The Impact of Network Architecture in Cluster Parallel Algorithms Design: Matrix Multiplication on Infiniband

Gustavo Wolfmann* Fernando G. Tinetti**

Lab. de Computación - Universidad Nacional de Córdoba
Av. Velez Sarsfield 1611, 5000, Córdoba, Argentina, gwolfmann@gmail.com

III-LIDI - Universidad Nacional de La Plata
50 y 120, 1900, La Plata, Argentina, fernando@info.unlp.edu.ar

Abstract. Ethernet has been a standard technology used for cluster interconnection, which is based on a shared bus. This technology impacts in some way the kind of messages used for parallel algorithms optimization on clusters: point to point messages are only used when necessary since collectives communications (broadcasts, more specifically) are more efficient. The emergence of Infiniband as network technology for interconnecting nodes provides better bandwidth and, also, changes the topology of the network: it is based on serial point to point links connecting nodes. Thus, algorithms can use point to point messages without penalizing efficiency. Furthermore, point to point links suggest changes in the scheduling of the parallel tasks for parallel performance optimization. This paper shows of experiments measuring two matrix multiplication algorithms, one using broadcast communications and another using point to point communications, both running over the two kind of networks and the impact on the scheduling policies.

1 Introduction

Interconnection network technology has been focused on performance since many years ago, and has introduced strong changes compared to the standard Ethernet interconnection network. In some way, Ethernet has survived in the clusters high performance arena due to low cost and extensively used layer 2 switching devices. Infiniband is one of the relatively new interconnection networks, on which the basic hardware architecture is serial point to point instead of the classical Ethernet shared bus [12]. Given the focus on optimization in high performance computing, the underlying interconnection network necessarily has some impact on parallel algorithms design. More specifically, clusters are made up of high performance desktop computers interconnected by a high performance local area network and parallel algorithms tend to take advantage of available computing power as well

* Becario Universidad Nacional de Córdoba
** Investigador Comisión de Investigaciones Científicas de la Provincia de Buenos Aires

as avoiding performance penalties of communication architecture. And parallel algorithm design is influenced by both of this performance oriented guidelines.

Ethernet interconnection networks are currently widely spread out on clusters used for parallel computing. This interconnection technology in some way induces the broadcast communications, since it is expected a priori that broadcast communications have the best implementation (from the point of view of performance). On another hand, Infiniband necessarily induces point to point communications, since the interconnection technology is based on point to point physical links. Also, broadcast communications are expected to be penalized on Infiniband from the point of view of performance. This paper is focused on analyzing the parallel algorithms efficiency on clusters, where parallel algorithms can be based either on broadcast or point to point communications. Two clusters are used for relative performance comparison: an Ethernet cluster and an Infiniband cluster. MPI (Message Passing Interface) is used for algorithm implementation, given its wide acceptance in the parallel high performance community on distributed memory parallel architectures [13]. The matrix multiplication problem will be used as representative of the linear algebra applications, since it has strong and well defined computing requirements and, also, provides a fair and well known problem for performance comparison [14].

In general, algorithms performance will be analyzed from the point of view of the communication as well as computing patterns. Broadcast communications lead to a pattern where every process computes on local/owned data and "shared" or "common" data (sent/received via a broadcast message). On point to point based algorithms there is no idea of shared or common data, since each process computes on local data in the sense that is not shared with any other process at computing time, besides, it can be possibly received from a predefined and well known set of neighbors. The "wavefront pattern" [3], is a parallel programming pattern where data is disposed in a logical plane or space, and the computations start at an edge and advance in diagonal to the opposite edge like a *wave* and is the theoretical context used for the point to point algorithm implementation. The hypothesis in this paper is such that the processing sequence has to be taken into account with point to point communications. Also, the processing sequence will be defined in terms of application specific data dependence as well as computing dependency defined by the communication pattern, i.e. the data arriving from the process neighbors.

The rest of the paper is organized as follows: cluster configuration and algorithms are presented in the next section. The section 3 expose the results of the experiments. Related work and conclusions are the last sections.

2 Clusters and Algorithms

Cluster hardware presented in this section is relatively well known in general, and can be considered standard in the context of high performance computing. The cluster specification is given here just to provide the context on which the algorithms will be analyzed. However, the algorithms are not dependent on the

cluster hardware, they are just focused on performance optimization on each kind of clusters interconnection network, using either broadcast or point to point communications.

