Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp104-109)

Broadcasting Algorithm of Constant Complexity for Fully-Switched Clusters

SÁNDOR JUHÁSZ
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
1111 Goldmann Gy. ter 3., Budapest
HUNGARY

Abstract: - Clusters are high performance computation systems, built up out of standard off-the-self workstations connected with fast, but standard communication devices. This structure allows higher pure processing power and lower hardware costs compared to other supercomputers. One main disadvantage of clusters is the lower communication throughput between the processing elements, as standard methods usually provide weaker performance than the much more expensive special communication devices of supercomputers. Because of this it is very important to take the most advantage of the existing communication potential in cluster environment. This paper presents a method of enhancing the performance of the broadcast group communication primitive by using a new algorithm that takes advantage of message decomposition and asynchronous communication. When used in fully switched cluster environment the new solution provides a constant execution time independent of the number of participants. Test measurements show that the algorithm follows well the predicted behavior, and has superior performance, compared to the widely used binomial tree method used in standard message passing libraries. As broadcasting is a building block of various group communication primitives, improving its performance may have beneficial effect on several routine of message passing libraries.

Key-Words: - Group Communication, Broadcasting Performance, Fully Switched Clusters, Constant Time Complexity

1 Introduction

Clusters are high performance computational systems being built up out of standard PCs or workstations that are connected via high performance communication networks. Because of their high computation potential, low hardware costs, simple fault tolerance and good scalability clusters play an increasingly important role on the high performance computer market. As their standard communication devices usually provide a lower throughput than the expensive special designs used in supercomputer environments, in clusters the performance of the internode communication is a primary issue. This bottleneck often limits the performance of this architecture and hinders the efficient implementation of communication intensive algorithms.

These facts stimulated significant research effort on the hardware and software aspects of cluster communication. Thanks to hardware improvements, cluster systems can take benefit of new communication standards (Gigabit Ethernet, Myrinet, SCI, Quadrics, InfiniBand), providing high performance (more Gbps) and low latency (< 10 μs). Clusters systems usually build on a fully-switched network topology, reducing competition for physical bandwidth and allowing a collision-free environment for communication.

The communications performance is limited by the physical properties of the underlying network, but previous studies [1][2] concluded that different software overheads also have a significant impact on the performance of parallel applications. The distributed memory programming used in clusters is usually supported by various message passing libraries such as PVM or MPI. Due to the different inefficiencies and overheads at the levels of application, message passing subsystem, and operating systems, the physical transfer time itself – especially in case of smaller messages– is only a fraction of the total application-level delay.

This paper focuses on improving the performance on the level of group communication routines of message passing libraries, where broadcasting plays an emphasized role, because it is widely used in itself, and also is a building block of other communication primitives (*all-gather, all-to-all, all-reduce*). This paper presents a method for enhancing the performance of broadcasting by software means. We introduce a new algorithm using message decomposition and asynchronous communication, which has an execution time complexity of $O(1)$ achieved before only by the help of hardware support.

The rest of the paper is organized as follows: Section 2 introduces and compares the commonly used methods –both with and without hardware

support– for implementing the *broadcast* primitive in cluster environments. Section 3 details our symmetric algorithms providing a new approach of data distribution in fully-switched cluster systems. Section 4 compares the performance of the classical tree and the new symmetric algorithm, and verifies the correspondence of the measured curves and those performance predicted by the theory. The paper concludes with summarizing the results and showing their application possibilities, in Section 5.

2 Common Methods of Broadcasting

Following the recommendations of the MPI standard [3] most communication subsystems implement the group communication primitives based on the point-to-point transfer functions. Although this technique might not be the most efficient way, it certainly allows a fast and portable implementation of the group primitives. The efficiency of the different implementations is strongly influenced by the topology of the underlying connection network. As today the virtual crossbar (fully switched) topology is the overall dominant way of connecting cluster nodes, this paper only considers the reasonable implementations in such an environment. All execution time estimations use the widely accepted [4][5] linear model, where the communication time t_c equals to

$$t_c(n) = t_0 + nt_d \,, \qquad (1)$$

where n is message size, t_0 is the initial latency, and t_d is the time needed to transfer one data unit (reciprocal of the effective bandwidth).

