Using Quadtree Encoding for MPI Collective Algorithm Selection Process

Jelena Pješivac–Grbović, Graham E. Fagg, Jack J. Dongarra

May 15, 2006

Abstract

The performance of an MPI collective operation depends on a number of different factors, some of which are at the hardware level such as network characteristics and topology, while some others are at the software level: algorithm implementation, choice of segment size (if applicable), and the collective operation parameters. Selecting the best possible collective implementation for the particular set of parameters is important for overall application performance. In this paper, we focus on MPI collective algorithm selection process and explore the applicability of the quadtree encoding method to this problem. 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 shows that in some cases, the decision function based on a 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. We also evaluate the performance of prototype version of quadtree-based in-memory decision systems, and find that that the decision can be obtained from a 6-level quadtree in close to $170ns$. In comparison, fixed reduce decision in FT-MPI takes $45ns$ to evaluate and is not an exact decision. Our results indicate that quadtrees may be a feasible choice for both processing of the performance data and automatic decision function generation.

Contents

List of Figures

List of Tables

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, 3, 4, 5, 6, 7, 8, 9, 10, 11]. 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.

To ensure good performance of MPI applications, collective operations can be tuned for the particular system. The tuning process 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 experimental data collected on Grig cluster located at the University of Tennessee at Knoxville for broadcast and reduce MPI collectives, shows that the decision function based on quadtree structure with a mean depth of 3, on average can incur as little as a 5% performance penalty. The exact, experimentally measured, decision function for both collectives on this system could be fully represented using quadtrees with a maximum of 6 levels. We also measured performance of the prototype version of in-memory quadtree-based decision system and found that 6-level tree can return decision in close to $170ns$ on average. The 3-level tree search for both reduce and broadcast took around $150ns$ on average. This is comparable to $45.22ns$ and $110.53ns$ for fixed decision functions in FT-MPI for reduce and broadcast collectives respectively. These results indicate that quadtrees may be a feasible choice for processing of the performance data and decision function generation.

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; finally the appendix A provides some implementation details about the library.

2 Related work

The MPI collective algorithm selection problem has been addressed in many MPI implementations.

In the FT-MPI [12], 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. In the MPICH-2 MPI implementation, the algorithm selection is based on bandwidth and latency requirements of an algorithm, and the switching points are predetermined by the implementers [7]. 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.

It is also possible 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

	Non-categorical attributes		Categorical attributes		
Collective	**Communicator size**	**Message size**	**Algorithm name**	**Segment size**	**Method index**
Broadcast	24	1024	Linear	0	1
Broadcast	27	2048	Linear	0	1
Broadcast	28	131072	Binomial	8192	12
Reduce	11	122880	Pipeline	1024	32
Barrier	96	N/A	Bruck	N/A	64

Table 1: Interpreting the algorithm selection problem as a classification problem.

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 decision map for the reduce operation on Nano cluster. We then use the encoded structure to generate decision function code. To the best of our knowledge, we are the only group which has approached the MPI collective tuning process in this way.

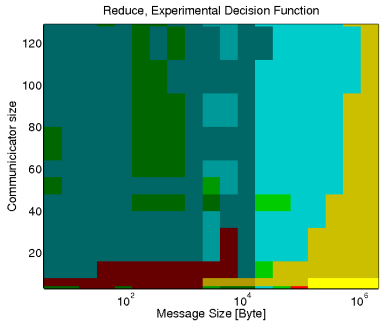

Figure 1: Reduce decision map from Nano cluster. Different colors correspond to different method indexes.

Communicator size (y-axis)	Message size (x-axis)	Algorithm	Segment size	Method index
3	1	Linear	none	1
3	2	Linear	none	1
...	...	...	...	...
16	128	Linear	none	1
...	...	...	...	...
128	64KB	BinaryTree	8KB	13
...	...	...	...	...

Table 2: Decision map example. The axis information relates to the decision maps in Figure 2.

3 Quadtrees and MPI collective operations

We use the collective algorithm performance information on a particular system to extract the information about the optimal methods and construct a *decision map* for the collective on that system. An example of a decision map is displayed in Table 2. The decision map which will be used 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).

3.1 Building the quadtree decision structure

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 Decision quadtree properties

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.

3.3 Generating decision function source code

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.

3.4 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].

[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.

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 could still occupy significant portion of the cache.

Our prototype implementation provides program for managing the in-memory decision function - however the node structure has 104B instead of minimal 44B. We believe that optimized version should achieve significantly better performance than the prototype version.

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 and the performance[3] 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.