2.1 Experiments Environment

The cluster used for the experiments is made up of four SMP (Symmetric Multiprocessing Systems), two of them are dual processor with Intel Xeon 5420 quadcore processors and eight gigabytes RAM, and the remaining two are dual processor with AMD Opteron 2200 dual-core processors and four gigabytes RAM. Two networks interconnect the nodes: 1 Gb/s Ethernet and 20 Gb/s Infiniband (Flextronics switch and Mellanox MHGS18-XTC 4x interfaces).

All computers in the cluster run the Centos 5.3 operating system. Infiniband library installed is Mellanox Open Fabric Enterprise Distribution 1.4 for linux [15], and Sun's cluster tools 8.1 [16] provides the MPI implementation. Programs are implemented in Fortran and compiled with Sun Studio Express [17], using Sun Sunperf library, which includes the BLAS (Basic Linear Algebra Subroutines) implementation from which the optimized matrix multiplication is used. OpenMP directives are used to take advantage of multiprocessing on each computer. More specifically, four threads are used on every computer in order to use the four cores available in the AMD based computers and four of the eight available in the Intel based computers. In this paper, as a first approach, the cluster will be used as a homogeneous cluster, i.e. as a cluster with four dual processor cluster, in which each processor is dual core, thus obtaining a total of 16 cores in the cluster. Issues such as heterogeneous computing power and workload balance will be out of the scope of this paper.

2.2 Algorithm Based on Broadcast Communications

A broadcast based parallel matrix multiplication algorithm was used in previous works [1] obtaining a speedup near the optimum one on clusters of multicore nodes. We have tested that multicore power is well exploited by the BLAS libraries used (Sun's sunperfect library), by only setting the number of threads, thus the accent is put on the distribution part of the algorithm. The algorithm is schematically described in Fig. 1 for four computers processing $C = A \times B$, where matrices A and C are distributed by row blocks and matrix B is distributed by column blocks. Thus, every node in the cluster has a subset of A and C row blocks, and a subset of B column blocks as shown in Fig. 1-a). There are as many A, B, and C blocks as computers in the cluster. The algorithm follows a well defined sequence of B block broadcasts, Bi, and C partial computing on each node as shown in Fig. 1-b).

2.3 Algorithm Based on Point to Point Communications

Data distribution is similar to the previous one given that matrices A and C are also distributed by row blocks and B is distributed by column blocks. However,

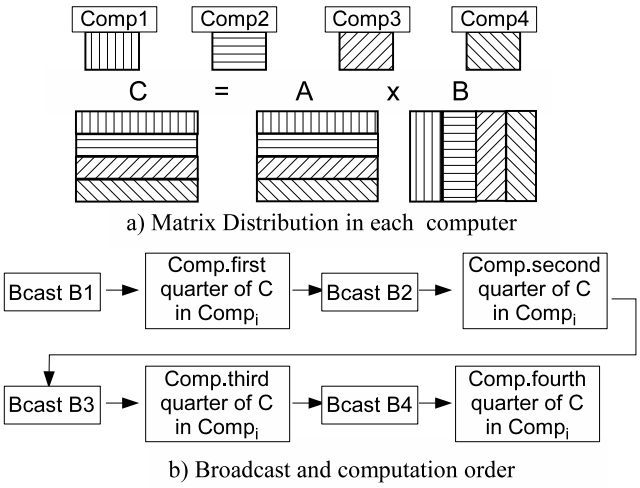

Fig. 1. Broadcast Based Matrix Multiplication Algorithm.

it is expected that the number of blocks per process is more than one, and block cyclic distribution is used as shown in Fig. 2-a) for 4 processes and 2 blocks per process. The number of blocks per process is a performance determined parameter, the *natural* starting value would be 2 (i.e. two blocks per process). Computing is determined in terms of the square blocks computed from each A_i and B_j blocks, i.e. $C_{i,j} = A_i * B_j$ on each process, which has only the A_i blocks and needs to receive most of the B_i blocks from other process. Blocks $C_{i,j}$ are computed in "waves" as shown in Fig. 2-b) which defines a sequence of computing and, also, the corresponding send-receive messages, as defined in the *wavefront* pattern [2], where the computing advances from the upper left corner to the lower right corner according the pattern data distribution.

