MPI Collective Algorithm Selection and Quadtree Encoding

Jelena Pješivac–Grbović, Graham E. Fagg,
Thara Angskun, George Bosilca, and Jack J. Dongarra

Innovative Computing Laboratory,
University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
{pjesa, fagg, angskun, bosilca, dongarra}@cs.utk.edu

Abstract. In this paper, we focus on MPI collective algorithm selection process and explore the applicability of the quadtree encoding method to this problem. During the algorithm selection process, a particular MPI collective algorithm is selected based on the collective operation parameters. We construct quadtrees with different properties from the measured algorithm performance data and analyze the quality and performance of decision functions generated from these trees.
The experimental data indicates that in some cases, the decision function based on quadtree structure with a mean depth of 3 can incur as little as a 5% performance penalty on average. The exact, experimentally measured, decision function for all tested collectives could be fully represented using quadtrees with a maximum of 6 levels. These results indicate that quadtrees may be a feasible choice for both processing of the performance data and automatic decision function generation.

1 Introduction

The performance of MPI collective operations is crucial for good performance of MPI application which use them [1]. For this reason, significant efforts have been put on design and implementation of efficient collective algorithms both for homogeneous and heterogeneous cluster environments [2–10]. Performance of these algorithms varies with the total number of nodes involved in communication, system and network characteristics, size of data being transferred, current load, and if applicable, the operation that is being performed as well as the segment size which is used for operation pipelining. Thus, selecting the best possible algorithm and segment size combination (*method*) for every instance of collective operation is important.

In order to achieve the best possible performance, one can tune the collective operations for a particular system. The process of tuning MPI collective operations often involves detailed profiling of the system possibly combined with communication modeling, analyzing the collected data, and generating a *decision function*. During run-time, the decision function selects close-to-optimal method for a particular collective instance. This approach relies on the ability of the decision function to accurately predict algorithm and segment size to be used for the particular collective instance. Alternatively, one could construct an in-memory decision system which could be queried/searched at the run-time to provide the optimal method information. In order for either of these approaches to be feasible, the memory footprint and the time it takes to make decisions need to be minimal.

This paper studies the applicability of the quadtree encoding method as a storage and optimization technique within the MPI collective method selection process. We assume that the system of interest has been benchmarked and that detailed performance information exists for each of available collective communication algorithm. With this information, we focus our efforts on investigating whether the quadtree encoding is a feasible way to generate static decision functions as well as, to represent the decision function in memory.

We implemented a prototype quadtree implementation and programs to analyze the experimental performance data, construct the quadtree decision functions, and analyze their performance penalty in comparison to the exact decision function. We collected detailed profiles for broadcast and reduce MPI collective algorithms on two different clusters, and analyzed the quality of decisions from quadtrees built using this data but under different constraints.

The paper proceeds as follows: Section 2 discusses existing approaches to the decision making/algorithm selection problem; Section 3 describes the quadtree construction and analysis of quadtree decision function in more detail; Section 4 presents experimental results; Section 5 concludes the paper with discussion of the results and future work.

2 Related work

The algorithm selection problem can be solved using various techniques.

Currently, in the FT-MPI [11], the decision function is generated manually using visual inspection method augmented with Matlab scripts used for analysis of the experimentally collected performance data. This approach results in a precise albeit complex decision functions. The advantage of this approach is the potential to recognize any possible problems in measured data and the ability to make decision to exclude peculiar data points.

In the MPICH-2 MPI implementation [12], the algorithm selection process is done statically based primarily on the message size of the collective (which can be either short or long vector) and whether the number of processors is a power of two or not [6]. In this library, the algorithm selection is based on bandwidth and latency requirements of an algorithm, and the switching points are predetermined by the implementers.

In the tuned collective module of the Open MPI [13], the algorithm selection can be done in either of the following three ways: via compiled decision function, via user-specified command line flags, or using rule-based run-length encoding scheme which can be tuned for particular system.

Another possibility is to view this problem as a data mining task in which the algorithm selection problem is replaced by an equivalent classification problem. The new problem is to classify collective parameters, *(collective operation, communicator size, message size)*, into a correct category, a method in our case, to be used at run time. The major benefit of this approach is that the decision making process is a well studied topic in engineering and machine learning fields. Decision trees are extensively used in pattern recognitions, CAD design, signal processing, medicine, biology, and search engines [14].

