

MPI Collective Algorithm Selection and Quadtree Encoding

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

*Innovative Computing Laboratory,
University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA*

Abstract

We explore the applicability of the quadtree encoding method to the run-time MPI collective algorithm selection problem. Measured algorithm performance data was used to construct quadtrees with different properties. The quality and performance of generated decision functions and in-memory decision systems was evaluated. Experimental data shows that in some cases, a decision function based on a quadtree structure with a mean depth of three, incurs on average as little as a 5% performance penalty. In all cases, experimental data can be fully represented with a quadtree containing a maximum of six levels. Our results indicate that quadtrees may be a feasible choice for both processing of the performance data and automatic decision function generation.

Key words: MPI collective operations; Performance optimization; Algorithm selection problem; Quadtree encoding; Performance evaluation

1 Introduction

The performance of MPI collective operations is crucial for good performance of MPI applications that use them [1]. Significant efforts have been spent on designing and implementing efficient collective algorithms both for homogeneous and heterogeneous cluster environments [2–9]. The performance of these algorithms depends on the total number of nodes involved in communication,

* Corresponding author. Tel.: +1-865-974-6722
Email address: pjesa@cs.utk.edu (Jelena Pješivac–Grbović).

system and network characteristics, size of the data being transferred, the current load on the network and, if applicable, the operation that is being performed as well as the segment size that is used for operation pipelining. Thus, selecting the best possible algorithm and segment size combination (*method*) for every instance of a collective operation is important.

To ensure good performance of MPI applications, collective operations can be tuned for a 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 a close-to-optimal method for a particular collective instance. This approach relies on the ability of the decision function to accurately predict the algorithm and segment size to be used for a particular collective instance. Alternatively, one could construct an in-memory decision system that could be queried/searched at run-time to provide the optimal method information. In order for either of these approaches to be feasible, the memory footprint and the computation time required to make the decision need to be minimal.

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

We developed a prototype quadtree implementation and the programs to analyze the experimental performance data, construct the quadtree decision functions, and determine their performance penalty in comparison to the exact decision function. We collected detailed profiles for two MPI collective algorithms (broadcast and reduce) on two clusters and analyzed the quality of the quadtree-based decisions functions built using this data 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.

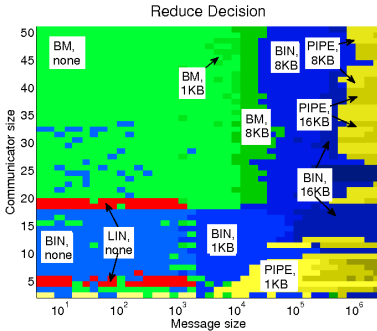
2 Related work

The MPI collective algorithm selection problem has been addressed in many MPI implementations. In FT-MPI [10], the decision function is generated manually using a visual inspection method augmented with Matlab scripts used for analysis of the experimentally collected performance data. This approach results in precise, albeit complex, decision functions. In MPICH-2[11], 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 Open MPI [12], the algorithm selection can be done via either a compiled decision function or user-specified parameters [13].

In addition, some of the high-performance networks support hardware or hardware-software hybrid collectives [14,15]. In such cases, the performance gain by using hardware based collectives can be orders of magnitude higher than using the most advanced software implementations based on point-to-point communication. The limitations of hardware-based collectives are that they often implement only a small subset of MPI collectives (such as Broadcast, Reduce, and Barrier) and are sometimes limited in their applicability (e.g., they can be used only on full partition of the system). In these cases, to achieve good performance one still needs to properly select either software- or hardware-based implementation when applicable.

Data mining techniques can also be applied to the algorithm selection problem. The problem is to classify the 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. Statistical learning methods have been applied to algorithm selection problems for matrix-matrix multiplication in a number of projects[16,17]. C4.5 decision trees have been applied to the MPI collective operation problem in [18].

Alternatively, one can interpret the information about 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 (a) illustrates a decision map for the reduce operation on the Frodo cluster at the University of Tennessee, Knoxville. The encoded structure can be used to generate either a decision function code or an in-memory decision structure that can be queried at run-time. To the best of our knowledge, we are the only group that has approached the MPI collective tuning process in this way.



(a)

Comm size	Msg size	Algorithm	Seg size	Method index
3	1	Linear	none	15
3	2	Linear	none	15
...
50	1MB	Pipeline	8KB	24
...

