to the logarithm of the number of groups of adjacent destinations. This provides an approach that can be implemented in practice with low computational cost, with the limitation that finding the optimal solution is not guaranteed. Nevertheless, in Section 5, it is shown that the greedy implementation always finds optimal solutions for networks below 4,096 nodes with less than 0.8 percent faulty nodes.

As an example, the solution to a multicast problem on a 64-node network where node 63 must send a message to the set of destinations $D = \{[0, 13], [15, 27], [29, 46], [48, 49], [51, 62]\}$ is shown in Fig. 10. The solution consists of a two-step tree, which is optimal provided the set of destinations is not contiguous. In this case, the group selected as first destination is [29, 46]. Fig. 10b shows the message routing on a network diagram. During the second and final steps, the following transactions are performed in parallel: node 63 sends a copy of the message to group [0, 13], node 29 to group [16, 27], node 30 to node 15, node 31 to group [48, 49], and node 32 to group [51, 62]. Fig. 10c shows a possible routing for the given transactions, since packets sent by nodes {30, 32, 31} are routed adaptively during all of or part of the route.

5 Algorithm Evaluation

As the greedy algorithm proposed to calculate hardware-based trees does not guarantee optimal solutions, an evaluation of the algorithm ability to obtain optimal solutions has been performed. This has been done by comparing the solutions the greedy algorithm obtains with the optimal solution found using the backtracking algorithm. Tests have been conducted on configurations ranging from 16 to 4,096-node networks, with faulty node percentages ranging from 0.1 percent to 10 percent. One thousand

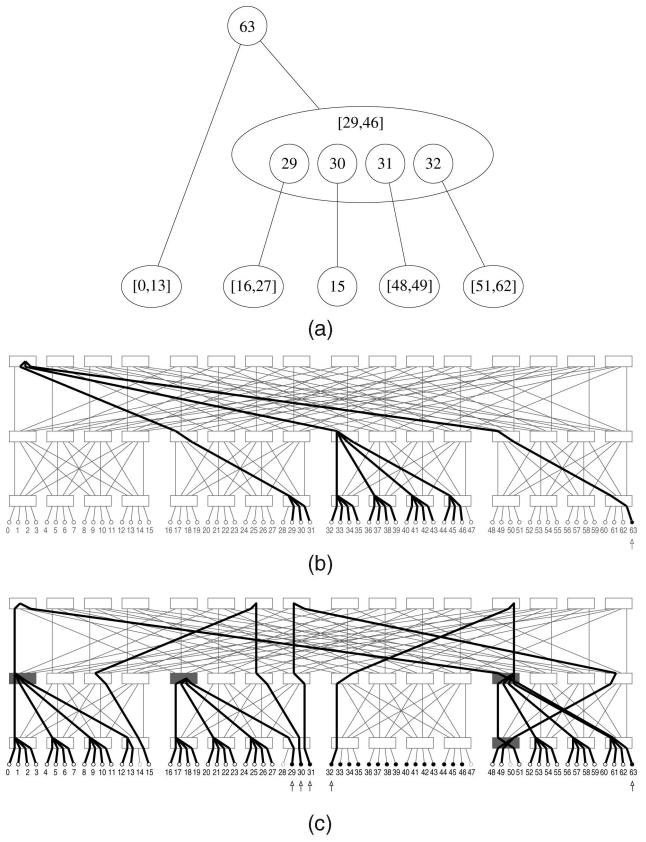


Fig. 10. Optimal tree for a multicast problem where the source is $S = 63$ and the destination is $D = \{[0, 13], [15, 27], [29, 46], [48, 49], [51, 62]\}$.

random faulty node mappings have been generated for each configuration.

Our tests show that the greedy algorithm finds an optimal solution for 99 percent of the cases. This percentage increases as the amount of faulty nodes decreases, being 100 percent when less than 1 percent of the nodes are faulty. A summary of the results is shown in Fig. 11. Each grayscaled area covers the configurations where the worst case tree depth (2, 3, or 4) is the same. It has to be mentioned that, as Fig. 11 shows worst-case values, the closer a given configuration is to the left border of the area it belongs, the higher the probabilities for that configuration to be solved in one step less.