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 Broadcast decision maps

Figure 2 shows six different quadtree decision maps for a broadcast collective on the Grig cluster. We considered five different broadcast algorithms (Linear, Binomial, Binary, Splitted-Binary, and Pipeline),[4] and four different segment sizes (no segmentation, 1KB, 8KB, and 16KB). The measurements

[3] Performance of quadtree is the time it takes to make a decision, should not be confused with performance penalty which is related to the accuracy of the quadtree-based decision function.

[4] For more details on these algorithms, refer to [11]

covered all communicator sizes between 2 and 28 processes and message sizes in 1B to 384KB range.

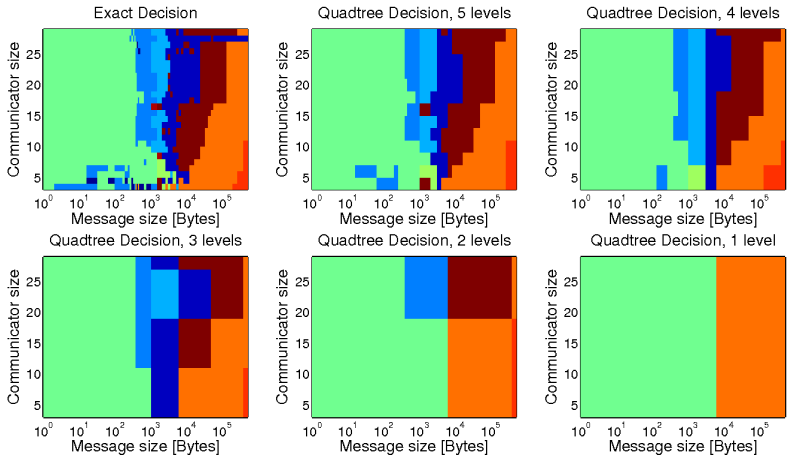

Figure 2: Broadcast decision maps from Grig cluster. Different colors correspond to different method indexes. The trees were generated by limiting the maximum tree depth. The x-axis scale is logarithmic. The crossover line for 1-level quadtree is not in the middle due to the "fill-in" points used to adjust the original size of the decision map from 25×48 to 64×64 form.

The exact decision map in Figure 2 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 3 shows the mean tree depth and related statistics for the decision maps presented in Figure 2.

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.

The reproducibility of measured results is out of scope of this paper, but we followed the guidelines from [15] to ensure good quality measurements.

Even so, the "exact" decision function corresponds to a particular data set, and the performance penalty of other decision functions was evaluated against the data that was used to generate them in the first place.

Figure 3 shows the performance penalty of decision quadtrees from Figure 2 and the Table 3 summarizes the properties and performance penalties for the same data. The analysis shows that even for noisy decision map in

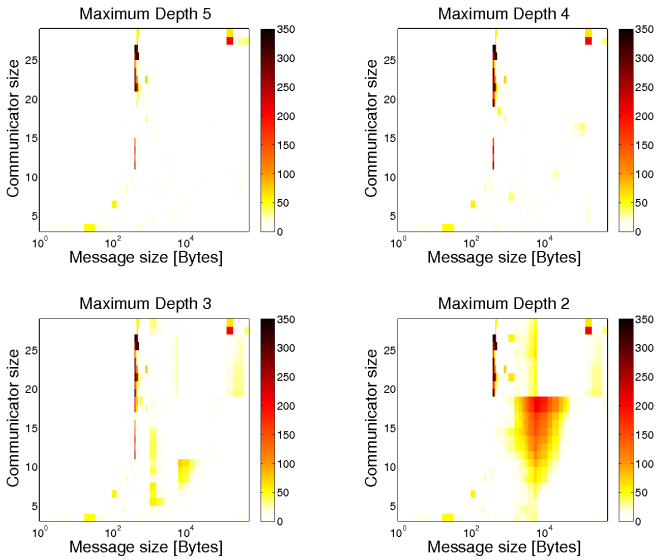

Figure 3: Performance penalty of broadcast decision function from Grig cluster. Colorbar represents relative performance penalty in percents. White color means less than 25%, yellow is between 25% and 75%.

Figure 2, 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 3.3 we mentioned that an alternative way to limit the size of quadtree is to specify the tree accuracy threshold.

Figure 4 shows the effect of varying the accuracy threshold on the mean performance penalty of a reduce quadtree decision function on two different

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 3: Statistics for broadcast decision quadtrees in Figure 2. 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.

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.