Alternatively, one can interpret the optimal collective implementation on a system, i.e. a *decision map*, as an image and apply a standard compression algorithms to it. Figure 1 illustrates a couple of decision maps for the broadcast operation on Grig cluster. In this work, we build quadtrees by interpreting the experimentally measured optimal decision map as a bit pattern and then encode it using a similar technique to an image encoding process. To the best of our knowledge, we are the only group which has approached the MPI collective tuning process in this way.

3 Quadtrees and MPI collective operations

3.1 Building a quadtree decision structure

We assume that detailed system profiling (either by extensive experimental measurements or via modeling) has been previously performed, and that the performance information for different collective algorithms for a range of communicator and message sizes and predetermined set of segment sizes is available.

We use this performance information to extract the information about the optimal methods and construct a decision map. An example of a decision map is displayed in Table 1. The decision map which will be used

Communicator size (y-axis)	Message size (x-axis)	Algorithm	Segment size	Method index
3	1	Linear	none	1
3	2	Linear	none	1
...	...	...	...	...
128	64KB	BinaryTree	8KB	13

Table 1. Decision map example. The axis information relates to the decision maps in Figure 1.

to initialize the quadtree must be a complete and square matrix with a dimension size that is a power of two, $2^k \times 2^k$. Complete decision map means that tests must cover all message and communicator sizes of interest. Neither of these requirements are real limitations, as the missing data can be interpolated and the size of the map can be adjusted by replicating some of the entries. The replication process does not affect the quadtree decisions, but may affect efficiency of the encoding (both in positive and negative manner).

Once a decision map is available, we initialize the quadtree from it using user specified constraints such as *accuracy threshold* and *maximum allowed depth* of the tree. The accuracy threshold is the minimum percentage of points in a block with the same "color", such that the whole block is "colored" in that "color". The quadtree with no maximum depth set and threshold of 100% is an exact tree. The exact tree truthfully represents the measured data. A quadtree with either threshold or maximum depth limit set allows us to reduce the size of the tree at the cost of prediction accuracy, as it is no longer an exact copy of the original data. Limiting the absolute tree depth limits the maximum number of tests we may need to execute to determine the method index for specified communicator and message size. Setting the accuracy threshold helps smooth the experimental data, thus possibly making the decision function more resistant to inaccuracies in measurements. Applying the maximum depth and/or the accuracy thresholds is equivalent to applying low-pass filters to the original data set.

3.2 Generating decision function source code

A property of any decision tree is that an internal node of the tree corresponds to an attribute test, and the links to children nodes correspond to the particular attribute values. In our encoding scheme, every non-leaf node in the quadtree corresponds to a test which matches both communicator and message size values. The leaf nodes contain information about the optimal method for the particular communicator and message size ranges. Thus, leaves represent the rules of the particular decision function. In effect, quadtrees allow us to perform a recursive binary search in a two-dimensional space.

We provide functionality to generate decision function source code from the initialized quadtree. Recursively, for every internal node in the quadtree we generate the following code segment:

if (NW) {*...*} *else if (NE)* {*...*} *else if (SW)* {*...*} *else if (SE)* {*...*} *else* {*error*}.

The current implementation is functional but lacks optimizations, i.e. ability to merge conditions with same color[1]. The conditions for boundary points (minimum and maximum communicator and message sizes) are expanded to cover that region fully. For example, the rule for minimum communicator size will be used for all communicator sizes less than the minimum communicator size.

3.3 In-memory quadtree decision structure

Alternative to generating the decision function source code is maintaining an in-memory quadtree decision structure which can be queried during the run time.

An optimized quadtree structure would contain 5 pointers and 1 method field, which could probably be a single byte or an integer value. Thus, the size of a node of the tree would be around 44B on 64-bit architectures[2]. Additionally, the system would need to maintain in memory the mapping of (algorithm, segment size) pairs to method indexes as well. The maximum depth decision quadtree we encountered in our tests had 6 levels. This means that in the worst case, the 6-level decision quadtree could take up to $\frac{4^7-1}{4-1} = 5461$ nodes, which would occupy close to 235KB of memory. However, our results indicate that the quadtrees with 3 levels can still produce reasonably good decisions. Three-level quadtree would occupy at most 3740B and as such could fit into 4 1KB pages of main memory. Even so, the smaller quadtree if cached would still occupy significant portion of the cache. We think that the memory usage and memory access time overhead of searching this structure could be prohibitively expensive at run time.

