Reliable Multicast Protocol Using Infiniband Remote Direct Memory Access

Shah Mansoor, J. Michael Meehan
Computer Science Department
Western Washington University
Bellingham, WA 98225
meehan@wwu.edu

Abstract

Commonly utilized communication patterns in algorithms for scientific computing on super-computer clusters make extensive use of one to many, and many to many messaging between nodes. Many of these codes are implemented using the standard MPI_BCAST (broadcast) library function. Unfortunately, the implementation of these operations in MPI relies upon point-to-point reliable level three transmissions rather than true hardware enabled level two multicast messaging. This is a result of the fact that hardware enabled multicast is not universally available in all systems and additionally it is datagram in nature. The semantics of MPI_BCAST must guarantee completion of delivery to all recipients before continuing the sending process although there is an issue concerning this with the way MPI_BCAST is implemented. We define and implement a reliable multicast protocol utilizing the hardware enabled multicast remote direct memory access capabilities (RDMA) of Infiniband connections commonly found in high-end computing clusters. Our protocol utilizes cumulative acknowledgments propagated up binomial trees. We present benchmark speedup results versus a standard MPI_BCAST implementation on a cluster equipped with an Infiniband switch. We demonstrate that our approach provides faster multicast messaging and guarantees delivery to all recipients before the transmitting process is continued.

keywords: Reliable Multicast, Infiniband networks, verbs library, MPI_BCAST, RDMA.

1. Introduction

The dominating limiting factor in many large-scale scientific calculations is the rate at which data can be transferred between nodes in a cluster. As the problem size scales to require a larger number of processors than can be effectively implemented with tightly-coupled shared memory multi-core and multi-processor approaches at a single node the limitations of inter-node bandwidth can dominate the overall computation time. It is therefore crucial to devise ways to minimize delays caused by inter-node messaging. Many commonly utilized algorithms in scientific computations make use of nodes *broadcasting* data to many other nodes. The standard tool utilized to implement these computations on current super-computer clusters is the Message Passing Interface (MPI).[10] While there are many commercial proprietary implementations of MPI such as ScaMPI from SCALI [11], MPI/Pro from MPI-Softtech Solutions [12], and MESH-MPI from MESH-Technologies [9] there are three widely used open-source implementations: MPICH [4], open-MPI [6] , and LAM-MPI [1]. We utilized open-MPI version 3.1 as the basis for our benchmark tests.

When a broadcast operation is carried out using the MPI library MPI_BCAST operation, level three point to point connections are utilized to deliver the message to the recipients.[1] The documentation and *man* pages indicate that the semantics of the MPI broadcast must *guarantee* that all recipients have *correctly* received the broadcast before the sending process continues.[2] Following is a quote from the man page for MPI_BCAST

```
"MPI_Bcast broadcasts a message from the process with rank root to all processes of the group, itself included.  It is called by all members of group using the same arguments for comm, root.  On return, the contents of root's communication buffer has been copied to all processes."
```

There is a problem with this absolute guarantee given the way that the MPI_BCAST is implemented. This is explained in detail below. The MPI_BCAST is implemented using reliable level three point to point connections even though networking technologies commonly utilized to connect systems in clusters have level two hardware supported multicast capabilities. While level two hardware support for multicast is not universally available for all network technologies the vast majority of network systems have some sort of level two multicast capability. The most ubiquitous network connection technology, Ethernet, has level two multicast capabilities. The degree to which there is hardware support for bridging level two and level three multicast varies depending on the type of switches utilizes. We concentrate our initial

[1] In this paper when we use the designation level three or level two to describe a message transmission we are referring to the International Standards Organization Open Software Interconnect model for protocol layer functionality. When we indicate level three we mean a reliable transport layer transmission. When we refer to level two we mean a datagram service.

[2] There is a non-blocking counterpart MPI_Ibcast

978-1-943436-12-5 / copyright ISCA, CAINE 2018
October 8-10, 2018, New Orleans, Louisiana, USA

implementation discussed in this work on Infiniband technology with Remote Direct Memory Access (RDMA) capabilities found in high-end computing clusters. The reliable multicast protocol presented here could be implemented on top of Ethernet multicast as well. Hardware enabled level two multicast capability is a level two datagram service. We seek to exploit these level two multicast capabilities in support of application level multicast semantics. We will present an implementation of a reliable multicast protocol that utilizes Infiniband hardware enabled multicast using Remote Direct Memory Access (RDMA) in conjunction with point to point reliable connections for propagating acknowledgments that will guarantee correct receipt of the multicast message before the transmitting process is resumed. We will demonstrate how by incorporating level two hardware enabled multicast we can achieve significant speedup over standard MPI_BCAST implementations.