The simplest method of broadcasting is the linear one when the data transfer is controlled from a single source. This technique is simple and easy to implement, but not very efficient: it has a linear increase of execution time as the number p of destination nodes grows:

$$t_c(n,p) = p(t_0 + nt_d) \Rightarrow O(p) \,, \qquad (2)$$

Because the source node plays a central role as a single sender, this algorithm reduces collisions on a shared medium, that is why it was preferred in the early (middle of the '90-es) implementations of communication libraries (LAM/MPI [7]).

The linear complexity of the broadcasting can be reduced by taking advantage of distributed implementation. The most general way is to parallelize the control of the communication using a binomial tree topology. In this case as the originator forwards its data to other nodes, those later will also act as secondary sources increasing the number of senders in each step. This achieves an execution time of $O(\log_2 n)$ complexity.

Compared to the linear implementation the main advantages are the reduced complexity, and better load balancing. This implementation scales better, and is used in most current MPI implementations (e.g. MagPIe [8] or MPICH [9]).

With additional implementation efforts, and by sacrificing the portability, environment specific protocol stacks (such as GAMMA [2]) can be developed to further enhance the network throughput. Different methods can take advantage of hardware multicast or broadcast support. Here the originator has to send the data only once, and the hardware layer takes care of the rest In theory this results in a complexity of $O(1)$, thus these solutions provide a high performance with perfect scalability. This way each receiver node gets exactly the same messages, implying that the problems of reliability, handling large messages, and forming arbitrary groups must be solved by the developers at the software level.

The problem of reliability is a well-treated topic [10][11], and is usually solved by a kind of acknowledgment mechanism. As all the reply messages are sent back to a single originator, their processing may be costly on a large scale (ACK flooding [11]). This problem can be alleviated using lazy-acknowledgment protocols and multilevel ACK collection [10]. Acknowledgment systems may also take care of large messages, which must be broken down into smaller pieces that can be handled by the lower network layers. In this case, the originator always has to be aware of the amount of empty buffer space in all the destination nodes, and the whole broadcasting can advance only at the pace of the slowest partner. Although the desired O(1) complexity is not always reached in the practice, the hardware support still offers the fastest and most scalable solution for broadcasting. Unfortunately, the price is paid by the higher implementation efforts and the total loss of portability. The properties of the above mentioned methods are summarized in Table 1. For the sake of completeness the new symmetric algorithm to be introduced in the next section is also included.

3 Symmetric Asynchronous Broadcast

Although most message passing libraries allow using asynchronous communication, their group primitives are synchronous. This way the total time of communication is the sum of the consecutive message sending steps as described in Equations (1) and (2). However previous studies [1][2][12] showed, that losses from the software inefficiencies can be reduced by allowing various communication

Table 1. Comparison of different methods of broadcasting

Method of broadcasting	Flexibility			Performance		Scalability
	complexity of implementation	portability (op. system, networks)	implementing reliability	min. delay at the source	execution time	
Linear	simple	simple	simple	n messages	*O(n)*	bad
Binary Tree	medium	medium	simple	2 messages	$O(log_2 n)$	medium
Hardware sup.	complex	none	complex	1 message	*O(1)*	exellent
Symmetrical	medium	medium	simple	1 message	*O(1)*	limited

steps to overlap with the computation and as well as with each other. Asynchronism helps the parallel messages to make better use of the physical bandwidth, so it can improve the performance of the traditional methods by hiding the effect of the initial setup time (t_0).

To achieve better results we combine asynchronous communication with message decomposition. The latter increases the parallelism, while asynchronous communication eliminates the effect of the additional initial t0 latencies introduced by the greater number of smaller messages. The proposed symmetric algorithm consists of a complex scheme divided into two overlapping phases presented separately in Fig 1. In the first phase the source node sends a different part of the original message to each destination node, which all act as secondary sources in the second phase. With a number p of the destination nodes the algorithm works as follows:

Phase 1. The source node cuts the message of size n to be broadcasted into p pieces. To avoid rounding errors, piece i is formed as

$$address = \left\lfloor \frac{(i-1)n}{p} \right\rfloor \quad length = \left\lfloor \frac{i*n}{p} \right\rfloor - \left\lfloor \frac{(i-1)n}{p} \right\rfloor, \qquad (3)$$

where the notation $\lfloor x \rfloor$ gives the largest integer less than or equal to x. The message fragments are completed with administrative information in order to allow the final reconstruction at the target nodes. The source node forwards piece i to destination node i, but does all transfers are done in parallel thereby eliminating the effect of the additional t_0 latencies. Thus the total time cost of phase 1 is

$$t_c(n,p) = t_0 + p\frac{nt_d}{p} = t_0 + nt_d \qquad (4)$$

Phase 2. When receiving first a message part, destination node j allocates a memory space for the whole message of size n, and copies the part just arrived to its final place. The further arriving message parts are copied to the relevant locations in the already existing buffer. When an incoming message part comes from the original source (was created in phase 1), then current destination node is responsible for forwarding this fragment to each of the remaining *p-1* destination nodes. While distributing this message fragment to the partners, the node is still able to receive other message fragments due to the asynchronous communication.

The algorithm is finished when all the message fragments have arrived at all the destination nodes.

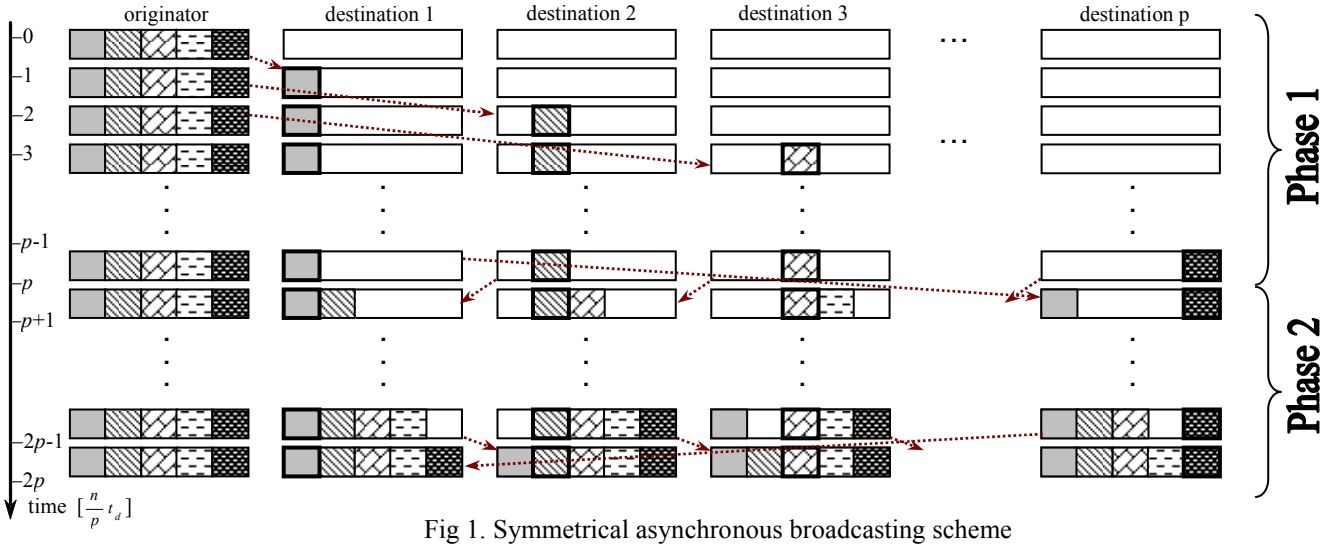

Fig 1. Symmetrical asynchronous broadcasting scheme

It is important to note, that this algorithm does not

make any assumption on the order of the arriving message parts. It is possible that the destination nodes receives some of the message parts from the second phase getting their part to distribute from the source node.

Considering a cluster system where the nodes are connected through a switching hub in full-duplex mode, the ideal execution time of the algorithm is only limited by the physical bandwidth of the links. The critical (slowest) path of execution is the distribution of the last fragment, and this piece leaves the source node according to Equation (4). The secondary distributor node must forward this message of length *n/p* to the remaining *p-1* nodes, resulting in a total execution time of

$$t_c(n,p) = t_0 + nt_d + t_0 + \frac{(p-1)nt_d}{p} = 2t_0 + nt_d(2 - \frac{1}{p}) \quad . (5)$$

The execution time shown in Equation (5) has an asymptotic complexity of *O(1)* for arbitrary values of t_0 and t_d parameters. Because of its symmetry the algorithm is perfectly scalable in theory, and provides significantly better execution times than the widely used tree algorithm. Although the complexity is equal to that of the hardware aided solution, it is visible, that the broadcasting takes the time of two full message transfers, which is double of the expected time with hardware support.