The sequence shown in Fig. 2-b) is given in terms of wave, from 0 to 14 for A and C matrices divided in 8 row blocks and B matrix divided in 8 column blocks, which determines the 8×8 $C_{i,j}$ blocks shown in the figure. Headings in Fig. 2-b) show the processes containing the data needed for computing each $C_{i,j}$ block, and blocks in the same wave can be computed in parallel if they belong to different processes. There are 4 blocks being computed in parallel in most of the waves, i.e. in those waves containing 4 or more blocks. In the first wave (wave 0), $C_{0,0}$ is computed in process 0 with local data. In the next step (wave 1) processes 0 and 1 compute $C_{0,1}$ and $C_{1,0}$ respectively. Block B_1 has to be sent from process 1 to process 0 and block B_0 from process 0 to process 1. Next computing waves are similar, every processor related to the wave: a) send and receive B blocks involved in the corresponding $C_{i,j}$, and b) the $C_{i,j}$ are effectively computed in parallel.

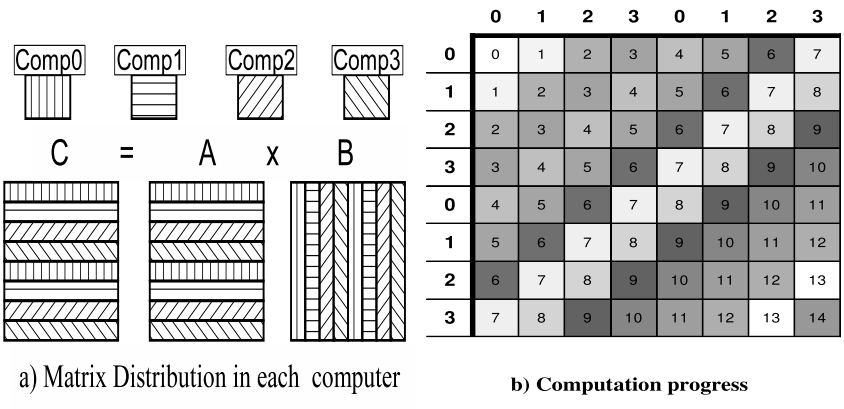

Fig. 2. Wavefront Parallel Pattern

3 Experiments and Results

In the implementation of the above wavefront pattern, non-blocking messages were used in the sequence *send-receive-wait until received-compute*, allowing to ovelap computing and communications. Table 1 shows the execution time for the broadcast and point to point algorithms, over Ethernet and Infiniband.

Table 1. Experimental Results Over Ethernet and Infiniband

Matrix Size	PtP Eth (secs.)	Bcast Eth (secs.)	PtP/Bcast Eth	PtP Inf (secs.)	Bcast Inf (secs.)	PtP/Bcast Inf
2000	0.886	0.610	1.452	0.691	0.628	1.100
4000	3.090	2.310	1.338	2.483	2.022	1.223
8000	17.204	12.647	1.360	14.378	11.271	1.276
12000	56.186	39.219	1.433	45.145	36.088	1.251

The point to point algorithm implies a performance penalty from 30% to 45% over the broadcast algorithm when the computers are interconnected by Ethernet. When the computers are interconnected by Infiniband, the performance penalty for the point to point algorithm is about 25%.

To see graphically the behavior of the messages in both algorithms, we use a tool for profiling and tracing MPI applications, Sun's analizer [18], that shows the trace of MPI messages. Fig. 3 shows a graphical sample for the broadcast algorithm and Fig. 4 shows a graphical sample for the point to point algorithm. Each row in the figures represent one of the four MPI processes, white (lighter) areas represent computing times of the application and light blue (darker) areas represent the time of the MPI routines. In both cases, the first two rows corre-

spond to Xeon based computers and the last two to Opteron based computers. The timing behavior is about the same in all the matrix sizes experimented.