(b)

Fig. 1. (a) Reduce decision map from Frodo cluster. Different colors correspond to different method indexes. (b) An example of decision map in tabular form.

3 Quadrees 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 particular system. An example of a decision map is displayed in Figure 1 (b). The decision map that will be used to initialize the quadtree must be complete and a square matrix with a power of two dimension size ($2^k \times 2^k$). A 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 the efficiency of the encoding (both in positive and negative manner).

3.1 Quadtree decision structure and its properties

Once a decision map is available, we initialize the quadtree from it using the 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 a threshold or a maximum depth limit 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 in order to determine the method index for the specified communicator and message size. Setting the accuracy threshold helps smooth the experimental data, thus possibly making the decision function more resistant to anomalies in measurements. Applying

the maximum depth and/or the accuracy thresholds is equivalent to applying low-pass filters to the original data set.

Each of the internal nodes in a decision 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 that 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 regions into which the decision map is divided, and the internal nodes represent the rules of the corresponding decision function. As a consequence, quadtrees allow us to perform a recursive binary search in a two-dimensional space.

3.2 Generating decision function source code

We provide the functionality to generate decision function source code from the constructed 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}
```

where NW, NE, SW, and SE correspond to north-west, north-east, south-west, and south-east quadrants of the region, respectively.

The current implementation is functional but lacks some possible optimizations, such as the ability to merge conditions with the same color. The conditions for boundary points (minimum and maximum communicator and message sizes) are expanded to fully cover that region. For example, the decision for minimum communicator size will be applied to all communicator sizes smaller than minimum.

3.3 In-memory quadtree decision structure

An alternative to generating the decision function source code is maintaining an in-memory quadtree decision structure, which can be queried at run-time.

An optimized quadtree structure contains four pointers and one 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 36 bytes on 64-bit architectures.¹ In addition, the system needs to maintain an in-memory mapping of (algorithm,

¹ In this analysis, we ignore data alignment issues which could lead to even larger size of the structure.

segment size) pairs to method indexes, as well as, the communicator and message sizes used to construct the quadtree.

The maximum depth decision quadtree we encountered in our tests had six 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 around 192KB of memory. However, our results indicate that the quadtrees with three levels can still produce reasonably good decisions. A 3-level quadtree would occupy at most 3060 bytes and as such could fit into one 4KB page of main memory. As the decision function will be called occasionally (i.e., once for each tuple (*collective, message size*)), the in-memory quadtree will not be cached. Therefore, each invocation of the decision function is expected to generate a large number of cache misses.