2. MPI Implementation Specifics

In order to avoid the delays and communication bottleneck that would result from serially transmitting n copies of a message from a single source to all recipients of a broadcast, MPI implementations utilize a binomial tree to propagate broadcast messages. Note, however, that each of the connections in the binomial tree is a level three reliable point-to-point connection. No actual hardware enabled multicast is employed. We give a quick review of binomial trees and how they are used to implement multicast messages in MPI.

2.1 Binomial Trees

Definition: A binomial tree B_k may be defined recursively as follows: B_0 consist of a single node For $k \geq 1$, $\boldsymbol{B}_k$ is a pair of $\boldsymbol{B}_{k-1}$ trees, where the root of one $\boldsymbol{B}_{k-1}$ tree becomes the leftmost child of the other.

For example, $\boldsymbol{B}_1$ is a pair of $\boldsymbol{B}_0$ trees with one connected as the leftmost child of the other. You will observe that a binomial tree $\boldsymbol{B}_k$ has a height k and 2^k nodes. Additionally, $\boldsymbol{B}_k$ has

$$\binom{k}{j}$$

nodes, i.e. ***(k!/j!(k-j)!)***, on level j, whence the name. Refer to figure 1 below. The nodes labeled 0 through 3 constitute an order 2 binomial tree with the subtree of nodes labeled 2 and 3 forming an order 1 binomial tree attached to the order 1 binomial tree of nodes 0 and 1 at its root node 0. The nodes label 0 through 7 constitute and order 3 binomial tree. The entire figure represents an order 4 binomial tree.

2.2 Propagation of Messages in a Binomial Tree with MPI_BCAST

Referring again to figure 1 consider an MPI_BCAST originating from node 0. Observe the connecting arc labels to the right of the connecting lines. These numbers indicate how the MPI_BCAST is propagated through the binomial tree. The broadcast begins with node 0 (the originator of the broadcast) sending the message to node 8. During the second phase node zero sends the message to node 4 and node 8 sends its copy of the message to node 12. You can observe that there will be four phases of message transmissions through the binomial tree to propagate the message to all recipients. Each of the message transmissions represented by the lines in the figure is a reliable level three connection.

2.3 Loss of Guaranteed Delivery Before Continue Semantics

The speedup achieved in employing an underlying binomial tree to propagate a broadcast versus a serial transmission to each of n nodes is significant requiring only $\log_2 n$ transmission periods rather than serially transmitting n messages. Such an implementation does, however, create a problem with faithfully implementing a guarantee that the originating process is not continued until all nodes have received the broadcast. A beneficial side-effect of the inefficient serial transmission to n nodes utilizing a level three reliable transmission is that after the n^{th} node is serviced you are *guaranteed* that all nodes have correctly received the message before continuing the originating process. When propagating the broadcast through the binomial tree after the root makes its last transmission there is only a *probabilistic assurance* that all other nodes have actually at that point received the message. The assurance relies on the assumption that all transmissions propagated from nodes below the root will complete in an amount of time less than or equal to the transmissions that are serviced at the root level. While this may be extremely likely in practice it cannot be absolutely guaranteed. For this reason it is common practice in codes utilizing MPI _BCAST which must be completely assured that all recipients have received the broadcast before continuing to follow the broadcast with an MPI _Barrier synchronization. This adds an extra time penalty to the overall amount of time needed to accomplish the multicast and proceed. We present a solution utilizing true level two hardware enabled multicasts followed by *reliable point to point level three acknowledgments* that provides both faster multicast messaging and guarantees receipt by all nodes prior to continuing the transmitting process. Our benchmark tests will demonstrate faster multicast messaging even when compared to the MPI _BCAST without subsequent MPI _Barrier synchronization.