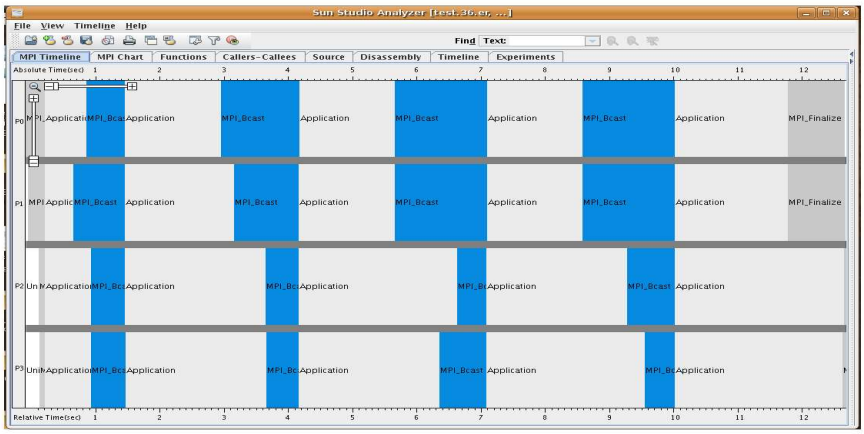

Fig. 3. MPI Timing of Broadcast Algorithm Over Ethernet

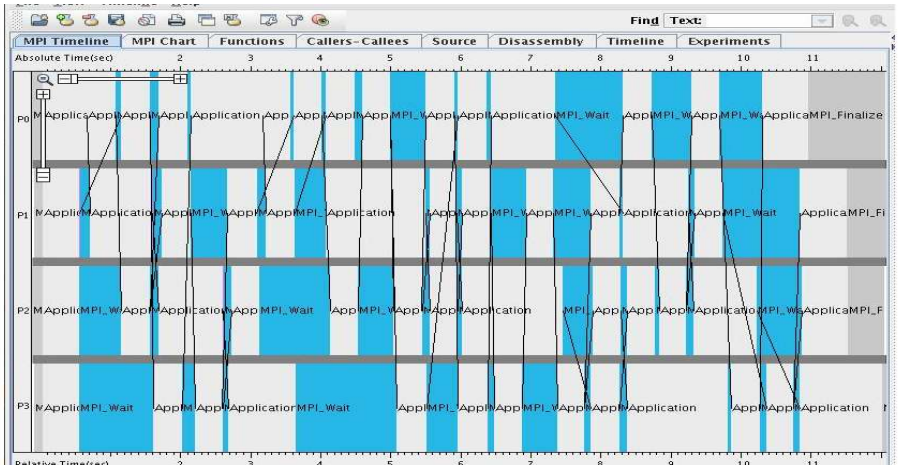

Fig. 4. MPI Timing of Point to Point Wavefront Algorithm Over Infiniband

In the first case, Fig. 3, it can be seen that the MPI broadcast routine takes long times in waiting the message finalization and synchronizing all processes. This fact is one of the motivations of this work: to make a better use of the communication time. The wavefront pattern implementation is shown in Fig. 4,

with many more but small light blue areas representing the communication time. The black lines between processes in Fig. 4 represent synchronizations. All the messages in the experiment are the same size, so the difference in the time among two messages, beyond a logical small difference are due to synchronization.

The wavefront parallel pattern acts like a pipeline [2] in the sense that both share the concepts of filling up and emptying the pipe. This behavior is exposed in the figure 4, but there are some other delay in the middle of the processing. What is the origin of these delays? The answer can be found by analyzing the sequence of the $C_{i,j}$ computing in each process.

Table 2 presents by column the message calls (send/receive) done by process. Each number in the body represents the processor that send / receive the message to / from the processor at the heading column, and the subindex represents the wave number in the wavefront sequence according to Fig. 2-b). For wave 0,

Table 2. Beginning of the sequence of calls in each processor

Processor 0		**Processor 1**		**Processor 2**		**Processor 3**	
Sender	Receiver	Sender	Receiver	Sender	Receiver	Sender	Receiver
0_0	0_0	–	–	–	–	–	–
1_1	1_1	0_1	0_1	–	–	–	–
2_2	2_2	1_2	1_2	0_2	0_2	–	–
3_3	3_3	2_3	2_3	1_3	1_3	0_3	0_3
$0_4^{(1)}$	$0_4^{(1)}$	3_4	3_4	2_4	2_4	1_4	1_4
$0_4^{(2)}$	$0_4^{(2)}$	–	–	–	–	–	–
$1_5^{(3)}$	$1_5^{(4)}$	$0_5^{(4)}$	$0_5^{(3)}$	3_5	3_5	2_5	2_5
$1_5^{(5)}$	$1_5^{(6)}$	$0_5^{(6)}$	$0_5^{(5)}$	–	–	–	–
$2_6^{(7)}$	$2_6^{(8)}$	$1_6^{(9)}$	$1_6^{(9)}$	$0_6^{(8)}$	$0_6^{(7)}$	$3_6^{(10)}$	$3_6^{(10)}$
$2_6^{(11)}$	$2_6^{(12)}$	$1_6^{(13)}$	$1_6^{(13)}$	$0_6^{(12)}$	$0_6^{(11)}$	–	–
$3_7^{(14)}$	$3_7^{(15)}$	2_7	2_7	1_7	1_7	$0_7^{(15)}$	$0_7^{(14)}$
3_7	3_7	2_7	2_7	1_7	1_7	0_7	0_7
...	...	...	...	...	...	...	...