[1] The code segment generated for each internal node contains at least 21 lines – 5 lines for conditional expressions, 10 lines for braces, a line for error handling, and at least a line per condition.

[2] In this analysis, we ignore data alignment issues which would lead to even larger size of the structure.

4 Experimental results and analysis

In order to determine whether quadtrees are a feasible choice for encoding the automatic method selection process for MPI collective operations, we analyzed the accuracy of quadtrees built from the same experimental data but under different constraints. Under the assumption that the collective operations parameters are uniformly distributed across communicator size and message size space, we expect that the average depth of the quadtree is the average number of conditions we need to evaluate before we can determine which method to use. In the worst case, we will follow the longest path in the tree to make the decision, and in the best case the shortest.

Based on memory requirements analysis in Section 3.3 we decided not to implement the in-memory quadtree-based decision structure yet. Instead, we implemented the library which generates the source code for the decision function. The number of tests in generated decision function could potentially be reduced by merging the rules for blocks of the same color. However, in the worst-case scenario when every block has a different color, this optimization is not possible. Thus, we believe that analysis of the unoptimized implementation is acceptable.

The performance data for broadcast and reduce collective algorithms was collected on Grig cluster located at the University of Tennessee at Knoxville and Nano cluster located at the Lawrence Berkeley National Laboratory.

4.1 Decision quadtree examples

Figure 1 shows six different quadtree decision maps for a broadcast collective. The experimental data was collected on the Grig cluster at University of Tennessee. We considered five different broadcast algorithms (Linear, Binomial, Binary, Splitted-Binary, and Pipeline),[3] and four different segment sizes (no segmentation, 1KB, 8KB, and 16KB). The measurements covered all communicator sizes between 2 and 28 processes and message sizes in 1B to 384KB range.

The coarsening of the decision function in Figure 1 was achieved by limiting the maximum depth of the quadtree. The exact function exhibits trends, but there is a considerable amount of information for intermediate size messages (between 1KB and 10KB) and small communicator sizes. Limiting the maximum tree depth smoothes the decision map and subsequently decreases the size of the quadtree. Table 2 shows the mean tree depth and related statistics for the decision maps presented in Figure 1.

4.2 Performance penalty of decision quadtrees

One possible metrics of merit is the performance penalty one would incur by using a restricted quadtree instead of the exact one. To compute this, one can use the performance information for methods suggested by the restricted tree for particular set of communicator and message size values, and compare them to the performance results for methods suggested by the exact tree.

While this is a straight-forward approach, one must be aware of the following issues. The main problem arises from the fact that the exact decision function fits to the particular experimentally measured data collected on a particular date and time, software, and hardware. The discussion of measurement quality and reproducibility is out of scope of this paper, however, one must be aware that exact timing of collective operations is a very hard problem in systems without synchronized clock or extremely fast hardware barriers[4]. As with most experimental measurements, measurement introduces an error. In our tests, we kept the variation of measurement to less or equal to 5%. However, for some algorithms and collective parameter sizes, the performance difference between two methods was less than 5%. In this case, the exact tree would use the method which achieved absolute minimum, even though it is possible that the other method was a

[3] For more details on these algorithms, refer to [10]

[4] The system we used had neither a synchronized clock nor hardware barrier. The main test system in this paper, Grig cluster, is a cluster of PCs connected by Fast Ethernet.