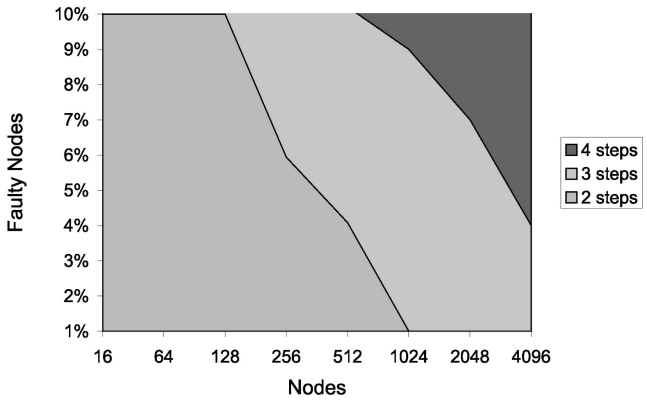


Fig. 11. Maximum steps required by a hardware-based multicast tree.

TABLE 2
Optimal Tree Steps for Different Configurations

Network Size	Faulty Nodes 0.2%	0.4%	0.6%	0.8%	1%
1024	2	2	2	2	2
2048	2	2	2	2	2
4096	2	2	2	2	2/0.582 3/0.418

Detailed results of multicast trees on large-scale QsNet clusters are shown in Table 2. These tests have been designed to analyze the greedy algorithm performance when a fraction of the nodes is not operational. Networks with 1,024, 2,048, and 4,096 nodes have been simulated. The number of nodes not taking part in the multicast transaction has been varied from 0.2 percent to 1 percent in 0.2 percent intervals. One thousand random problems for each particular configuration have been solved using the algorithms presented in this paper. The number of steps for the optimal trees are summarized in the table, where the notation x/y denotes an x-step tree with probability y.

The results show that any practical situation, with a realistic amount of faulty nodes, is likely to be solved using a hardware-based multicast tree with only two steps. Only the largest configuration tested, 4,096 nodes, will require three steps for 42 percent of the cases when 1 percent of the nodes are faulty. This result is extremely significant since, in practice, any situation where some faulty nodes avoid the utilization of hardware support for collective communication can be solved using a hardware-based tree which is optimal in terms of latency.

6 Experimental Evaluation

The experiments described in this section were performed on the *crescendo* cluster (Performance and Architecture Laboratory, Los Alamos National Laboratory) with the characteristics shown in Table 3.

The tested network configurations ranged from 2 to 32 nodes. Bandwidth and latency results are shown for those tests involving data exchange. Latency is reported for the case of barrier synchronization tests. Unless otherwise stated, results correspond to the average metric obtained over 10,000 repetitions of the test. To isolate the network

TABLE 3
Platform Characteristics

Component	Characteristic
Node	
Type	Dell 1550
CPU	Dual 1.13 GHz Pentium-IIIs
Memory	1GB ECC RAM
OS	Red Hat 7.1 Linux
Network	
Type	QsNet
Size	32 nodes
Topology	4-ary 3-tree
NIC	QM-400 Elan3 NIC
Interface	66MHz/64-bit PCI
Communication	Elanlib QsNet messaging layer

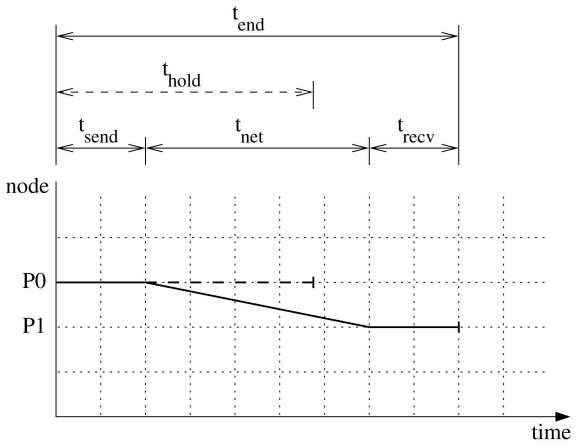


Fig. 12. The unicast communication model.

performance from the effect of the software implementation, the lowest level programming libraries have been used for the benchmarks.

6.1 Parameterized Communication Model

In order to obtain optimal, or near-optimal, hardware-based multicast trees for any communication pattern, an accurate communication model of the network is required. This model provides an abstraction of the underlying network and such a technique has been demonstrated to be very useful in the design of optimal multicast trees [22]. The parameterized communication model for abstract parallel architectures, which is based on the LogP model [23], characterizes the point-to-point communication of a network for both 1-port and α-port architectures, and is at the base of the mechanisms used to obtain optimal software-based multicast trees.