Our prototype implementation provides tools for managing the in-memory decision functions - however the internal node structure occupies 48 bytes instead of the minimal 36 bytes. The associated structure for method mapping requires 6B per method, and the communicator and the message sizes are represented using integer arrays (2B per element). Querying the structure involves determining the indexes of specified communicator and message sizes. We believe that a fully-optimized version should achieve better performance than this 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 of quadtrees built from the same experimental data but using different constraints.

Under the assumption that the collective operation's parameters are uniformly distributed across communicator size and message size space, the mean depth of the quadtree corresponds to the mean number of conditions that need to be evaluated 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 the Frodo and Grig clusters located at the University of Tennessee, Knoxville. The Frodo cluster has 64 node, dual AMD OpteronTMprocessors at 1.4GHz with 2GB RAM and supports Fast Ethernet and Myrinet 2G interconnects. Measurements on Frodo in this paper were obtained using the MX library. The Grig cluster has 64 node, quad Intel[®]XeonTMprocessors at

3.20GHz with 4GB RAM and supports Fast Ethernet and Myrinet 2G interconnects. On Grig, the Fast Ethernet network was used for reported measurements.

The measurements on the Frodo cluster were collected using the Open MPI version 1.3 release candidate and the SKaMPI[19] benchmark, while the results from Grig were collected using MPICH-2 version 1.0.3 and the OCC [20] benchmark.²

4.1 Broadcast decision maps

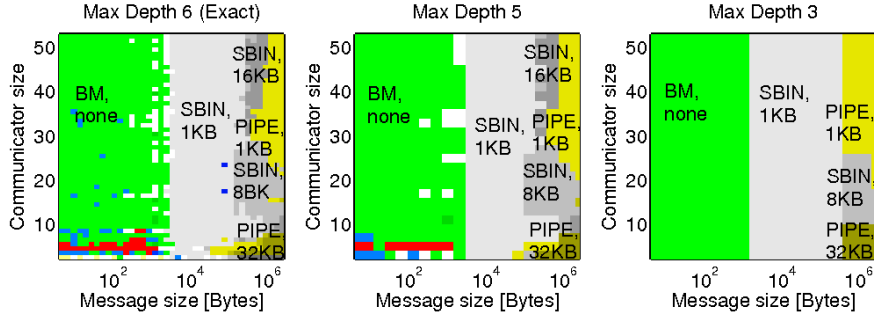
Figure 2 shows three different quadtree decision maps for a broadcast collective on the Frodo and Grig clusters, respectively. Different colors in the figures correspond to different method indexes. Both figures use the same color scheme. The trees were generated by limiting the maximum tree depth. The x-axis scale is logarithmic. The crossover line in these figures is not in the middle due to the “fill-in” points used to adjust the original size of the decision map from 49×38 and 25×48 , respectively, to 64×64 form.

We considered five different broadcast algorithms (Linear, Binary, Binomial, Split-Binary, and Pipeline),³ and seven different segment sizes (no segmentation, 1KB, 8KB, 16KB, 32KB, 64KB, and 128KB). The measurements on the Frodo cluster, Figure 2a, covered all communicator sizes between two and 50 processes and message sizes in the 4B to 2MB range. On Grig, Figure 2b, results cover communicator sizes two to 28 and message sizes in the 1B to 384KB range.

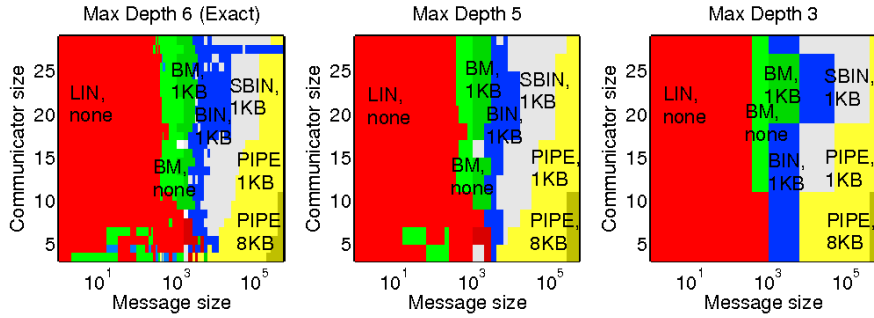
The exact decision maps for the Frodo and Grig clusters in Figure 2 are quite different. Using the Myrinet interconnect (Figure 2a), benchmarks achieve best performance using binomial tree and larger segment sizes, while over Fast Ethernet (Figure 2b), a combination of a linear algorithm and methods with 1KB segments is a better choice. Exact decisions exhibit trends, however in both cases, there are regions with a high information density. Limiting the maximum tree depth smoothes the decision map and subsequently decreases the size of the quadtree. Tables 1 and 2 show the mean tree depth and related statistics for these decision maps.