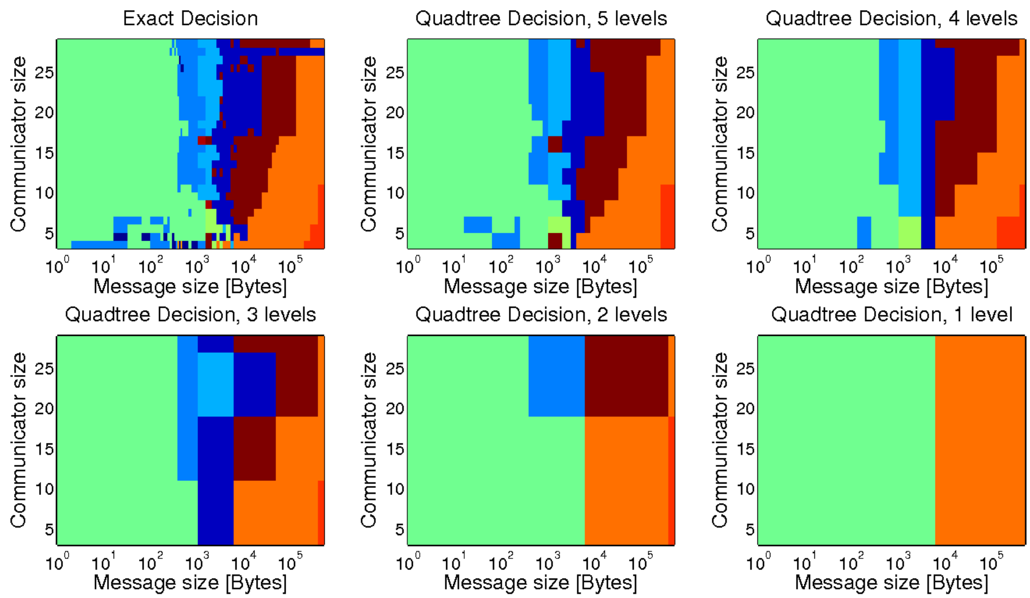

Fig. 1. Broadcast decision maps from Grig cluster. Different colors correspond to different method indexes. The trees were generated by limiting the maximum tree depth. The x-axis scale is logarithmic. The crossover line for 1-level quadtree is not in the middle due to the "fill-in" points used to adjust the original size of the decision map from 25×48 to 64×64 form.

better choice overall. Thus, a fair comparison must compare the performance of a quadtree decisions against the data that was used to generate them in the first place.

Figure 2 shows the performance penalty of decision quadtrees from Figure 1 and the Table 2 summarizes the properties and performance penalties for the same data. The analysis shows that even for noisy decision

Tree Depth			Performance Penalty [%]				Number of	Function size
Max	**Min**	**Mean**	**Min**	**Max**	**Mean**	**Median**	**Leaves**	**[# of lines]**
1	1	1.0000	0.00	346.05	37.11	0.00	4	24
2	2	2.0000	0.00	436.02	18.63	0.00	16	82
3	2	2.9655	0.00	436.02	08.83	0.00	58	330
4	2	3.8554	0.00	391.53	06.29	0.00	166	932
5	2	4.7783	0.00	356.47	05.41	0.00	442	2,496
6	2	5.6269	0.00	000.00	00.00	0.00	973	5,505

Table 2. Statistics for broadcast decision quadtrees in Figure 1. The number of leaves corresponds to the number of regions we divided the (communicator size, message size) space into. The number of lines in decision function includes lines containing only braces, error handling, etc.

map in Figure 1, a 3-level quadtree would have less than 9% performance penalty on average, while the exact decision could be represented with a total of 6 levels.

4.3 Quadtree accuracy threshold

In Section 3.2 we mentioned that an alternative way to limit the size of quadtree is to specify the tree accuracy threshold.

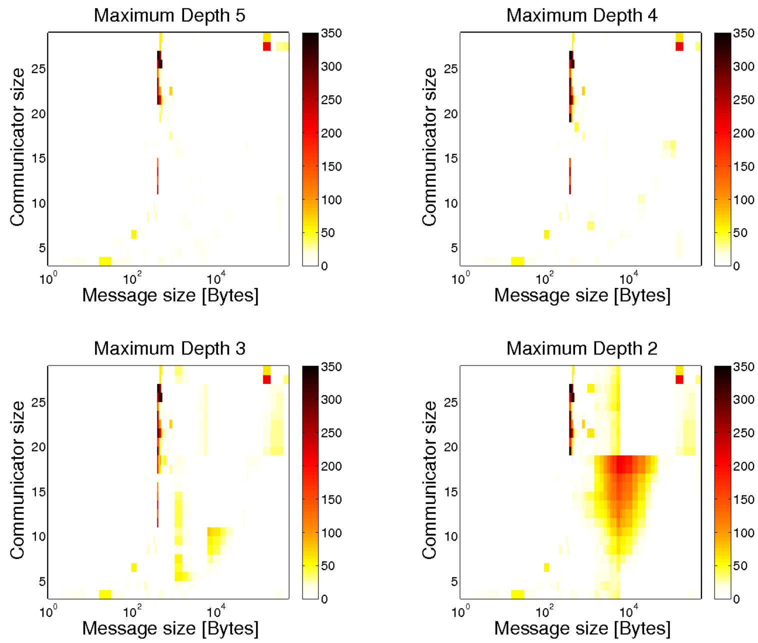

Fig. 2. Performance penalty of broadcast decision function from Grig cluster. Colorbar represents relative performance penalty in percents. White color means less than 25%, yellow is between 25% and 75%.