The unicast, or point-to-point, communication is the basic communication pattern of all communication subsystems. We will use the model of the unicast communication as a reference for modeling the multicast communication. The model for unicast communication is based on five parameters:

- Sending latency (t_{send}) is the latency in preparing the message to be sent including the overhead of packetization, checksum computation, etc.
- Network latency (t_{net}) is the time required to transmit the message across the network.
- Receiving latency (t_{recv}), similar to t_{send}, is the overhead at the receiver.
- End-to-end latency (t_{end}) is the interval between the sender starts sending a message until the receiver finishes receiving it.
- Holding time (t_{hold}) is the minimum time interval between two consecutive send or receive operations.

The timing for sending a single message from the sender (P_0) to the receiver (P_1) according to the proposed model is shown in Fig. 12. Measuring t_{send}, t_{net}, and t_{recv} is rather difficult as it would require some special techniques such as using a hardware monitor and a software probe. On the other hand, the measurement of t_{end} and t_{hold} is quite simple and can be performed at user-level. These two parameters have been shown in [22] to completely represent the network performance under point-to-point communication and are particularly useful to design optimal multicast trees. For this reason, we will exclusively use the multicast versions of t_{end} and t_{hold} in our model.

TABLE 4
QsNet Multicast Latencies for a 32-Node Network

Message Size (bytes)	t_{mend} (μs)	t_{mhold} (μs)
64	2.95	2.80
256	3.54	3.38
1024	6.09	5.94
4096	15.50	15.35
16384	53.24	53.10
65536	205.76	205.66

The purpose of our model is the characterization of the hardware-based multicast service provided by the network, by extending the point-to-point model. This service is used by the two basic collective communication primitives implemented by the QsNet programming libraries, barrier and multicast, when the processes in the group are located in adjacent nodes.

Similar to the unicast communication model, the multicast model has two network parameters:

- Multicast end-to-end latency (t_{mend}) is the elapsed time since the multicast is issued until the last processor receives the message.
- Multicast holding time (t_{mhold}) is the minimum time interval between two consecutive multicast operations.

The measurement of t_{mend} is performed by a unidirectional ping-pong hardware-based multicast test. In this test, two root nodes alternatively send a multicast message to the same group of destinations. Each root processor belongs to the destination set of the other root processor, in this way each root processor waits for the completion of the previous multicast to become a sender. By measuring the average latency at one end-point and dividing it by two, the end-to-end latency is obtained.

t_{mhold} is measured by a multicast ping test in which the sender tries to perform hardware-based multicasts to the same set of destinations as soon as it can.

Average results for 100,000 multicast tests over a 32-node group with messages ranging from 64 byte to 64 Kbytes are shown in Table 4. These results show negligible differences between the two latencies. The latency is constant below 64-byte messages, which are packetized in a single transaction (and always 64 bytes are transmitted), and grows linearly with the message length above 64 bytes.

As an example, results for the multicast end-to-end latency with 4 Kbytes messages versus the number of nodes are depicted in Fig. 13. As can be seen, the latency remains almost constant as the number of destination nodes is increased. The most significant effect in latency is due to the number of switch layers traversed to reach the destinations. This effect can be clearly seen in Fig. 14, where the t_{mend} latency is shown for different group sizes. No significant differences arise for group sizes in the intervals [2, 4], [5, 16], and [17, 32]. From 2 to 4 nodes, one switch layer is

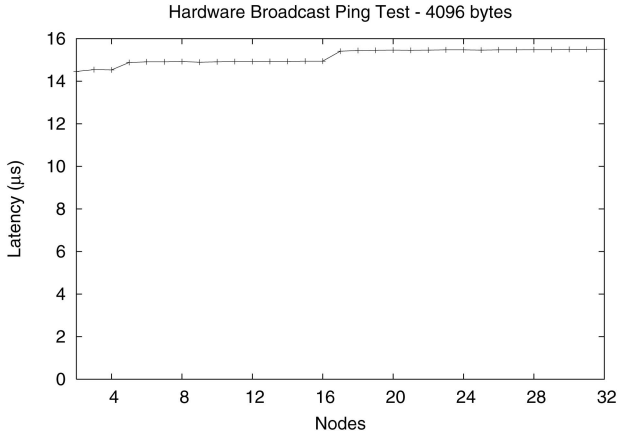


Fig. 13. End-to-end multicast latency.