² The high-resolution version of Figures in this Section can be found either at http://www.cs.utk.edu/~pjesa/Papers/pcse_2007.html or by contacting the authors.

³ For more details on these algorithms, refer to [21].



(a)



(b)

Fig. 2. Maximum-depth limited broadcast decision maps from the (a) Frodo and (b) Grig clusters. In these Figures, “LIN” stands for “Linear”, “BM” for “Binomial”, “BIN” for “Binary”, “SBIN” for “Split-Binary”, and “PIPE” for “Pipeline” algorithm, while “none”, “1KB”, “8KB”, and “32KB” are corresponding segment sizes.

4.2 Performance penalty of decision quadtrees

One possible metric of merit is the performance penalty one would incur by using a restricted quadtree instead of the exact one. To compute the penalty for a particular communicator and message size tuple, one can compare the performance of the method suggested by the restricted tree with the performance of the method suggested by the exact tree.

The reproducibility of the measured results is not within the scope of this paper, but both of the benchmarks we utilized follow the guidelines from [22] 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 relative performance penalty of the decision quadtrees

from Figure 2. The colorbar represents the relative performance penalty in percentage: white means less than 5%, yellow is between 10% and 25%, red is 50% and above. Tables 1 and 2 summarize the properties and performance penalties for the same data.

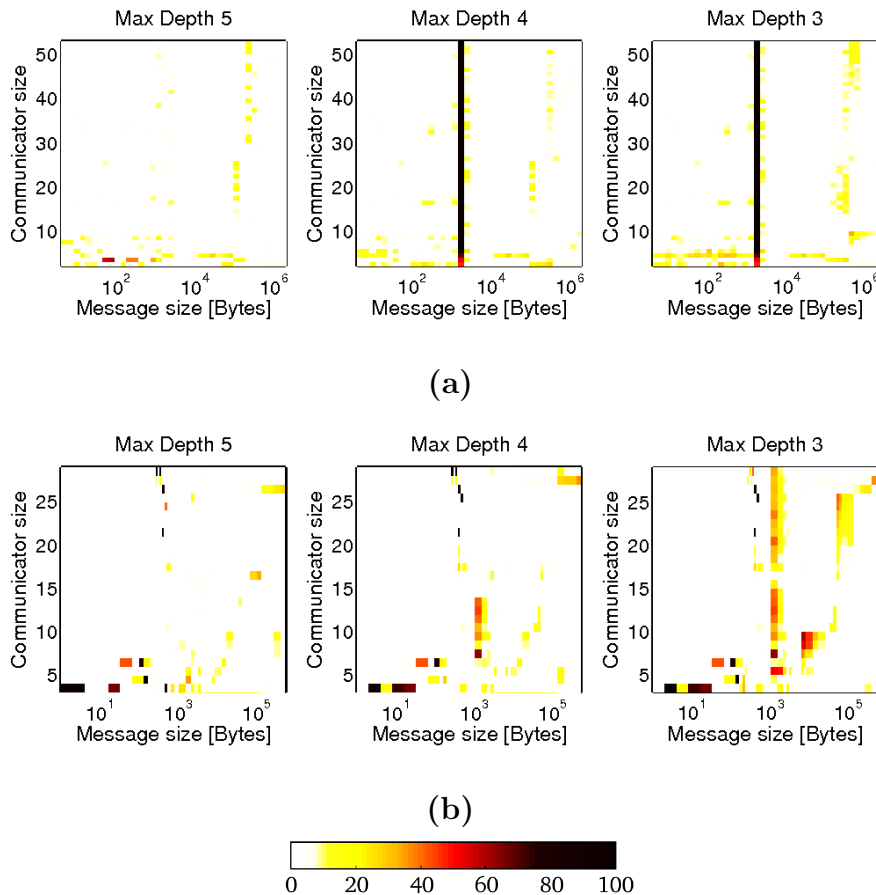


Fig. 3. Performance penalty of maximum-depth limited broadcast decision function from (a) Frodo and (b) Grig (Figure 2).

Tree Depth			Performance Penalty [%]				Number of	
Max	Min	Mean	Min	Max	Mean	Median	Leaves	Nodes
1	1	1.00	0.00	560.42	74.62	2.88	4	5
2	1	1.92	0.00	743.16	12.56	0.00	13	17
3	1	2.50	0.00	743.15	12.43	0.00	22	29
4	2	3.63	0.00	743.15	12.01	0.00	91	121
5	2	4.61	0.00	31.14	0.67	0.00	328	437
6	2	5.65	0.00	0.00	00.00	0.00	1153	1537

Table 1

Complete statistics for maximum-depth limited broadcast decision quadtrees on Frodo (Figure 2a).

The results on Grig (Figures 2b, 3b, and Table 2) show that a 3-level quadtree would have less than a 6% mean performance penalty. On Frodo (Figures 2a,

Tree Depth			Performance Penalty [%]				Number of	
Max	Min	Mean	Min	Max	Mean	Median	Leaves	Nodes
1	1	1.00	0.00	337.43	37.10	0.00	4	5
2	2	2.00	0.00	391.53	18.54	0.00	16	21
3	2	2.76	0.00	247.20	05.75	0.00	37	49
4	2	3.75	0.00	247.20	03.25	0.00	106	141
5	2	4.61	0.00	225.30	01.29	0.00	227	369
6	2	5.70	0.00	0.00	00.00	0.00	1024	1365

Table 2

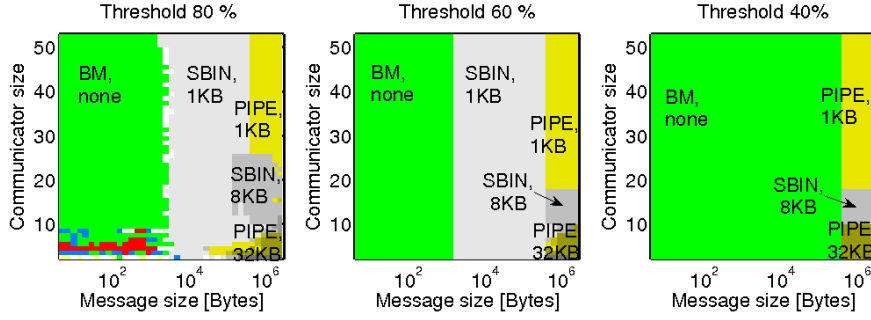
Complete statistics for maximum-depth limited broadcast decision quadtrees on Grig (Figure 2b).

3a, and Table 1) 2-, 3-, and 4-level quadtrees have very similar performance penalty statistics, primarily due to the high performance penalty for a 1448B message on all communicator sizes greater than three. For this message size, the restricted trees use the Split-Binary algorithm with 1KB segments, instead of Binomial with no segmentation. The experimental data reveals a spike for the Split-Binary with 1KB segments for 1448B message size on all communicator sizes: measured time jumped to 300+ μs in comparison to 64+ μs for 1024B and 65+ μs for 2048B message. Moreover, all points with more than 40% performance penalty on 2-, 3-, and 4-level quadtrees were the ones for 1448B message. If we remove 1448B data points, the mean performance penalty for a 3-level tree on Frodo drops to 1.7%.