3. Reliable Multicast Protocols

In order to create reliable multicast semantics from an underlying level two datagram hardware multicast there must be either an acknowledgment or a negative acknowledgment returned from each recipient of a muticast message. This is true despite the fact that the level two hardware enabled multicast is in practice extremely reliable. We must be able to **guarantee** the **correct** delivery of the multicast message to all recipients before continuing the transmitting process. In a negative acknowledgment based protocol the initiator of the multicast message receives reports back from receivers that a given message did **not** arrive. These types of systems are possible only when the receiver knows it should have received a message. These schemes are employed in the context of using multicasting to send entire files for example where the receiver can be informed of the total number of blocks that are expected to be received and the transmitted blocks are serialized. After the last block is multicast to all receivers the sender can be informed by the receivers as to

which block numbers did not arrive. The missing blocks can then be retransmitted to individual receivers as required, typically using level three reliable point-to-point transmissions. [8] Negative acknowledgment schemes are **not** applicable to systems in which the receiver can be sent an arbitrary message at any time and has no prior knowledge of when messages are to be expected or how many to expect. You can't indicate that you did not receive a message you did not know you were suppose to receive. For these types of free-form interactions positive acknowledgments of the receipt of a message must be utilized. We concentrate on this more general case and present a protocol based on positive acknowledgments of receipt.

The fundamental problem with utilizing acknowledgments to implement a reliable multicast protocol is known as the *ACK implosion*. The benefit of utilizing multicast in the first place is to reduce the number of physical transmissions that must be made on the medium. If a transmitter sends a multicast to n receivers and then is swamped by n acknowledgments we have created a communication bottleneck in the network at the transmitting node and must delay while n acknowledgments are serially processed at the transmitter. Imagine 16 nodes connected to a switch where all the nodes are trying to send acknowledgments back to a single node. This bottleneck does not exploit the switch's ability to perform multiple simultaneous transmissions between distinct pairs of nodes as all the acknowledgments are heading to the same destination. Obviously, such an approach does not scale efficiently to large numbers of nodes. The common approach utilized in distributed computing is to utilize some form of tree in which the information can be accumulated from the bottom of the tree and propagated to the top.

3.1 Cumulative Acknowledgments in a Binomial Tree

In order to solve the *ACK implosion* problem we propose a protocol in which the receivers propagate the acknowledgments back to the transmitter utilizing *cumulative acknowledgments* moving through a binomial tree. The purpose of forming the recipients into a tree is so that leaf nodes can send acknowledgments to nodes at the next level up in the tree. When an inner node in turn sends an acknowledgment further up the tree it serves as a cumulative acknowledgment representing itself and all nodes below it in the tree. This prevents the root from being swamped by acknowledgments.

3.2 Acknowledgments Propagating Up a Binomial Tree

Consider a collection of nodes that have received a multicast message. Ultimately, the root (broadcast originator) must receive acknowledgments indicating that all nodes have successfully received the message. For example, consider the collection of 16 nodes shown in the binomial tree of figure 1. This is an order 4 binomial tree. There will be 4 cycles of acknowledgment messages sent. The labels on the left side of the edges connecting the nodes indicate during which acknowledgment cycle a message is passed between nodes.

During the first cycle of acknowledgments all of the odd numbered nodes will send acknowledgments to even nodes.