(1) for $C_{4,0}$; and (2) for $C_{0,4}$ computed on wave 4
(3) for $C_{5,0}$; and (4) for $C_{4,1}$ computed on wave 5
(5) for $C_{1,4}$; and (6) for $C_{0,5}$ computed on wave 5
(7) for $C_{6,0}$; and (8) for $C_{4,2}$ computed on wave 6
(9) for $C_{5,1}$; and (10) for $C_{3,3}$; computed on wave 6
(11) for $C_{2,4}$; and (12) for $C_{0,6}$ computed on wave 6
(13) for $C_{1,5}$ computed on wave 6
(14) for $C_{7,0}$; and (15) for $C_{4,3}$ computed on wave 7

process 0 is shown as sending to and receiving from himself, but cases in which sender and receiver are the same are treated specially to gain efficiency. For wave 1, process 0 send to and receive from process 1, and for wave 3 process 0 send to and receive from processor 2. In the meanwhile, process 3 is waiting that process 0 ends all the send / receive / compute previous to wave 3 in the wavefront in

order to start working. This delay can be interpreted as a *pipeline filling delay*. The next time process 3 exchanges data (send - receive) with process 0 is at wave 7, but until that, process 0 has to complete six $C_{i,j}$ blocks and process 3 only three of them, generating another delay, in this case explicitly due to the wavefront pattern. Theses delays can be explained as *computing dependency* as oposed to data dependency, because they arise from the compute processing order.

An implementation technique is used for avoiding the delays implied by the wavefront pattern whitout changing the $C_{i,j}$ processing order. The basis for delay avoidance is to increase the number of non blocking send / receive calls in each computer. This was implemented by launching in each node a pair of non blocking send - receive for each of the other nodes in the cluster, and taking into account the tags launched and pending to be completed. With this new outline of data communications, the computing dependency delays are removed because more than one processor has pending communications, allowing to complete some of them while the others are awaiting. Table 3 exposes the times obtained in this variant, over Infiniband, for the point to point algorithm and compared with broadcast implementation. It can be seen that times are about the same for both algorithms, with some cases in which point to point is better.

Table 3. Execution times of both algorithms over Infiniband

Matrix Size	**PtP Infiniband (secs.)**	**Broadcast Infiniband (secs.)**	**PtP/Bcast. Infiniband**
2000	0.584	0.625	0.934
4000	1.993	2.001	0.996
8000	11.748	11.195	1.049
12000	36.852	35.880	1.027

4 Related Work

Matson et.al. [2] defines a parallel pattern for algorithms named "Geometric Decomposition Pattern" where data is decomposed according to updates in the computation order and cite as example a mesh-computation program. The wavefront pattern can be seen as a form of this pattern. In [2] it is mentioned that good results can be obtained if computation and communication can be overlapped but they do not provide any performance information about the application of the involved patterns.

A parallel programming wavefront pattern is well defined by Snir [3]. Data is distributed in a logical plane or space, and the computations start at an edge and advances in diagonal to the opposite edge like a *wave*. It is used to solve different kind of problems, like the ones in genetics, LU factorization and others, with good performance results in shared memory systems [4].

A recent work about dynamic task scheduling in the linear algebra domain [5] is oriented to maintain scalability in a tiled algorithm, where the tasks are assigned dynamically by a centralized *task manager*. In this work barriers are eliminated, data is distributed among processors in a 2D pattern, and synchronization and communication are done by two specialized threads put aside of computing. This work was done in the context of PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures) project [6] devoted to take advantage of the multicore features of the new processors.

Parallel tiled algorithms for matrix factorizations are well analyzed in [7]. The focus is on fine granularity and asynchronous computing inside multicore systems. A graph of task dependencies is used to launch task asynchronously. There are not possible extensions to distributed systems. Also in the tiled algorithms field, Drosinos and Kosiris [8] presents a work about parallelization of tiled algorithms on SMP clusters. A general kind of algorithms is considered. MPI is used as the communication library and OpenMP for task distrubution in threads, suggesting several alternatives about a combined parallelization, focusing on load balance. Only Ethernet based results are given.