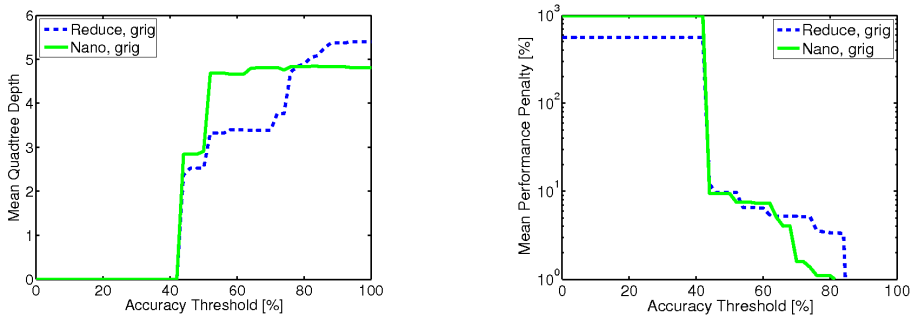

Figure 4: Effect of the accuracy threshold on mean quadtree performance penalty.

4.4 Accuracy threshold vs. limiting maximum depth

Figure 5 shows the mean performance penalty of broadcast and reduce decisions on Grig cluster (See Figures 2, 3, and 4, and Table 3) 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

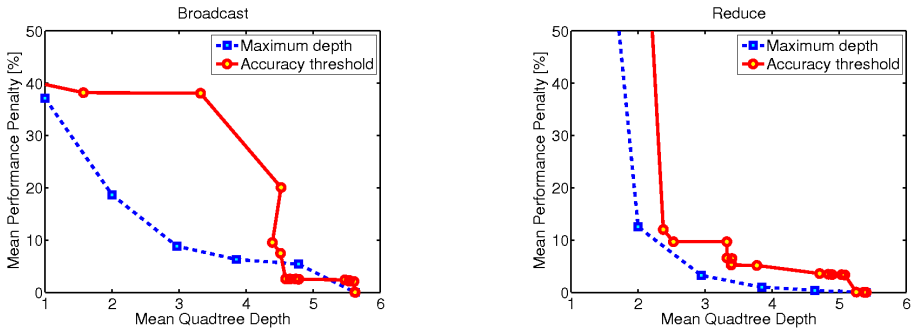

Figure 5: Accuracy threshold vs. maximum depth quadtree construction.

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.

4.5 In-memory quadtree-based decision system

Table 4 contains the performance results for in-memory quadtree-based decision system for reduce collective on Grig cluster. The reported decision time value is the average time it took for the quadtree to make decision for a random communicator and message size from the $2-32$ and $1-16MB$ ranges respectively. Even though the broadcast decision function was more complex than reduce on this cluster, the time to decision was actually lower.

The results in Table 4 show that using even unoptimized quadtree implementation the time to decision for these quadtrees was around $30ns$ per level. Furthermore, the time to decision for the complete tree with 6 levels took less than $175ns$ for reduce and $165ns$ for broadcast collective. The 3-level trees, which incur less than 10% performance penalty, took around $150ns$ for both collectives.

In comparison, the fixed reduce and broadcast decisions in FT-MPI took $45.22ns$ and $110.53ns$ on average. Even though we did not compute the

Mean tree depth	Mean performance penalty [%]	Number of leaves	Number of nodes	Memory [Bytes]	Decision time [nsec]	Decision time per level [nsec]
0.00	561.67	1	1	104	22.22	22.22
1.00	143.59	4	5	520	66.38	33.19
2.00	12.51	16	21	2,184	109.26	36.42
2.95	3.23	55	73	7,592	146.78	37.16
3.84	0.96	157	209	21,736	161.84	33.44
4.63	0.37	334	445	46,280	170.02	30.20
5.40	0.00	610	813	84,552	174.95	27.34

Table 4: Performance of prototype implementation of in-memory, quadtree-based decision system for reduce collective on Grig cluster. Time per level was computed as $\frac{\text{decision time}}{\text{mean depth}+1}$.

performance penalty for the fixed decision functions, we know they perform well on Grig cluster. Still, the results using unoptimized quadtree-based decision system are comparable to the fixed ones.

The optimized quadtree implementation will have both smaller memory footprint and better memory layout due to non-recursive construction. We expect that this quadtree implementation will decrease the time to make decision considerably.

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.

Our experimental results 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.

Our analysis of the performance of the in-memory quadtree decision systems was based on unoptimized quadtree implementation, and as such should be interpreted as worst case scenario. The results with this sys-

tem show that in around $170ns$ on average, we can get the experimentally optimal decision. Not surprisingly, the fixed decision function in FT-MPI outperformed the in memory system in terms of performance. Even so, the current results are good enough to persuade us to explore this track further. We expect that the performance of an in-memory system will depend greatly on the implementation efficiency and the application access pattern. It is possible that in some cases and or in combination with other methods, it could achieve very good performance. n

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, e.g. 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 $2^k \times 2^k$ affects encoding efficiency of the quadtree. 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.