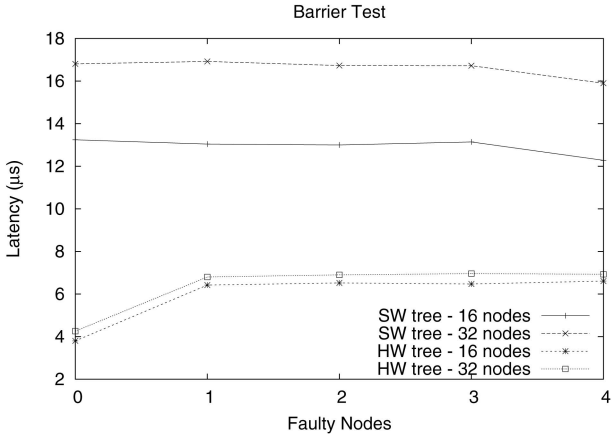


Fig. 15. Barrier synchronization latencies.

traversed, with 5 to 16 nodes two layers are needed, while with 17 to 32 nodes three layers are used.

The main conclusion of the parameterized multicast communication model is that the holding time for the multicast is approximately equal to the end-to-end latency. Again, this issue has an important implication in the construction of optimal multicast trees. In particular, for any given step of the multicast tree, either the source nodes of previous multicast steps, or the destinations can be used as sources for the next step of the tree, with no significant performance penalty.

Another important result is the fact that a hardware multicast takes approximately the same time to complete, no matter what the destination group size is (Figs. 13 and 14). Taking into account these considerations, hardware-based multicast trees with fewer steps will always provide shorter latencies.

6.2 Hardware-Based Multicast Trees Evaluation

As described in Section 1, hardware-based multicast trees provide a mechanism to reduce the impact on performance of collective communication resulting from destination sets that are not formed by consecutive nodes. In particular, our proposal is focused on those cases where a few nodes are excluded from a collective operation. The most representative example of such a situation is the case where a large-scale cluster has a few faulty nodes.

To evaluate the performance and scalability of hardware-based multicast trees, an experimental evaluation of the two basic collective communication primitives provided by the QsNet, barrier synchronization and broadcast, is presented in this section. Our proposal has been implemented in the QsNet system software and used in the implementation of barrier synchronization and broadcast (or multicast). In order to analyze the behavior of our proposal, results comparing the performance of the original mechanism implemented by the QsNet libraries are shown together with the result obtained using hardware-based multicast trees. As the QsNet collective communication is implemented on top of a software-based tree, for those cases where the hardware support is not directly usable, the corresponding results are indicated with the label "SW tree." On the other hand, results provided by the collective communication services using hardware-based multicast trees are labeled as "HW tree."

In this section, we evaluate the impact of our proposal on the barrier synchronization and broadcast when a few faulty nodes avoid the destination set to be composed of adjacent nodes. Experiments have been conducted using configurations on 16 and 32-node networks. The number of nodes not taking part into the operation has been varied from zero, which produces a contiguous set of destinations, to four. The results have been obtained by averaging 10,000 repetitions of the test using random distributions of the faulty nodes. The tree construction time is not taken into account, neither for the software- nor the hardware-tree, since it is computed once for each configuration.

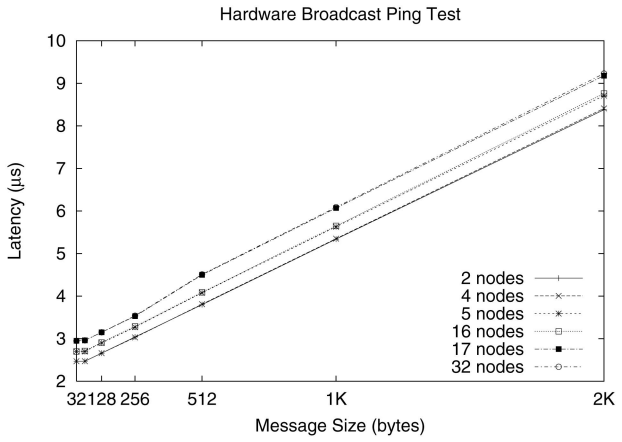


Fig. 14. End-to-end multicast latency for different group size intervals.

6.2.1 Barrier Synchronization

Fig. 15 shows the average time to perform a barrier synchronization on an empty network. Results for 16- and 32-node network configurations are shown versus the number of faulty nodes. As expected, the latency of the software-based barrier used by the QsNet original libraries is insensitive to the number of nodes not taking part in the collective communication, when this is low. The latency for both network configurations remains approximately constant when the faulty nodes range from zero to four. This is due to the fact that a slight reduction on the destination nodes has no significant effect on the multicast tree depth [13].