Despite of its elegance, the algorithm suffers from some drawbacks limiting its usability. The algorithm is not efficient for very short messages, because small packets carry a larger relative overhead of the network protocols, and the symmetric algorithm forces the generation of great number ($\sim p^2$) of *n/p* sized fragments. In case of small message sizes it is better to distribute the whole information in a single step during phase 1. A second problem is the validity of the presumption that all the nodes are able to send and receive at the full speed of their link capacity. In practice the current computers can easily cope with the full-duplex bandwidth of the network adapter, but the switching hub can prove to be a bottleneck. For the algorithm to scale perfectly the switching hub must be able to provide full speed at all of its ports at the same time. Although the amount of the incoming and outgoing data is evenly distributed in time, the switching hub has to handle the competition of more nodes sending data packets to the same destination. As the scalability of the algorithm is limited by the saturation point of the active network equipments, the efficient usability of the new algorithm is practically limited to small cluster environments (a few tens of nodes).

4 Performance Measurements

This section demonstrates the practical merits of the newly introduced asynchronous algorithm by comparing its execution time to that of the widely used binary tree method. To preserve the fairness of the comparison the same hardware and software environment were used during the measures. The testbed used for the validation measures was built up out of 15 uniform PCs having an Intel Pentium IV processor of 2.26 GHz, 256 MB RAM, and an Intel 82801DB PRO/100 VE network adapter of 100 Mbits. The nodes were connected through a 3Com SuperStack 4226T switching hub. All the nodes were running Windows XP operating system. An implementation of MPICH (NT-MPICH v1.3.0 [9]) was used to implement and test both the proposed and the common tree-based algorithms.

The right choice of the measurement domain is an important question. By studying the communication patterns of real-world, MPI based parallel applications Vetter and Mueller found in [13], that message sizes in group communication is relatively small. Another study [14] examined the message size distribution of the NAS Parallel Benchmark suite [15]. The NAS Benchmarks include multiple kernels working on datasets of five different sizes to model a wide palette of application types. This study found that half of all the messages are smaller than 10 kB, and only a fraction is greater than 1 MB. As a result we have chosen to handle message sizes between 2 bytes and 512 kB with logarithmic steps.

Two kinds of scenarios were considered. In the first case, the own broadcast primitive of the MPICH library was tested, which is based on tree topology. In the second case the asynchronous algorithm was tested implemented using the asynchronous message transfer primitives (MPI_Irecv, MPI_Isend). In order to allow measuring the communication time on the source node each destination node sends a short reply message when all the broadcast data has successfully arrived. In both cases, the total communication time spans from the invocation of the broadcast primitive to the arrival of the last acknowledgement. To equalize the variations in execution times due to any reason 10 separate measurements were made in each point, and their arithmetical mean is considered to be the result.

The complexity is more apparent on a linear scale, but the domain of message sizes span over multiple orders of magnitude, thus Fig. 2 shows the same execution time results in both linear and logarithmic scale. It is clearly visible that the symmetric