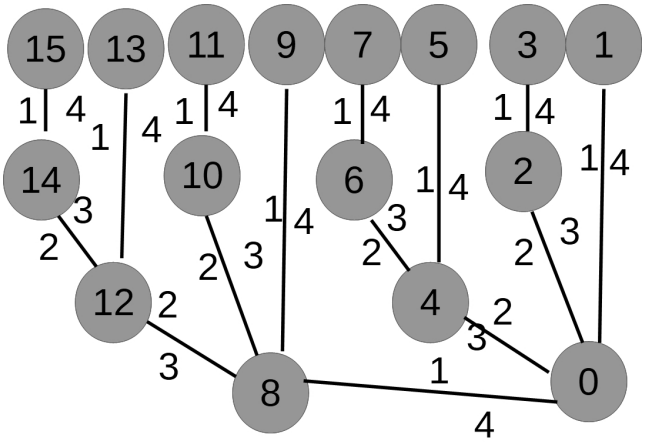

Figure 1: Order 4 Binomial Tree

This fact is leveraged in the code implementing the protocol enabling us to utilize a simple and efficient bit shift operation. In the second cycle of acknowledgments node 14 will send its acknowledgment to node 12. This serves as a cumulative acknowledgment for both nodes 15 and 14. As the acknowledgments move up the binomial tree they serve as cumulative acknowledgments for all nodes in the subtree below the node sending the acknowledgment. In the second cycle of acknowledgments note that there are a total of 4 acknowledgments transmitted. In the third cycle there are 2 and in the final fourth cycle there is only 1. Also note that the root receives and acknowledgment on *each cycle*. In general, in a binomial tree of order n there are 2^{n-i} acknowledgments transmitted during cycle i. This suggests an approach for determining what the appropriate timeout values should be for each node. This will be discussed in more detail later. It should be noted that while the multicast message in our protocol is transmitted utilizing true hardware enabled level two multicast the acknowledgments are transmitted in a point to point manner utilizing reliable level three connections. It should be made clear that the context for our investigation is multiple source single group (MSSG). We are concerned with the implementation of the underlying transport protocol utilized to deliver multicast messages on actual hardware. We are not concerned here with propagation trees for other classifications of systems such a Single Source Single Group (SSSG), Single Source Multiple Groups (SSMG), or Multiple Source Multiple Groups (MSMG). [7]

3.3 Practical Considerations Required to Implement the Protocol

3.3.1 Message Originator becomes the Root of a Binomial Tree of Acknowledgments

When processes joins the multicast group there must be some designated manager that they inform they are joining. This manager then must inform all other current members of the multicast group of the number of processes in the mulicast group and what the joining node's position is so that it can determine where in the tree it is in order to know who will send acknowledgments to them and to whom

they should send acknowledgments. How should the manager process be chosen and how do the other nodes know how to contact it? There are many well know distributed algorithms including the Ring algorithm, Bully algorithm, Gallager-Humblet-Spira, GoogleâĂŹs Chubby and Apache Zookeeper that solve the leader election problem that can be utilized for the selection of the distinguished manager node in a robust and fault tolerance fashion. [3] [2] [5] For our proof of concept implementation we have simply implemented a fixed node that the other nodes know to contact that serves as the manager to collect and distribute multicast group membership.

When a node performs a multicast the identity of that node is transmitted as meta-data with the multicast. Each transmitter then becomes the root of a binomial tree utilized for the purposes of receipt and forwarding of acknowledgments for that multicast transmission. Since all nodes have a standardized list of all the nodes in the multicast group provided to them by the manager node, each node has all the information needed to independently compute the form of the binomial tree to be utilized for acknowledgment propagation for that multicast in a canonical fashion with the originator of the multicast as the root of the binomial tree. Since each multicast transmission causes that node to become the root of a binomial tree used to propagate the acknowledgments for that multicast the communication patterns of the acknowledgments propagating up the binomial tree makes very effective use of the ability of a switch to perform simultaneous transmissions across the backplane and avoiding single destination bottlenecks. This remains true even in the event of multiple overlapping multicast transmissions originating from distinct nodes.

3.3.2 Non-power-of-two Collections of Nodes

Another practical consideration that must be dealt with results from the fact that binomial trees always contain an exact power of 2 number of nodes. As nodes incrementally join the multicast group we must be able to deal with the number of nodes in the group being any number not just powers of 2. This impacts the way that the acknowledgments propagate up the tree.

3.3.3 Pairing of Acknowledgments with Evolving Multicast Groups

Each client that wishes to join the multicast group needs to connect to manager node and register itself as multicast client. In response the manager node assigns the client a communication number and sends a complete list of currently registered multicast clients with their communication numbers, ip addresses and GUIDs. Upon receiving this response from the manager the client joins multicast group and can start listening for multicasts. Obviously, all clients can not join at once so the multicast group is built up incrementally. After every new registration request to the manager to either join the multicast or leave the multicast group the manager node sends an updated client list to all currently registered nodes. This list must contain a unique designation which will be used as metadata when multicasts are transmitted in order than the members of the multicast group can match the proper acknowledgments with the version of the multicast group that existed when a specific multicast was transmitted. All members of a given multicast group *independently* compute the corresponding binomial tree in a canonical fashion that determines who they expect to receive acknowledgments from and who they are to send them to.