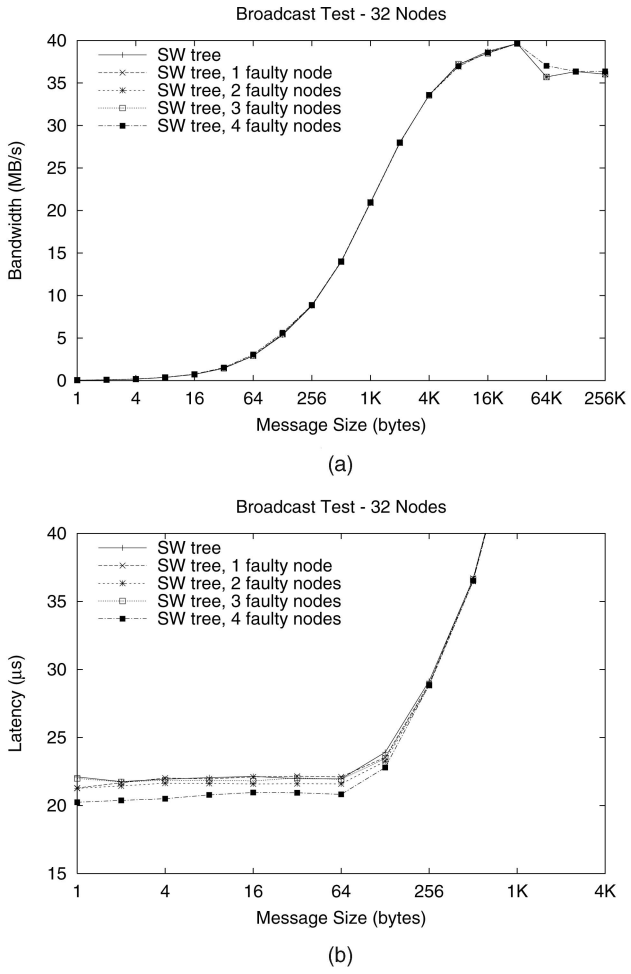


Fig. 16. Software-based broadcast.

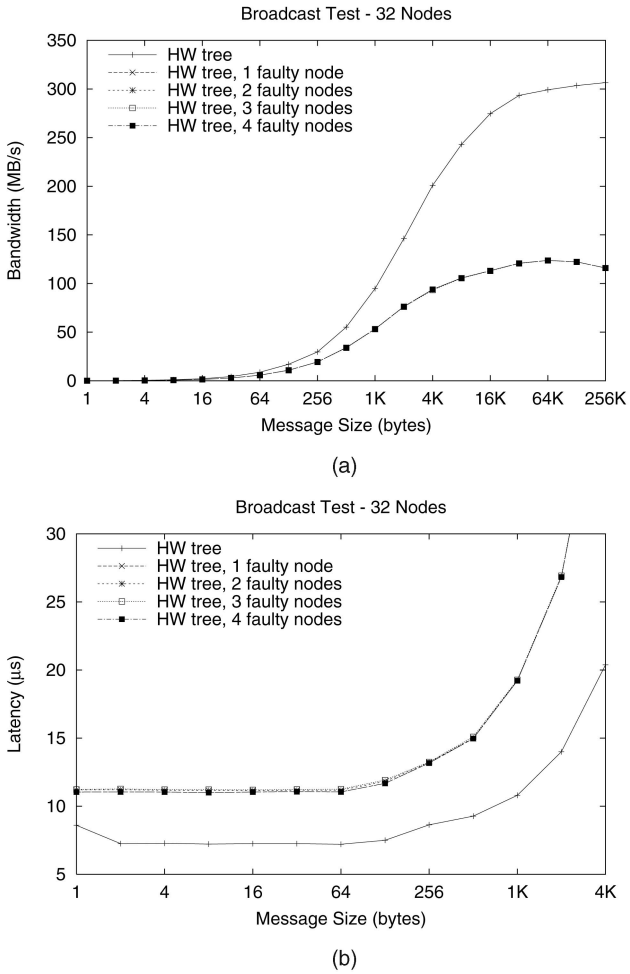


Fig. 17. Hardware-based broadcast.

On a 16-node configuration with less than four faulty nodes, the latency is constant at approximately 13 μs; while it decreases to 12 μs when four faulty nodes are considered. For a 32-node configuration, the latency is constant at 17 μs, except for four faulty nodes where 15.9 μs are obtained.

On the other hand, it can be seen that the synchronization based on the hardware tree developed by us, significantly outperforms the software tree used by the QsNet. The latency required to barrier synchronize with no faulty nodes is 4 μs in both configurations tested (16 and 32-node networks), which corresponds to a single multicast step. The time required to barrier synchronize the network with four faulty nodes or less is constant at 6.5 μs for 16 nodes and 7 μs for 32 nodes, since only two multicast steps are required. As can be seen, when the number of faulty nodes is low, our methodology provides between 50 percent and 60 percent faster synchronizations, for the configurations tested. We expect this performance gap to broaden with larger network configurations, since, as shown in Table 2, the scalability of our multicast mechanism is very good, while we have shown in [12] that the hardware support for multicast on the Quadrics network provides outstanding performance on extreme-scale configurations.