To the best of our knowledge, there are not works comparing relative efficiency for Ethernet and Infiniband based cluster algorithms. Most of them are oriented to evaluate the Infiniband implementation of the MPI library [9] [10] [11].

5 Conclusions and Further Work

The hypothesis presented is that the Infiniband architecture in a cluster allows to obtain results not expected by inefficiencies in an Ethernet based cluster under some communications patterns. By another side, the order in which individual tasks are computed must be suitable to the communication architecture. It could be determined that for the matrix multiplication problem in a small cluster of multicore nodes, based on an Infiniband network, to use an algorithm based on point to point communications is as good as, or in same cases better than the classic broadcast based algorithm. The order in which the computations are made must be adapted to the features of the Infiniband network. A wavefront-like order is valid but is not optimum because of the dependencies generated in the task computing order, so multiples non blocking pending send / receive operations must be launched in each processor to overcome the dependencies.

Pending to prove is the validity of these results under a cluster with more nodes. The MPI's broadcast primitive implementation use tree based algorithms for distribute the messages over all the nodes. Large number of nodes in the cluster allow to expect the confirmation of the conclusions.

Open to study is the validation of the present conclusions in algorithms where the data dependency is stronger, like those in matrix factorization algorithms, such as LU, QR and Cholesky methods, where real dependencies must be maintained. The parallel implementation of these algorithms are traditionally based on collective communications, although they are good examples for using point

to point messages due the nature of its computations. Also open is the existence of better scheduling policies that improve the results obtained.

References

[1] Tinetti Fernando G., Wolfmann Gustavo: Parallelization analysis on clusters of multicore nodes using shared and distributed memory parallel computing models. CSIE 2009, Los Angeles, USA, ISBN 978-7695-3507-4, 466–470 (2009)

[2] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill: A Pattern Language for Parallel Programming; Addison Wesley Software Patterns Series; 2004.

[3] Snir, Marc: The wavefront pattern.
http://www.cs.uiuc.edu/homes/snir/PPP/patterns

[4] John Anvik, Steve MacDonald, Duane Szafron, Jonathan Schaeffer, Steven Bromling and Kai Tan: Generating Parallel Programs from the Wavefront Design Pattern. Proceedings of the 7th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'02), Fort Lauderdale, Florida, April 2002.

[5] Song, F., Yarkhan, A., Dongarra, J.: Dynamic Task Scheduling for Linear Algebra Algorithms on Distributed-Memory Multicore Systems. University of Tennessee Computer Science Technical Report, UT-CS-09-638, April 2009.

[6] Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) project. University of Tennessee. http://icl.cs.utk.edu/plasma/

[7] Buttari A., Langou J., Kurzak J. and Dongarra J.: A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing, 35(1):38–53,2009, Elsevier Science, Amsterdam, The Netherlands.

[8] N. Drosinos and N. Koziris: Efficient Hybrid Parallelization of Tiled Algorithms on SMP Clusters International Journal of Computational Science and Engineering,2006.

[9] A.R. Mamidala, Jiuxing Liu, D.K. Panda: Efficient Barrier and Allreduce on Infiniband clusters using multicast and adaptive algorithms, cluster, 135–144, Sixth IEEE International Conference on Cluster Computing (CLUSTER'04), 2004

[10] Sayantan Sur, Hyun-Wook Jin, Dhabaleswar K. Panda: Efficient and Scalable All-to-All Personalized Exchange for InfiniBand-Based Clusters, icpp, pp.275-282, International Conference on Parallel Processing (ICPP'04), 2004

[11] Hoefler T., Siebert C., Rehm W.: A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast Parallel and Distributed Processing Symposium, IPDPS 2007 1–8, March 2007, Long Beach, CA, USA.

[12] InfiniBand Trade Association: InfiniBand Architecture www.infinibandta.org

[13] Open MPI project: www.open-mpi.org

[14] J. Choi: A Fast Scalable Universal Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers, ipps, pp.310, 11th International Parallel Processing Symposium (IPPS '97), 1997

[15] Mellanox Technologies www.mellanox.com

[16] Sun HPC ClusterTools http://www.sun.com/software/products/clustertools

[17] Sun Studio Express http://developers.sun.com/sunstudio/

[18] Sun Studio Performance Analizer
http://developers.sun.com/sunstudio/overview/topics/analyzer_index.html