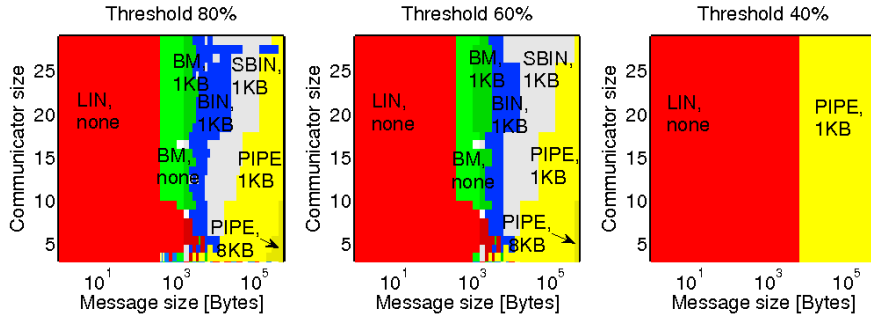
4.3 Quadtree accuracy threshold

In Section 3.2 we mentioned that an alternative way to limit the size of a quadtree is to specify the tree accuracy threshold. Figure 4 shows accuracy-threshold limited, broadcast decision maps on the Frodo and Grig clusters. The data in these figures corresponds to the data in Figure 2.

Figure 5 shows the effect of varying the accuracy threshold on the mean quadtree depth and mean performance penalty of broadcast and reduce quadtree decision functions on the Frodo and Grig clusters. In all cases, the mean quadtree depth flattens out once a high enough accuracy threshold is achieved (from 45% for Reduce on Frodo to almost 70% for Reduce on Grig). Based on the mean performance penalty data, the trees generated with accuracy-threshold values higher than this limit are very similar to the exact tree.



(a)



(b)

Fig. 4. Accuracy-threshold limited broadcast decision maps from the (a) Frodo and (b) Grig clusters. In these images, the abbreviations are identical to the ones in Figure 2.

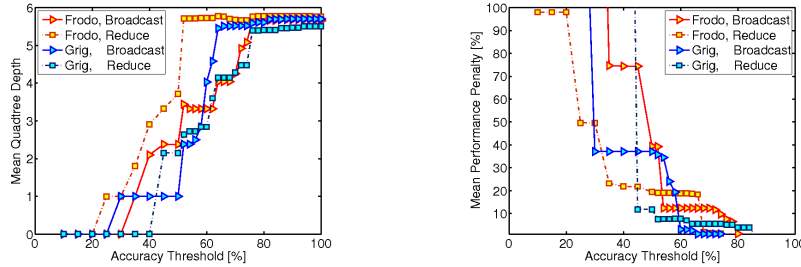


Fig. 5. Effect of the accuracy threshold on mean quadtree depth and performance penalty.

4.4 Accuracy-threshold vs. Maximum-depth constrained trees

One of the objectives of this study is to determine which of the two methods for restricting the quadtree size gives higher-quality decision functions. Figure 6 shows the mean performance penalty of broadcast and reduce decisions as a function of the mean depth of the accuracy-threshold and maximum-depth constrained trees.

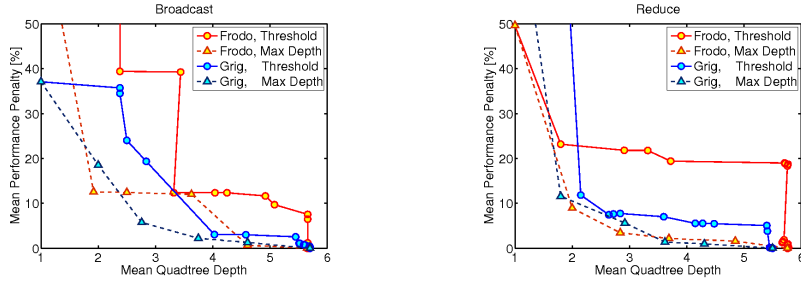


Fig. 6. Accuracy-threshold vs. maximum-depth quadtree construction for broadcast and reduce collectives on Frodo and Grig cluster. Corresponds to data in Figures 2, 3, 4, 5, and Tables 1, 2.

The results indicate that maximum-depth limited quadtrees achieve a lower mean performance penalty than similar accuracy-threshold limited trees. Moreover, for results with experimental data, which did not exhibit unexpected spikes (both collectives on Grig, and reduce on Frodo), maximum-depth limited trees improve their accuracy with each additional level. This is not the case for accuracy-threshold limited trees: we can see a range of mean quadtree depths corresponding to the same mean performance penalty.

4.5 In-memory quadtree-based decision system

Table 3 contains the performance results for an in-memory quadtree-based decision system for reduce collective on the Grig cluster. The reported *decision time* value is the average time it took for the quadtree to make decisions for a random communicator and message size from the 2 – 64 and 1 – 16MB ranges, respectively. All measurements were taken with the same seed value. The measurements were taken on an AMD Athlon™64 Processor 3500+ at 2.2GHz with 512KB cache and 1GB RAM.

In all cases, mean-time to decision of an in-memory 3-level quadtree decision system was between $50ns$ and $75ns$. The decision time for the corresponding compiled decision function was between $13ns$ and $19ns$.

In comparison, Open MPI broadcast and reduce decision functions took $18.90ns$ and $21.14ns$, respectively. The computed mean performance penalty of the Open MPI decision functions computed against the data collected on the Frodo cluster was below 20%, with a median value around 5%. This is similar to or higher than the mean performance penalty of the 3-level quadtree decision function on both systems.