6.2.2 Broadcast

Fig. 16 shows the results obtained with software-based broadcasts over a 32-node network, and buffers globally allocated in main memory (that is, with the same virtual address in all processes). As the results obtained for the barrier experiments in previous section, the performance of the software-based multicast is insensitive to the number of faulty nodes when there are only a few. Only the configuration with four faulty nodes, as can be seen from Fig. 16b, shows a slight improvement in latency for messages shorter than 128 bytes. The peak bandwidth of 40 Mbyte/s is obtained for 32-Kbyte messages, while messages shorter than 64 bytes are delivered in less than 22 μs.

When all 32 nodes in the network take part in the collective communication, the hardware-based multicast can be directly used. In this case, a peak bandwidth of 306 Mbytes/s has been measured for 256-Kbyte messages (top curve in Fig. 17a). This result illustrates the maximum bandwidth achievable using the hardware support for multicast. On the other hand, if several faulty nodes are considered, our hardware-based multicast tree is used, since the default mechanism is not directly usable. The results obtained are shown in Fig. 17. As can be seen, the performance of the multicast tree developed by us is insensitive to the number of faulty nodes for the configurations we tested, since the destinations can always be reached in two multicast steps. The measured bandwidth of the hardware-based multicast tree reaches a peak of 124 Mbytes/s for 64-Kbyte messages. For the tested message sizes, the bandwidth provided by the hardware-based multicast tree, with faulty nodes, ranges between

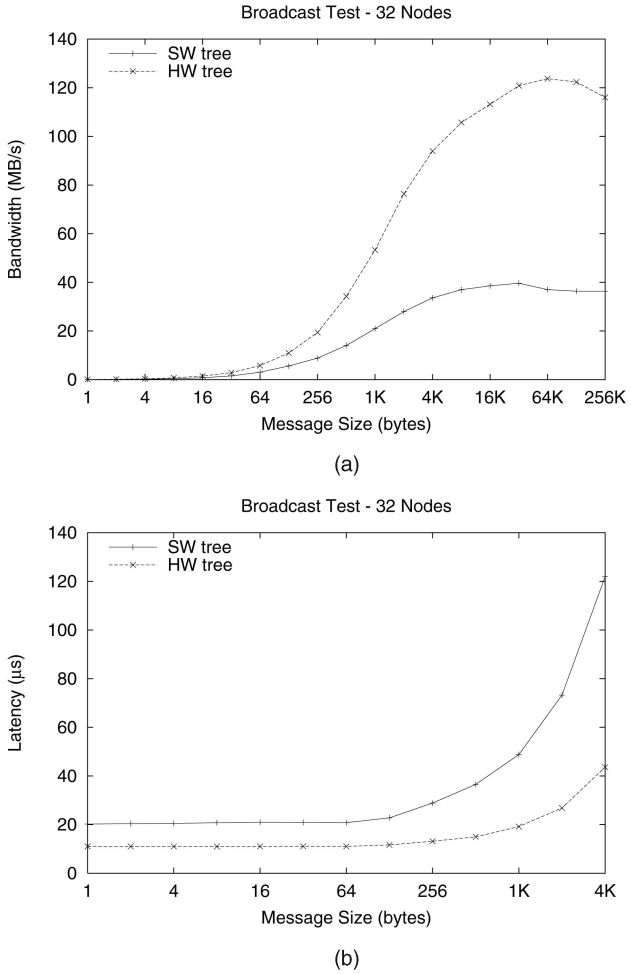


Fig. 18. Hardware and software-based broadcast comparison.

60 percent and 40 percent of the bandwidth obtained with no faulty nodes. In terms of latency, a close look to the results for short messages shows that the latency for messages shorter than 64 bytes remains constant at 11 μs.

For the sake of clarity, Fig. 18 shows the broadcast results for tests on a 32-node network with four faulty nodes using both the QsNet software-based tree and our hardware-based tree. As mentioned above, the performance benefits of our approach range between doubled bandwidth for short messages and tripled bandwidth for long messages.

These results show that the hardware-based multicast tree significantly outperforms the software-based multicast tree, providing a collective communication that is twice as fast in the worst case of short messages. For longer messages, the performance benefits of our proposal are evident since the peak bandwidth is tripled.