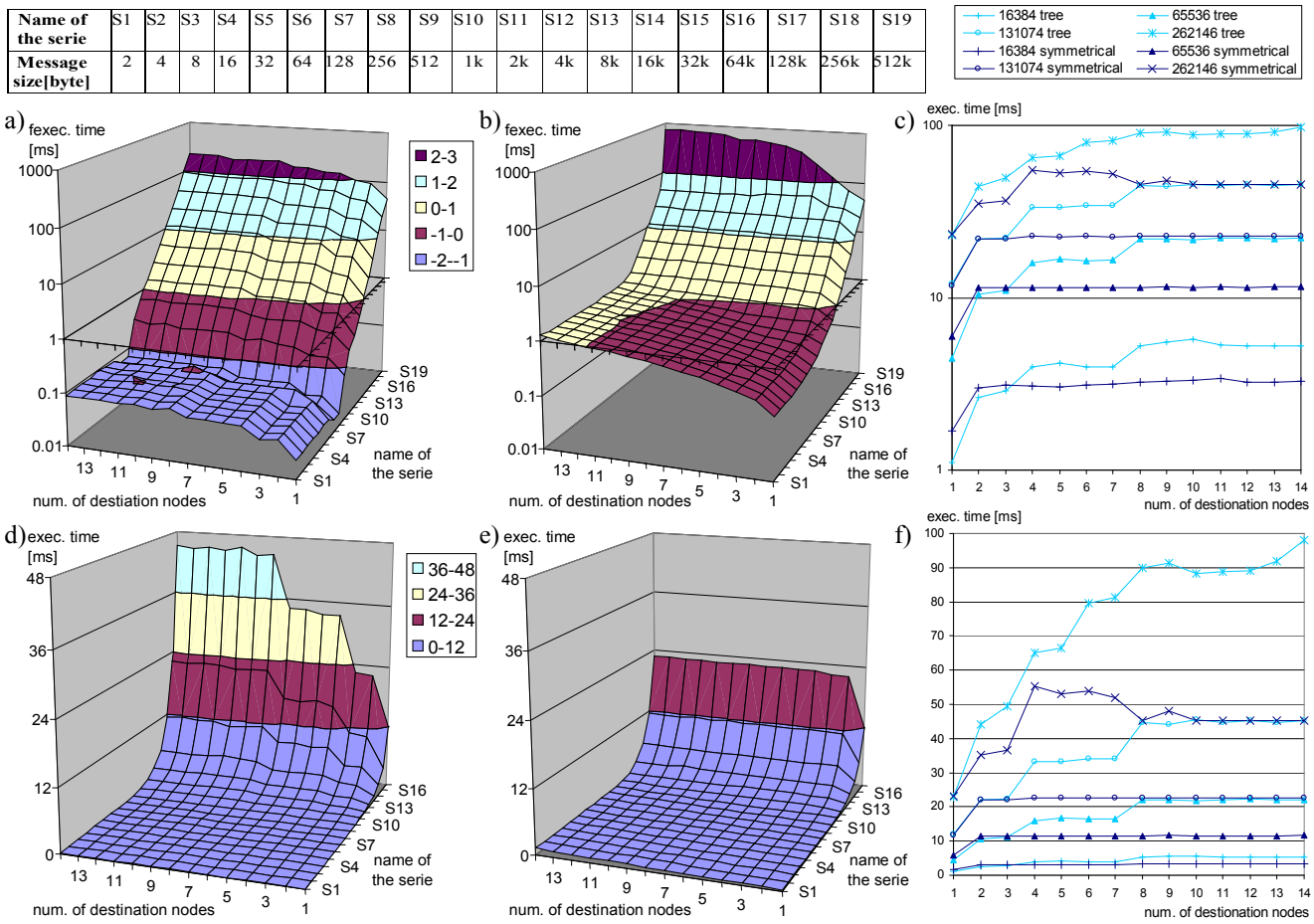

Fig. 2. Execution times of the tree (a,d) and of the symmetric (b,e) algorithms, and comparison in the same chart (c,f). All figures are represented using both logarithmic (a,b,c) and linear (d,e,f) scales

algorithm is also able to keep its constant complexity in the practice for a large domain of message sizes, although for small and large messages the execution time grows as the number of nodes increases. For small messages this growth is due to the overhead of message fragmentation (the size of the different protocol envelopes is comparable with the message size), and in case of large messages the growth is caused by the saturation effect of the network switch.

The performance comparison of the tree and the symmetric algorithms, in Fig. 2, includes two charts (c,f) presenting the execution times coming from the two types of algorithm in the same diagram. For a single destination node the two algorithms are equivalent, as both have to send all the data in one piece to the only destination node. The tree algorithm shows a logarithmic increase and As expected from Equations (5) the symmetric one exhibits a hyperbolic behavior, as the number of nodes grows. For small messages (≤ 2 kB) the tree algorithm is slightly better, but for the remaining part of domain the performance of the symmetric algorithm is always superior to that of the tree approach. When broadcasting to more than 9 nodes the symmetric algorithm proves to be even twice as fast as the traditional tree method.

