An Adaptive, Scalable, and Portable Technique for Speeding Up MPI-Based Applications

Rosa Filgueira[1], Malcolm Atkinson[1], Alberto Nuñez[2], and Javier Fernández[3]

[1] University of Edinburgh, School of Informatics, Edinburgh EH8 9AB, U.K.
{rosa.filgueira,mpa}@ed.ac.uk
[2] University Complutense de Madrid, Dept. Sistemas Informáticos y Computación, 28040 Madrid, Spain
alberto.nunez@pd.ucm.es
[3] University Carlos III de Madrid, Dept. Arquitectura de Computadores, 30 28911 Leganés, Spain
jfernand@arcos.inf.uc3m.es

Abstract. This paper presents a portable optimization for MPI communications, called *PRAcTICaL-MPI* (Portable Adaptive Compression Library- MPI). *PRAcTICaL-MPI* reduces the data volume exchanged among processes by using lossless compression and offers two main advantages. Firstly, it is independent of the MPI implementation and the application used. Secondly, it allows for turning the compression on and off and selecting the most appropriate compression algorithm at runtime, depending on the characteristics of each message and on network performance.

We have validated *PRAcTICaL-MPI* in different MPI implementations and HPC clusters. The evaluation shows that compressing MPI messages with the best algorithm and only when it is worthwhile, we obtain a great reduction in the overall execution time for many of the scenarios considered.

Keywords: MPI Library, Parallel techniques, High-Performance Computing, Compression algorithms, Adaptive systems, Portable optimizations.

1 Introduction

Parallel computation on cluster architectures has become the most common solution for developing High-Performance Computing applications. The Message Passing Interface (MPI) standard [1] is one of the most commonly used communication middleware frameworks on clusters.Several implementations of MPI are available, like MPICH [2], XT-MPI, OPENMPI [3], and LAM [4].

The current trend in High-Performance Computing is to use multicore clusters in order to increase computation capability, thus allowing an increase in the number of processes per application. Despite the fact that networks used in multicore clusters are fast and have low latency, the number of transferred messages may cause a bottleneck in the communication system, as communication-intensive,

C. Kaklamanis et al. (Eds.): Euro-Par 2012, LNCS 7484, pp. 729–740, 2012.

parallel MPI applications spend a significant amount of their total execution time exchanging messages between processes. This problem may lead to poor performance and scalability in many cases.

In this paper, we present a portable optimization of MPI called *PRAcTICaL-MPI* (Portable Adaptive Compression Library-MPI), which is fully transparent both to applications and MPI implementations. The main goal of *PRAcTICaL-MPI* is to enhance the performance and scalability of MPI-based applications and to reduce the volume of communications by applying run-time lossless compression in a transparent way for applications and MPI implementations. *PRAcTICaL-MPI* is capable of using the following compression algorithms: RLE [5], Huffman [6], Rice [7], FPC [8], and LZO [9]. Furthermore, the technique presented applies the Run-time Adaptive Strategy (RAS) developed in [10] to select the most appropriate compression algorithm to be used dynamically for each message exchange, and the size threshold form which a benefit is achieved by using data compression.

We have implemented *PRAcTICaL-MPI* by using the standard MPI profiling interface (PMPI) with the lowest possible overhead. The major contributions of *PRAcTICaL-MPI* can be summarised by looking the following properties:

- Transparency: *PRAcTICaL-MPI* uses the standard MPI profiling interface (PMPI), allowing transparent data compression for different applications and MPI implementations.
- Portability: *PRAcTICal-MPI* can be run by any MPI implementation that supports PMP, and is hence fully portable.
- Scalability: Since *PRAcTICaL-MPI* applies run-time compression to reduce the volume of messages transferred, the execution time of the application is reduced, thus enhancing the performance and scalability of MPI-based applications.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 summarises our Run-time-Adaptive Strategy. Section 4 introduces the *PRAcTICaL-MPI* architecture in detail. Section 5 presents an extensive evaluation of *PRAcTICaL-MPI*I in several scenarios. Finally, Section 6 presents conclusions and a discussion of potential future work.

2 Related Work

Two main background techniques and existing contributions are reviewed in this section: The standard MPI profiling interface, and the most popular works to extend MPI with compression capabilities.

2.1 PMPI: Standard MPI Profiling Interface

The MPI Forum defined [1] the MPI profiling interface (PMPI) as a mechanism for application developers to obtain high-level performance information about the behavior of both the application algorithm and the parallel system. Note that PMPI is part of the MPI standard specification.