7 Conclusions

After being verified in Section 5 that the proposed mechanism, promised significant performance gain to collective communication, an in-depth experimental performance evaluation is presented. Our results for the two basic collective communication primitives, barrier synchronization and broadcast, obtained on a 32-node cluster, show that the hardware-based multicast tree mechanism significantly outperforms the QsNet software-based multicast (barrier latencies are reduced to a half, while broadcast bandwidths are doubled, and even tripled for long messages). As a result, collective communication on a QsNet-based system, where some nodes are faulty, will not be dramatically degraded.

This effect is likely to increase as the network size increases due to the poor scalability of the software-based multicast when compared to the hardware-based multicast tree we propose. We have demonstrated that two-step multicast trees will cover most of the situations. And, two-step hardware-based multicast trees are optimal in terms of latency. That means that the hardware-based multicast tree mechanism would significantly outperform the behavior of the software-based multicast, used by the QsNet, on a large-scale network. These results show that hardware-based multicast trees provide an efficient, scalable, and fault-tolerant alternative to software-based multicast trees, while overcoming the limitations of the hardware support for collective communication.

References

[1] P.K. McKinley, Y. jia Tsai, and D.F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *Computer*, vol. 28, no. 12, pp. 39-50, 1995.

[2] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings, "Predictive Performance and Scalability Modeling of a Large-Scale Application," *Proc. ACM/IEEE Conf. Supercomputing (SC '01)*, Nov. 2001.

[3] F. Petrini, D. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," *Proc. ACM/IEEE Conf. Supercomputing (SC '03)*, Nov. 2003.

[4] S.L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Proc. Seventh Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '96)*, pp. 26-36, 1996.

[5] The BlueGene/L Team, "An Overview of the BlueGene/L Supercomputer," *Proc. ACM/IEEE Conf. Supercomputing (SC '02)*, Nov. 2002.

[6] E. Frachtenberg, D. Feitelson, J. Fernández, and F. Petrini, "Parallel Job Scheduling under Dynamic Workloads," *Proc. Ninth Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, in conjunction with HPDC12/GGF8, June 2003.

[7] E. Frachtenberg, F. Petrini, J. Fernandez, and S. Coll, "Scalable Resource Management in High Performance Computers," *Proc. Fourth IEEE Int'l Conf. Cluster Computing (CLUSTER '02)*, pp. 305-314, 2002.

[8] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll, "Storm: Lightning-Fast Resource Management," *Proc. ACM/IEEE Conf. Supercomputing (SC '02)*, Nov. 2002.

[9] D.J. Kerbyson, A. Hoisie, and H.J. Wasserman, "Use of Predictive Performance Modeling during Large-Scale System Installation," *Proc. First Int'l Workshop Hardware/Software Support for Parallel and Distributed Scientific and Eng. Computing*, Sept. 2002.

[10] F. Petrini, E. Frachtenberg, A. Hoisie, and S. Coll, "Performance Evaluation of the Quadrics Interconnection Network," *Cluster Computing*, vol. 6, no. 2, pp. 125-142, 2003.

[11] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie, "Hardware- and Software-Based Collective Communication on the Quadrics Network," *Proc. IEEE Int'l Symp. Network Computing and Applications (NCA '01)*, pp. 24-35, Oct. 2001.

[12] F. Petrini, S. Coll, J. Fernandez, and E. Frachtenberg, "Scalable Collective Communication on the ASCI Q Machine," *Proc. 11th Symp. High Performance Interconnects (HOTI '03)*, pp. 54-59, Aug. 2003.

[13] S. Coll, J. Duato, F. Mora, F. Petrini, and A. Hoisie, "Collective Communication Patterns on the Quadrics Network," *Performance Analysis and Grid Computing*, V. Getov, M. Gerndt, A. Hoisie, A. Malony, and B. Miller, eds., chapter I, pp. 93-107, Kluwer Academic, Sept. 2003.

[14] J. Fernández, E. Frachtenberg, F. Petrini, K. Davis, and J.C. Sancho, "Architectural Support for System Software on Large-Scale Clusters," *Proc. Int'l Conf. Parallel Processing (ICPP '04)*, Aug. 2004.