5 Conclusion

The communication subsystem of the cluster often embodies a weak point of performance, and numerous efforts have been made for its improvement. This paper has addressed one of these issues in the domain of group communication primitives. The aim was to improve the speed of the broadcast primitive without changing the network infrastructure and the used protocols. The proposed new algorithm significantly differs from the traditional methods. It has the advantage of being portable and easy to implement but also benefits from a execution time complexity of *O(1)*, achieved only with hardware support so far. The presented algorithm builds on asynchronous, reliable point-to-point communication operations and uses message decomposition. The symmetry in its communication pattern allows good scalability and automatic load balancing.

The algorithm is intended to work in the very common, fully-switched cluster environment. As their ideal performance is achieved by maintaining

continuous communication between each pair of nodes, the performance strongly relies on the quality of the network switch. The saturation point of the switching hub puts a practical limit on the scalability perfect in theory, thus the algorithm can be used efficiently in a small cluster environment (a few tens of nodes) only. As the number of generated messages grows with the square of the number of the destination nodes, more messages are produced than those by the traditional methods. More messages generate more overhead, causing the algorithm to perform worse for very small message sizes.

To demonstrate the usability of the new method, its performance was compared to the traditional binomial tree implementation of the broadcast primitive. During the tests the tree version was outperformed significantly by the new symmetric method for message sizes greater than 2 kB. For large messages and more than 9 destination nodes the new algorithm had a double performance compared to the built-in communication primitive.

As broadcasting is an important building block of other message passing primitives, the results presented in this paper can be used directly for improving the performance of group communication in message passing libraries for cluster environments. Efficiency and easily predictable behavior also help to increase and tune the performance of the distributed algorithms.

Acknowledgments

The fund of “Mobile Innovation Centre” has supported in part, the activities described in this paper. Their help is kindly acknowledged.

References:

[1] R. Martin, A. Vahdat, D. Culler, and T. Anderson, “Effects of Communication Latency, Overhead and Bandwidth in a Cluster Architecture”, 24th Annual International Symposium on Computer Architecture, Denver, 1997, pp. 85–97.

[2] G. Chiola, G. Ciaccio, “Efficient Parallel Processing on Low-Cost Clusters with GAMMA Active Ports”, Parallel Computing 26, Elsevier Science, 2000, pp. 333–354.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, “MPI–The Complete Reference”, Volume 1 - The MPI-1 Core, 2nd edition. The MIT Press, 1998.

[4] U. Meyer et al., “Algorithms for memory hierarchies”, LNCS 2625, Springer-Verlag, Berlin, 2003, pp. 320–354.

[5] J.W. Baugh Jr., R.K.S. Konduri, “Discrete element modeling on a cluster of workstations”, Engineering with Computers 17, Springer-Verlag, London, 2001, pp. 1–15.

[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, “A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard”, Parallel Computing, 22(6), 1996. pp. 789–828.

[7] Indiana University, Indiana University's Open Systems Lab, “LAM/MPI”, http://www.lam-mpi.org/

[8] T. Kielmann, Rutger F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, “MagPIe: MPI's collective communication operations for clustered wide area systems”, ACM SIGPLAN Notices, 34(8), 1999. pp. 131–140.

[9] RWTH Aachen, Lehrstuhl für Betriebssysteme: *Multi-Platform MPICH*. http://www.lfbs.rwth-aachen.de/mp-mpich/

[10] S. Pingali, D. Towsley, and J. F. Kurose, “A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols”, Sigmetrics Conference on Measurement and Computer Systems, ACM Press, New York, NY, USA, 1994. pp. 221–230.

[11] D. Buntinas, D. K. Panda, and R. Brightwell, “Application-Bypass Broadcast in MPICH over GM”, International Symposium on Cluster Computing and the Grid (CCGRID ’03), May 2003.

[12] S. Juhász, H. Charaf, “Exploiting Fast Ethernet Performance in Multiplatform Cluster Environment”, 19th Annual ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004. pp. 1407-1411.

[13] J. S. Vetter, F. Mueller, “Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures”, IPDPS, April 2002.

[14] M. Lobosco, V. S. Costa, and C.L. de Amorim, “Performance Evaluation of Fast Ethernet, Giganet and Myrinet on a Cluster”, International Conference on Computational Science 2002, The Netherlands, 2002. pp. 296–305.

[15] D.H. Bailey, J.T. Barton, T.A. Lasinski, and H.D. Simon, “The NAS Parallel Benchmarks”, Tech. Report NASA memorandum 103863, NASA Ames Research Center, USA, July, 1993.