The main concern of PMPI [11] is to provide a mechanism by which the developers of profiling (and other) tools can collect performance information they require without access to the underlying implementation. The mechanism is based on each MPI-routine having a corresponding PMPI-routine with identical syntax and functionality, so that it can be used to intercept all MPI calls and change their functionality. Therefore, tools can create wrappers for any MPI_routine and then insert them "between" the MPI library and the application. This is a powerful feature that is exploited in many applications and tools, such as the performance visualization tool Jumpshot [12]. Note that one of the major features of PMPI is that it allows selective replacement of MPI routines at link time without the need to re-compile or re-link the MPI implementation.

2.2 Adding Compression to MPI

The use of compression within MPI is not new, although it has been only used in specific ways for very few special cases. Major examples of such approaches include cMPI, PACX-MPI, COMPASSION, MiMPI, CoMPI, Adaptive-CoMPI.

PACX-MPI (PArallel Computer eXtension to MPI) [13,14] is an on-going project of the HLRS, Stuttgart. It enables an MPI application to run on a meta-computer consisting of several, possibly heterogeneous machines, each of which may itself be massively parallel. Compression is used for TCP message exchange among different systems in order to increase bandwidth, but a fixed compression algorithm is used and compression is not used for messages within single sub-system. *cMPI* [15,16] has similar goals o those of *PACX-MPI*, namely to enhance the performance of inter-cluster communication with a software-based data compression layer. Compression is added to all communication, so it does not offer any flexibility as to how to configure when and how to use compression.

COMPASSION [17] is a parallel I/O run-time system which includes chunking and compression for irregular applications. The LZO algorithm is used for fast compression and decompression, but again it is only used for the I/O part of irregular application.

MiMPI [18] is a prototype of a multithread implementation of MPI with thread-safe semantics that adds run-time compression of messages sent among nodes. Although the compression algorithm can be changed (providing more flexibility), the use of compression is global for all processes pertaining to an MPI application.

CoMPI [19] was the first work in which a compression library was fully integrated into MPICH. *CoMPI* is based on run-time compression of the MPI messages exchanged among applications. The user can choose the compression algorithm from a pool of algorithms, and all the communications will be compressed with the same algorithm. The problem with this approach is that the user can not always select the most suitable compression algorithm, and compression is always turned on by default.

Adaptive-CoMPI [10] allows for turning the compression on and off. It also selects the most appropriate compression algorithm at run-time. Although

Adaptive-CoMPI is independent of the application, it is dependent to the MPI implementation.

3 The Run-time-Adaptive Strategy

For the *Adaptive-CoMPI* technique [10] we developed two strategies, the *Run Time Strategy* (RAS) and the *Guided Strategy* (GS), to decide whether to apply compression or not on a message-by-message basis, as well as to decide which compression algorithm should be applied. The GS strategy makes these decisions by analysing the structure of the messages off-line. Once this selection process has been completed, the decisions are applied to the next executions of the same application with the same input parameters. In contrast to this, the RAS strategy makes these decisions at run-time, while the application is being executed. Because the GS strategy is not completely independent of the application, the RAS strategy has been chosen to be implemented in *PRAcTICaL-MPI*. With this in mind, we describe the main features of the RAS strategy in more detail.

As we explained RAS decides at run-time per message whether to compress a message before sending it or not, and which compression algorithm to apply. To make these two decisions, there are some cases in which RAS has to estimate the speedup, we will describe which ones these are late. To calculate this speedup, some network and compression information is needed. In order to provide RAS with this information, we have developed two modules:

- The Network Behavior module estimates the latency and bandwidth in order to predict the time needed to send a message, generating a network-behavior heuristics file for each installation.
- The Compression Behavior module selects the best compression algorithm depending on the message datatype and its redundancy level. Also, this model estimates the time needed to compress and decompress a message with different compression algorithms. Furthermore, it generates a compression-behavior heuristics file for each installation. This file is used to decide which algorithm to choose in order to compress a message depending on the message features.

These two modules have to be generated once per cluster, in order to obtain the heuristics files. Furthermore, the Network Behavior model also needs to be updated when there is a change in the network topology, to capture the new situation.