[15] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann, Aug. 2002.
[16] F. Petrini and M. Vanneschi, "Performance Analysis of Wormhole Routed k-Ary n-Trees," *Int'l J. Foundations of Computer Science,* vol. 9, pp. 157-177, June 1998.
[17] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro,* vol. 22, pp. 46-57, Jan./Feb. 2002.
[18] C.M. Chiang and L.M. Ni, "Multi-Address Encoding for Multicast," *Proc. First Int'l Workshop Parallel Computer Routing and Comm. (PCRCW '94),* pp. 146-160, 1994.
[19] R. Sivaram, D.K. Panda, and C.B. Stunkel, "Efficient Broadcast and Multicast on Multistage Interconnnection Networks Using Multiport Encoding," *Proc. Eighth IEEE Symp. Parallel and Distributed Processing (SPDP '96),* pp. 36-45, Oct. 1996.
[20] S. Coll, J. Duato, F. Petrini, and F.J. Mora, "Scalable Hardware-Based Multicast Trees," *Proc. ACM/IEEE Conf. Supercomputing (SC '03),* p. 54, 2003.
[21] J. Zhou, X.-Y. Lin, and Y.-C. Chung, "Hardware Supported Multicast in Fat-Tree-Based Infiniband Networks," *J. Supercomputing,* vol. 40, pp. 333-352, June 2007.
[22] J.Y.L. Park, H.-A. Choi, N. Nupairoj, and L.M. Ni, "Construction of Optimal Multicast Trees Based on the Parameterized Communication Model," *Proc. Int'l Conf. Parallel Processing (ICPP '96),* pp. 180-187, Aug. 1996.
[23] D. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. Fourth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPOPP '93),* pp. 1-12, May 1993.

Salvador Coll received the BS degree in electronic engineering and the MS and PhD degrees in computer science from the Universidad Politecnica de Valencia, Valencia, Spain, in 1992, 1996, and 2005, respectively. He is currently an associate professor in the Departamento de Ingenieria Electronica, Universidad Politecnica de Valencia. During 2001, he was a graduate research assistant at the Los Alamos National Laboratory. His research interests include multicomputer systems and interconnection networks, with special interest in routing algorithms and collective communications. He is a member of the IEEE and the IEEE Computer Society.

Francisco J. Mora received the MSc degree in telecommunication engineering from the Universidad Politecnica de Cataluña, Spain (and continued electronic engineering studies at the European Laboratory for Physics Research (CERN), Geneva, thanks to a "Doctoral Student" grant) and the PhD degree from the Universidad Politecnica de Valencia, Valencia, Spain, in 1997. He is currently a professor in the Departamento de Ingenieria Electronica, Universidad Politecnica de Valencia. His research interests include data acquisition system design in particle physics, multiprocessor systems, and interfacing. He is a senior member of the IEEE and the IEEE Computer Society.

Jose Duato received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. He is currently a professor in the Departamento de Informatica de Sistemas y Computadores (DISCA), Universidad Politecnica de Valencia, Valencia, Spain. He was also an adjunct professor in the Department of Computer and Information Science, Ohio State University. His current research interests include interconnection networks and multiprocessor architectures. He has published more than 400 refereed papers. He proposed a powerful theory of deadlock-free adaptive routing for wormhole networks. Versions of this theory have been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E supercomputer, the on-chip router of the Alpha 21364 microprocessor, and the IBM BlueGene/L supercomputer. He is the first author of the book *Interconnection Networks: An Engineering Approach*. He has served as a member of the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, and the *IEEE Computer Architecture Letters*. He was the general cochair for the 2001 International Conference on Parallel Processing, the program committee chair for the Tenth International Symposium on High Performance Computer Architecture (HPCA-10), and the program cochair for the 2005 International Conference on Parallel Processing. He also served as a cochair, a member of the steering committee, a vice-chair, or a member of the program committee in more than 55 conferences, including the most prestigious conferences in his area (HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, and HiPC). He is a member of the IEEE and the IEEE Computer Society.

Fabrizio Petrini is a senior researcher of the Cell Solutions Department, IBM T.J. Watson Research Laboratory, Yorktown Heights, New York. Before his appointment at T.J. Watson, he was a laboratory fellow at the Pacific Northwest National Laboratory, a member of the technical staff at the Los Alamos National Laboratory, and a research fellow at the Computing Laboratory of the Oxford University. His research interests include various aspects of multicore processors and supercomputers, including high-performance interconnection networks and network interfaces, fault tolerance, job scheduling algorithms, parallel architectures, operating systems, and parallel programming languages. He has received numerous awards from the US Department of Energy (DOE) for contributions to supercomputing projects, and from other organizations for scientific publications. He is an associate editor of the *IEEE Transactions on Parallel and Distributed Processing*. He is a member of the IEEE and the IEEE Computer Society.