Figure 3 shows the effect of varying the accuracy threshold on the mean performance penalty of a reduce quadtree decision function on two different systems. On both systems, the mean performance penalty of the reduce decision was below 10% for an accuracy threshold of approximately 45%. This threshold corresponds to the quadtree structures of maximum depth 3. This means that the quadtree decision which would on average potentially cause a 10% performance penalty would be evaluated at most in 3 expressions.

4.4 Accuracy threshold vs. limiting maximum depth

Figure 4 shows the mean performance penalty of broadcast and reduce decisions on Grig cluster (See Figures 1, 2, and 3, and Table 2) as a function of the mean quadtree depth for quadtrees constructed by specifying accuracy threshold and maximum depth. The results indicate that in the cases we considered, constructing the decision quadtree by restricting the maximum depth of the tree directly incurs a smaller mean performance penalty than the tree of similar mean depth constructed by setting the accuracy threshold.

The results for the broadcast decision function show that when the quadtree is deep enough to cover almost the whole initial data set, the tree constructed using an accuracy thresholds achieves the smaller mean performance penalty. This is not the case for the quadtree-based reduce decision functions. This is probably due to the fact that the reduce decision function was smoother to start with, so smoothing it with an accuracy threshold had no further positive effects. Still, we believe that the example of the broadcast decisions indicates that the accuracy threshold setting could be used to avoid over-fitting the data when the tree depth is not a concern.

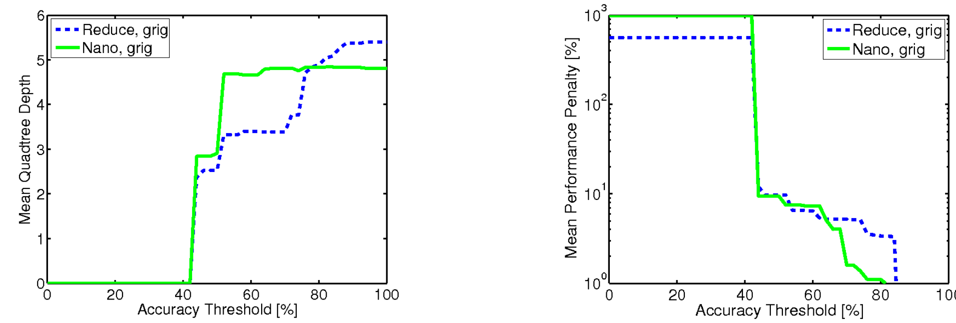

Fig. 3. Effect of the accuracy threshold on mean quadtree performance penalty.

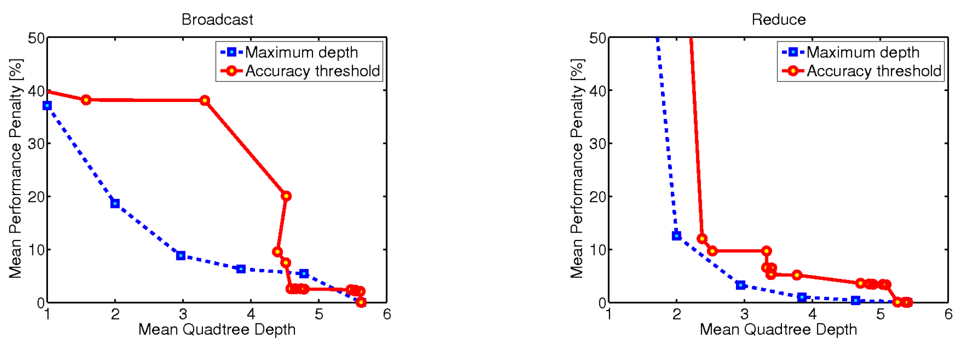

Fig. 4. Accuracy threshold vs. maximum depth quadtree construction.

5 Discussion and future work

In this paper, we studied the applicability of a modified quadtree encoding method to the algorithm selection problem for the MPI collective function optimization. We analyzed the properties and performance of quadtree decision functions constructed by either limiting the maximum tree depth or specifying the accuracy threshold a the construction time.