RAS uses length and datatype of the message, and the location of the processes to decide whether to compress or not and which compression algorithm to apply. RAS deactivates the compression when the processes involved in the communication are located in the same node. In other cases, when the processes are located in different nodes, RAS distinguishes between four kinds of datatype: Integer, floating-point, double precision floating-point, and "others" datatypes. The strategy analyses the four kinds of datatype separately and makes different decisions for each datatype. To choose the most appropriate algorithm

for each datatype, RAS consults the compression-behavior heuristics taking the message features into account. Moreover, it builds a compression window for each datatype, with two adaptive thresholds that state from which minimum size to which maximum size a benefit can be achieved by compressing the data as shown in figure 1. Thus, RAS only estimates the speedup to send a message compressed when the size of the message is between both thresholds. To calculate the speedup, the information provided by the network-behavior and compression-behavior heuristics are used.

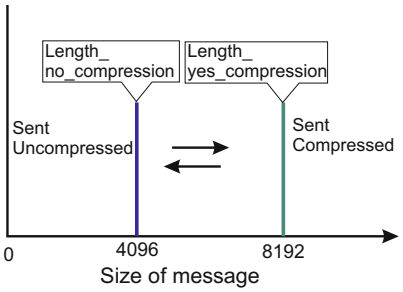

Fig. 1. Window compression

4 PRAcTICaL-MPI

The *PRAcTICaL-MPI* technique, is an optimization of MPI communications that exploits the MPI profiling interface (PMPI) to apply run-time lossless compression (and decompression), thus reducing the volume of communications. As Figure 2 shows, PMPI intercepts the MPI calls and wraps the *PRAcTICaL-MPI* technique around the actual MPI library invocation. *PRAcTICaL-MPI* is portable in the sense that it can be used with any MPI implementation, not just a with a specific MPI implementation. Besides, *PRAcTICaL-MPI* is transparent both to applications and MPI implementations, because it can be applied without changing their source code in any way.

We have built a library called *Practical*, where the most common routines of point-to-point and collective communications are wrapped inside a *PRAcTICaL-MPI* layer : MPI_Send, MPI_Isend, MPI_Bcast, MPI_Recv, MPI_Irecv, MPI_Wait, MPI_Waitall, MPI_Scatter, MPI_Gather. If we want to apply *PRAcTICaL-MPI* to another MPI communication, we only have to add a new wrapper to the respective routine in *Practical* library. The only requirement that *PRAcTICaL-MPI* makes is that the user needs to relink their applications with the *Practical* library to include our adaptive compression functionality.

Different compression algorithms are used depending on the specific characteristics of each communication. All compression algorithms have been included in a single library called Compression-Library. To include more compression

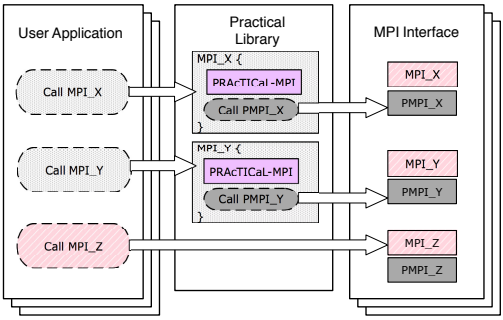

Fig. 2. *PRAcTICaL-MPI* architecture

algorithms, we only have to replace this library with a new version. Therefore, *PRAcTICaL-MPI* can be easily updated to include new compression algorithms. Currently, the compression library includes: RLE, Huffman, Rice8, Rice16, Rice32, rice8s, rice16s, rice32s, LZ, LZ77, LZ_f LZ77 Fast, Shannon-Fano, LZO, and FPC.

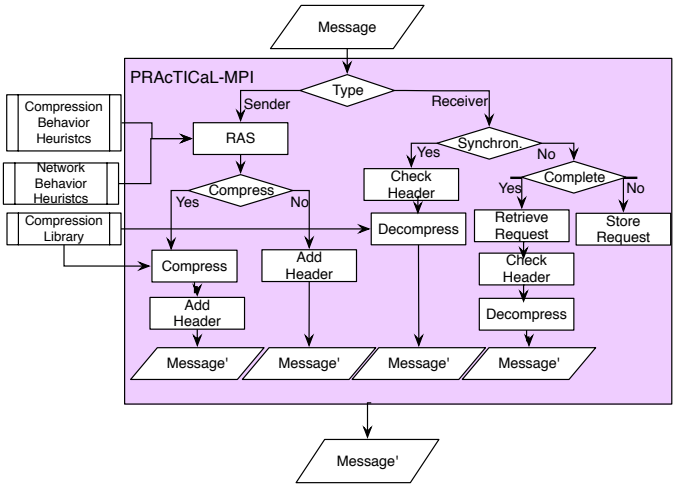

Fig. 3. *PRAcTICaL-MPI* schema

Figure 3 shows the internal workings of *PRAcTICaL-MPI* in more detail. The first step of the process is to identify which kind of operation has to be performed by the process that executes the MPI routine. If the process has to send data to other process, it is classified as a "sender". Otherwise, it is classified as a "receiver". For example, all the processes that execute a MPI_Send routine have to send data, so all these processes are classified as "senders". In the case of the MPI_Bcast routine, only the root process has to send data, and the others have to receive data. Therefore, only the root process is classified as "sender",