Mean tree depth	Mean performance penalty	In-memory quadtree			Decision function	
		Memory size [Bytes]	Decision time [ns]	Decision time per level [ns]	Size	Decision time [ns]
1.00	81.10	712	35.61	17.81	4	11.68
1.80	11.58	1096	36.78	13.14	12	10.70
2.92	5.63	3784	54.53	13.91	68	13.68
3.62	1.39	7048	54.70	11.84	136	13.87
4.31	0.97	13192	55.02	10.36	264	13.88
5.50	0.00	42952	71.54	11.01	884	16.47

Table 3

Performance of the prototype implementation of an in-memory, quadtree-based decision system, and the corresponding decision function for the reduce collective on Grig. Time per level was computed as $\frac{\text{decision time}}{\text{mean depth}+1}$. Memory size includes size of the quadtree, space for communicator and message sizes, and method map. Function size is the total number of *if* and *else if* statements in the decision function source code.

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 at 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 three, incurs less than a 5% performance penalty on the average. In other cases, deeper trees (five or six levels) were necessary to achieve the same performance. However, in all considered cases, a 3-level quadtree would incur at most 12% 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 a prototype quadtree implementation and as such should be interpreted as a worst case scenario. The results with this system show that in less than $75ns$ on average, we can get the optimal decision based on the experimental data. Not surprisingly, the fixed decision function in Open MPI outperformed the in-memory system in terms of time-to-decision performance. However, the compiled version of the 3-level quadtree decision function had comparable performance and higher accuracy than the default Open MPI decision function.

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 that are optimal for single communicator values, e.g. communicator sizes that are power of two. The same problem is exacerbated if the performance measurement data used to construct trees is too sparse. The sparse data set and single line rules are high-frequency information and applying low-pass filters to it can cause loss of important information.

In addition, quadtree encoding can only capture 2-D performance data. While octrees can be used to increase the number of parameters to three, extending this approach further is not necessarily feasible. However, as most current MPI implementations do not make additional run-time information (such as processor load or network utilization) globally available, limiting the input parameter space to two or possibly three dimensions may not be a real restriction.

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.

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 the right structure for this type of approach, but we plan to investigate this approach in the near future.

Acknowledgement

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] R. 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, 1999, pp. 77–85.

- [2] J. Worringen, Pipelining and overlapping for MPI collective operations, in: IEEE Conference on Local Computer Network, IEEE Computer Society, Boon/Königswinter, Germany, 2003, pp. 548–557.
- [3] L. P. Huse, Collective communication on dedicated clusters of workstations, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, 1999, pp. 469–476.