The experimental data collected on clusters located at the University of Tennessee at Knoxville for broadcast and reduce collectives, show that in some cases, the decision function based on a quadtree structure with a mean depth of 3, incurs less than a 5% performance penalty on the average. In other cases, deeper trees (5 or 6 levels) were necessary to achieve the same performance. However, in all cases we considered, a quadtree with 3-levels would incur less than a 10% performance penalty on average. Our results indicate that quadtrees may be a feasible choice for processing the performance data and decision function generation. In this work we chose not to explore the performance of the in-memory quadtree decision systems due to relatively large memory requirements associated with storing the tree. The performance of an in-memory system will depend greatly on the implementation efficiency and the application access pattern. While we believe that in general, an in-memory quadtree decision system may not be the best solution, it is possible that in some cases it could achieve very good performance. We plan to explore this issue in more depth in the future.

One of the limitations of the quadtree encoding method is that since the decision is based on a 2D-region in communicator size - message size space, it will not be able to capture decisions which are optimal for single communicator values, i.e. communicator sizes which are power of 2. The same problem is exacerbated if the performance measurement data used to construct trees is too sparse. The sparse data set is a high-frequency information and applying low-pass filters to it can cause loss of important information.

The decision map reshaping process to convert measured data from $n \times m$ shape to $2^k \times 2^k$ may affect encoding efficiency of the tree both positively and negatively. In our current study, we did not address this issue, but in future work we plan to further improve the efficiency of the encoding regardless of initial data space.

The major focus of future research will be comparing the quadtree-based decision functions, to the ones generated using run-length encoding and standard decision tree algorithms such as C4.5.

Finally, if one is interested in an application level optimization, assumptions based on the premise that the communication parameters are uniformly distributed across the communicator and message size space are probably optimistic. Thus, it is possible that it would make sense to refine the trees for frequently used message and communicator sizes while the rest of the domain is more sparse. Quadtrees may or may not be right structure for this type of approach, but we plan to investigate this approach additionally.

Acknowledgments

This work was supported by Los Alamos Computer Science Institute (LACSI), funded by Rice University Subcontract #R7B127 under Regents of the University Subcontract #12783-001-05 49.

References

1. Rabenseifner, R.: Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512. In: Proceedings of the Message Passing Interface Developer's and User's Conference. (1999) 77–85
2. Huse, L.P.: Collective communication on dedicated clusters of workstations. In: Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in PVM and MPI, Springer-Verlag (1999) 469–476
3. Worringen, J.: Pipelining and overlapping for MPI collective operations. In: 28th Annyal IEEE Conference on Local Computer Network, Boon/Königswinter, Germany, IEE Computer Society (2003) 548–557
4. Rabenseifner, R., Träff, J.L.: More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems. In: Proceedings of EuroPVM/MPI. Lecture Notes in Computer Science, Springer-Verlag (2004)
5. Chan, E.W., Heimlich, M.F., Purkayastha, A., van de Geijn, R.M.: On optimizing of collective communication. In: Proceedings of IEEE International Conference on Cluster Computing. (2004) 145–155
6. Thakur, R., Gropp, W.: Improving the performance of collective operations in MPICH. In Dongarra, J., Laforenza, D., Orlando, S., eds.: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2840 in LNCS, Springer Verlag (2003) 257–267 10th European PVM/MPI User's Group Meeting, Venice, Italy.
7. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: MagPIe: MPI's collective communication operations for clustered wide area systems. In: Proceedings of the seventh ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, ACM Press (1999) 131–140
8. Bernaschi, M., Iannello, G., Lauria, M.: Efficient implementation of reduce-scatter in MPI. Journal of Systems Architure **49**(3) (2003) 89–108
9. Bruck, J., Ho, C.T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Transactions on Parallel and Distributed Systems **8**(11) (1997) 1143–1156
10. Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G.E., Gabriel, E., Dongarra, J.J.: Performance analysis of mpi collective operations. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15, Washington, DC, USA, IEEE Computer Society (2005) 272.1
11. Fagg, G.E., Gabriel, E., Bosilca, G., Angskun, T., Chen, Z., Pješivac-Grbović, J., London, K., Dongarra, J.: Extending the mpi specification for process fault tolerance on high performance computing systems. In: Proceedings of the International Supercomputer Conference (ICS) 2004, Primeur (2004)
12. Mathematics, Computer Science Division, A.N.L.: MPICH-2, implementation of MPI 2 standard. http://www-unix.mcs.anl.gov/mpi/mpich2/ (2006) Accessed on March.
13. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (2004) 97–104
14. Murthy, S.K.: Automatic construction of decision trees from data: A multi-disciplinary survey. Data Mining and Knowledge Discovery **2**(4) (1998) 345–389