As the system operates over time nodes may leave to multicast group and new nodes may join. This implies that for a given muticast the acknowledgment tree that should be used must be the one that existed at the time the multicast was made. For this reason the manager assigns a unique identifier to each version of the multicast group membership. When a mutilcast is made. That identifier is transmitted as meta-data with the payload of the muticast message. In this way all nodes know the structure of the tree that is being used for the acknowledgments associated with that multicast message. As nodes join and leave the multicast group the manager sends a message to the other nodes of the multicast group containing the unique identifier and the membership of the group along with an indication of which node you are. Nodes keep the different incarnations of the multicast group membership in order to know how to propagate acknowledgments. The older versions of the tree can be discarded after all multicasts issued under that version of the membership have been verified to have been successfully transmitted.

3.3.4 Determination of Timeout for Non-receipt of Acknowledgments

Although in practice the level two hardware enabled multicast is extremely reliable we must still account for the possibility that athe multicast for some reason did not properly arrive at some destination node. This means we must implement a timeout value and assume that if an acknowledgment has not arrived within that timeout period that the original multicast message did not arrive properly. If the acknowledgment does not arrive from a subordinate node then the node holding a properly received version of the original multicast must retransmit the message to the node that failed to respond with an acknowledgment. The question is then, how should the timeout be determined? Recall the manner in which the acknowledgments propagate up the tree during a series of cycles. For a binomial tree of order n, there will be n cycles of acknowledgments. Think of the system as pulsation n acknowledgment cycles. Nodes at the level just above a lead node know that they should receive and acknowledgment during the first cycle. Nodes at the next level up the tree know that they should receive and acknowledgment during the second cycle and so on. Thus, the level in the tree can be used as a multiplier to determine the appropriate timeout period for a given node.

There is one other situation that must be accounted for concerning timeout of acknowledgments. It is possible for an interior node for a given acknowledgment tree to receive an acknowledgment for which it has no matching multicast transmission. This salutation can occur if the lower level subtree correctly received a multicast transmission but the interior node did not. In this case the interior node receiving the unmatched acknowledgment will request a retransmission of the original multicast message contents from the node that propagated the acknowledgment. This should be accomplished via a reliable point to point level three con-

nection between the pair of nodes. The time delay to accomplish the delivery of the message contents from the lower to to the node that failed to receive the original multicast will have possible negative consequences for nodes above it in the tree resulting from the fact that this will delay when it can propagate its acknowledgment up the tree which may trigger, possibly unnecessary, retransmissions at higher levels in the tree.

4. Benchmark Test Results

Our locally available cluster used to perform the benchmark tests has 24 nodes, each with two ES-2600 12 core (24 thread) processors, for a total of 288 cores (576 threads). All nodes are connected via Gigabit Ethernet switches. The system is equipped with a 24-port Infiniband switch, however, only 8 nodes are currently equipped with Infiniband Host Channel Adapters connecting them to the Infiniband switch. This limits the range of tests we can perform using our locally available hardware. Our Infiniband switch is theoretically capable of 40 Gigabits per second. We tested the total elapsed time required to transmit and receive multicast messages using our protocol and compared it to the standard MPI broadcast of the same data. The size of all multicast messages was 2048 bytes, the MTU of the Infiniband connection. The elapsed time for our protocol includes the time for the root (the originator of the multicast) to receive all acknowledgments. Since there are no acknowledgments propagated in the MPI implementation we noted the time at which each multicast message was received at each node and then correlated that with when the message was transmitted from the originator of the MPI multicast. The difference between the time the transmitter originated the multicast and the time the last node received the multicast message was used as the transmission time for the MPI multicast. The benchmark results are shown in table 1. The first column indicates the number of nodes in the test. The second column is the percentage decrease in time required by our protocol versus the standard MPI_BCAST implementation using a binomial tree to transmit the message via point to point reliable level three transmissions. The third column is the time required by the MPI_BCAST. The rightmost column indicates the time in required to transmit the multicast using Infiniband RDMA level two multicast and receive all acknowledgments using level three point to point reliable transmissions in the binomial tree. All times are in microseconds.