A Library Information

A.1 Implementation Details

Data Structures

The following data structures represent the core of the quadtree decision function library:

Decision map/Decision structure. The decision map in this context it is a one-dimensional array whose value at location `n` represent the method index to be used for communicator size `commsize[n/nummsgsizes]` and message size `msgsize[n%nummsgsizes]`. The decision structure contains the additional information neccessary to interpret the decision map, eg. communicator and message sizes, topology and segment sizes, etc.

Experimental performance data. The structure `exp_perf_data_t` maintains the experimental performance data information. It maintains the array of linked list where `exp_data[i][j]` contains sorted performance information for communicator size
`commsize[i]` and message size `msgsize[j]`. The requested timer values can be obtained using appropriate functions (`epd_find_perf_data` and `epd_find_perf_timer`). The structure also provides functionality to compute the performance penalty of external decision map. However, we can only get the performance penalty for the map whose sample points (communicator and message sizes) are covered by the performance data.
Currently, the experimental performance data structure can be initialized only from the OCC performance data files [16].

Quadtree. The quadtree structure is basic quadtree implementation which supports the maximum tree length and the tree accuracy threshold as explained in Section 3.2.

Full decision. This is an composite data structure which contains the experimental performance data, decision, and quadtree data structures, and provides the functionality to query the quadtree using the communicator and message size, as well as to generate the source code of the decision function using this particular combination of experimental data and quadtree parameters.

Utility Programs

We have provided three utility programs with the code:

analyze_qt_dec_quality.c – analyzes the quality of decisions generated by the quadtree decision functions,

output_single_decision_function.c – generates source code for the specified decision function, and

measure_in_memory_decision_time – measures the time it takes to come to a decision.

All three programs have similar look and feel, and they support same input arguments for describing the quadtree structure. We have also provided a shell script, `bin/generate_qt_decision_tree_stats.sh` which allows user to run `analyze_qt_dec_quality.c` program for range of threshold values.

A.2 Installation

The Quadtree Decision Function V2 software can be obtained by contacting the author via email.

To install the software follow the steps below:

1. unpack the archive containing the source code:
   # tar -xzvf qtdec_v2.tar.gz
2. enter the newly created root directory for the library
   # cd quadtree_decision_function_analysis
3. review Makefiles in root, src/, and tests/ directory.
4. make program from the root directory
   # make
5. If everything goes as planned, executables will be generated in `bin/` directory. The `generate_qt_decision_tree_stats.sh` will be in the same directory.

References

[1] Rolf Rabenseifner. 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*, pages 77–85, 1999.

[2] J. Worringen. Pipelining and overlapping for MPI collective operations. In *28th Annyal IEEE Conference on Local Computer Network*, pages 548–557, Boon/Königswinter, Germany, October 2003. IEE Computer Society.

[3] Lars Paul Huse. 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*, pages 469–476. Springer-Verlag, 1999.

[4] Rolf Rabenseifner and Jesper Larsson Träff. 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] Sándor Juhász and Ferenc Kovács. Asynchronous distributed broadcasting in cluster environment. In Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra, editors, *PVM/MPI*, volume 3241 of *Lecture Notes in Computer Science*, pages 164–172. Springer, 2004.

[6] Ernie W. Chan, Marcel F. Heimlich, Avi Purkayastha, and Robert M. van de Geijn. On optimizing of collective communication. In *Cluster*, 2004.

[7] Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number 2840 in LNCS, pages 257–267. Springer Verlag, 2003. 10th European PVM/MPI User's Group Meeting, Venice, Italy.

[8] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. 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 programmin g*, pages 131–140. ACM Press, 1999.

[9] Massimo Bernaschi, Giulio Iannello, and Mario Lauria. Efficient implementation of reduce-scatter in MPI. *J. Syst. Archit.*, 49(3):89–108, 2003.

[10] Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra. Automatically tuned collective communications. In *Proceedings of the 2000*

ACM/IEEE conference on Supercomputing (CDROM), page 3. IEEE Computer Society, 2000.

[11] Jelena Pješivac-Grbović, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. Performance analysis of mpi collective operations. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15*, page 272.1, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Graham E Fagg, Edgar Gabriel, George Bosilca, Thara Angskun, Zizhong Chen, Jelena Pješivac-Grbović, Kevin London, and Jack Dongarra. Extending the mpi specification for process fault tolerance on high performance computing systems. In *Proceedings of the International Supercomputer Conference (ICS) 2004*. Primeur, 2004.

[13] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[14] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[15] William Gropp and Ewing L. Lusk. Reproducible measurements of MPI performance characteristics. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in PVM and MPI*, pages 11–18. Springer-Verlag, 1999.

[16] Optimized Collective Communication Library. http://www.cs.utk.edu/∼pjesa/projects/occ/. Accessed on March 2006.