- [4] R. Rabenseifner, J. L. Träff, More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, no. 3241 in LNCS, Springer-Verlag, 2004.
- [5] S. Juhász, F. Kovács, Asynchronous distributed broadcasting in cluster environment., in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, no. 3241 in LNCS, 2004, pp. 164–172.
- [6] E. W. Chan, M. F. Heimlich, A. Purkayastha, R. M. van de Geijn, On optimizing of collective communication, in: Cluster, 2004.
- [7] R. Thakur, W. Gropp, Improving the performance of collective operations in MPICH, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, no. 2840 in LNCS, Springer Verlag, 2003, pp. 257–267.
- [8] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, R. A. F. Bhoedjang, MagPIe: MPI’s collective communication operations for clustered wide area systems, in: Proceedings of ACM SIGPLAN Symposium, ACM Press, 1999, pp. 131–140.
- [9] M. Bernaschi, G. Iannello, M. Lauria, Efficient implementation of reduce-scatter in MPI, Journal of Systems Architecture 49 (3) (2003) 89–108.
- [10] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pješivac-Grbović, K. London, J. Dongarra, Extending the MPI specification for process fault tolerance on high performance computing systems, in: Proceedings of the International Supercomputer Conference (ISC), Primeur, 2004.
- [11] MPICH - A portable implementation of MPI,
<http://www-unix.mcs.anl.gov/mpi/mpich2/>.
- [12] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Budapest, Hungary, 2004, pp. 97–104.
- [13] G. Fagg, G. Bosilca, J. Pješivac-Grbović, T. Angskun, J. Dongarra, Tuned: A flexible high performance collective communication component developed for open mpi, in: Proceedings of Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS), Innsbruck, Austria, 2006.

- [14] G. Almasi, C. Archer, J. Castanos, J. Gunnels, C. Erway, P. Heidelberger, X. Martorell, J. Moreira, K. Pinnow, J. Ratterman, Design and implementation of message-passing services for the Blue Gene/L supercomputer, *IBM Journal of Research and Development* 49 (2) (2005) 393–406.
- [15] F. Petrini, S. Coll, E. Frachtenberg, A. Hoisie, Hardware- and software-based collective communication on the quadrics network, *IEEE International Symposium on Network Computing and Applications* (2001) 00–24.
- [16] V. Eijkhout, E. Fuentes, T. Edison, J. J. Dongarra, The component structure of a self-adapting numerical software system, *International Journal of Parallel Programming* 33 (2).
- [17] R. Vuduc, J. W. Demmel, J. A. Bilmes, Statistical Models for Empirical Search-Based Performance Tuning, *International Journal of High Performance Computing Applications* 18 (1) (2004) 65–94.
- [18] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, J. J. Dongarra, Decision trees and MPI collective algorithm selection problem, Tech. Rep. UT-CS-06-586, The University of Tennessee at Knoxville, Computer Science Department, <http://www.cs.utk.edu/~library/2006.html> (2006).
- [19] SKaMPI: Special Karlsruher MPI Benchmark, <http://liinwww.ira.uka.de/~skampi/>.
- [20] OCC - Optimized Collective Communication Library, <http://www.cs.utk.edu/~pjesa/projects/occ/>.
- [21] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, J. J. Dongarra, Performance analysis of MPI collective operations, in: *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15*, IEEE Computer Society, Washington, DC, USA, 2005, p. 272.1.
- [22] W. Gropp, E. L. Lusk, Reproducible measurements of MPI performance characteristics, in: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag, 1999, pp. 11–18.