# Nodes	% Speedup	MPI	Binomial Tree ACK
7	39.34	2.3919	1.4510
6	35.88	2.2550	1.4459
5	33.56	2.2098	1.4682
4	50.23	2.1962	1.0930
3	49.11	2.1158	1.0768
2	67.03	2.0298	0.6692

Table 1: Benchmarks, MPI versus Reliable Multicast with Binomial Tree Acknowledgements

5. Conclusion

The benchmark test results indicate that our implementation of a reliable multicast protocol that utilizes Infiniband hardware enabled multicast using Remote Direct Memory Access (RDMA) and cumulative acknowledgments propagated up a binomial tree to the transmitter can result in reduced latency times for multicast operations compared to a standard MPI_BCAST implementation. Initial benchmark testing indicates at least a 30% speedup using our protocol versus standard MPI implementations. This can have a dramatic effect on the overall execution time for many large-scale scientific calculations on supercomputer clusters. Additionally, our protocol provides a true guarantee that all targets of the multicast have received the transmission prior to continuing the transmitting process. Frequently found cautionary applications of MPI barrier synchronizations which follow MPI broadcasts can be safely removed from many codes given the true guarantee of receipt before continuation. The implementation and benchmarking test codes can be obtained by sending an email to the author as shown in the heading of this paper with a subject line of *Reliable Multicast over Infiniband*.

6. Futute Work

The current implementation establishes a point to point reliable level three connection between nodes for the purpose of sending acknowledgments in the binomial trees. It then destroys this level three connection after transmitting the acknowledgment. This means that when a subsequent acknowledgment is needed to be send between those same two nodes and entirely new level three connection is once again created and destroyed to send that acknowledgment. It would obviously be more efficient to retain the level three connection between that pair of nodes and to simply reuse it when needed in the future. This would improve the benchmark test results over those shown resulting in an even better improvement versus the standard MPI implementation. The inefficient destruction and recreation of the level three connections in the current proof of concept implementation is simply a side-effect if the way the meta-data contained in the acknowledgment is managed and the lack of code in the current system to build the data structures needed to manage the list of retained point to point connections. We plan to augment the current implementation with the logic needed to take advantage of retaining the level three point to point pairwise connections utilized to transmit acknowledgments. This should result in a noticeably improved performance over the current implementation when many multicasts are performed that result in the same pair of nodes needing to send acknowledgments.

During our benchmark testing it was never the case that a level two RDMA multicast was not properly received and acknowledged by all recipients. Thus, we have not indicated

what the timing delay impact might be in the presence of intermittent failures. In order to obtain test results for the behavior of the protocol when lost transmissions occur we will need to inject artificially generated failures by having nodes programed to purposefully discard acknowledgments at a specified rate.

The local cluster we utilized for our benchmarking has a limited number of nodes connected via Infiniband switching. We plan to obtain access to a larger system to perform benchmark testing with larger problem sizes.

We did our first implementation and testing of our protocol utilizing Infiniband hardware to determine if it would provide a speedup on high-end computational clusters. The protocol can be implemented over the more ubiquitous Ethernet switch based systems. We plan to created an Ethernet hardware assisted multicast implementation for our local cluster and ultimately a hybrid implementation that can utilized both.

Finally, it would be instructive to take a standard distributed scientific calculation that utilizes MPI broadcast heavily and modify it to utilize our multicast protocol. Then execute a series of benchmark tests on our cluster to investigate what speedups we can produce for actual production applications.

7. References

[1] Greg Burns, Raja Daoud, and James Vaigl. Lam: An open cluster environment for mpi. In *Proceedings of Supercomputing Symposium, 379–386*, 1994.

[2] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[3] Wan Fokkink. *Distributed Algorithms: An Intuitive Approach*. The MIT Press, 1st edition, December 2013.

[4] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the mpi message passing interface standard, booktitle =.

[5] IBM-Analytics. https://www.ibm.com/analytics/hadoop/zookeeper.

[6] Software in-the Public-Interest. http://www.open-mpi.org/.

[7] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, reissue edition edition, March 2011.

[8] Qian Liu and Robert D. Russell. IBRMP: A reliable multicast protocol for infiniband. In *IEEE 22nd Annual Symposium on High-Performance Interconnects*, 2014.

[9] Mesh-Technologies. http://www.meshtechnologies.com.

[10] Peter Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, San Francisco, CA, 1st edition, 1996.

[11] SCALI. http://www.scali.com/.

[12] Softtech. http://www.mpi-softtech.com/.