and the rest of processes as "receivers". The reason for this classification is that *PRAcTICaL-MPI* takes different actions in each case. In the case of the "sender" type, *PRAcTICaL-MPI* tries to compress the message with the best algorithm possible. In the case of a "receive" operation, *PRAcTICaL-MPI* decompresses the message in case it was sent compressed.

More specifically, the actions performed by *PRAcTICaL-MPI* can be described as follows:

- Send Actions: Firstly, *PRAcTICaL-MPI* applies the RAS strategy to select the appropriate compression algorithm for the message depending on the location of the node and its datatype. Note that if the two processes involved in the communication are located in the same node, the message is sent without compression. Secondly, RAS compares the size of the message with the two adaptive thresholds corresponding to the datatype of the message. As a result of this operation, the decision to compress the message or not is taken. Thirdly, in case RAS decides to compres the data, the data is compressed and also the size of the compressed message is checked. If the size of the compressed data is larger than the original data, the original message is sent without compression. Otherwise, it is sent compressed. Finally, the method adds a header to the message in order to notify the receiver whether the message has to be decompressed and which decompression algorithm has to be used after receiving it.
- Receive Actions: The decompression operation is performed in two different places depending on whether message passing is synchronous or asynchronous. For asynchronous communication, such as MPI_Irecv, the decompression is performed only after message transfer is complete. Therefore, for the asynchronous receive routines, *PRAcTICaL-MPI* only stores the request pointer of the operation in a global table. Once reception has been completed, probably during the execution of MPI_Wait or MPI_Waitall, the request pointer is retrieved from the global table, and after this the decompression is performed. On the other hand, for synchronous communication, message decompression is performed when the receiver has received the complete message. To decompress a message, *PRAcTICaL-MPI* checks the header of the message in order to know whether the message has to be decompressed and which algorithm has to be employed. Finally, it applies the decompression algorithm indicated by the sender.

The ways in which the *PRAcTICaL-MPI* technique is applied depend on the characteristics of each routine. For example, in case of MPI_Send, first *PRAcTICaL-MPI* is applied to compress the data, and PMPI_Send is called after that. On the other hand, for MPI_Recv, the data is received with the PMPI_Recv routine first, and *PRAcTICaL-MPI* is applied to decompress the data after that.

5 Evaluation

We evaluate our approach using the BIPS3D application with different input meshes representing different semiconductor devices. We compare the performance

of *PRAcTICaL-MPI* with the MPICH2.3 and XT-MPI distributions. The experiments were conducted using two different High-Performance Clusters called HECToR and EDDIE. We start with an overview of the BIPS3D application in section 5.1. Section 5.2 describes the HPC clusters used in our evaluation. The evaluation results themselves are presented in section 5.3.

5.1 The BIPS3D Application

BIPS3D is a 3-dimensional simulator of BJT and HBT bipolar devices described in [20]. The goal of the 3D simulation is to relate electrical characteristics of the device to its physical and geometrical parameters. The basic equations to be solved are Poisson equations and models describing electron and hole continuity in a stationary state.

Finite element methods are applied in order to discretize the Poisson equation, hole and electron continuity equations by using tetrahedral elements. The result is an unstructured mesh. In this work, we have used three different meshes, as described later.

Using the METIS library [21], the meshes are divided into sub-domains, in such a manner that one sub-domain corresponds to one process. The next step is decoupling the Poisson equation from the hole and electron continuity equations. They are linearized using the Newton method. Then we construct the part corresponding to the associated linear system for each sub-domain in a parallel manner. Each system is solved using domain decomposition methods. Finally, the results are written to a file.

For our evaluation BIPS3D has been executed using three different meshes: mesh1 (47200 nodes), mesh2 (732563 nodes) and mesh3 (289648 nodes). BIPS3D associates a data structure with each node of a mesh. The contents of these data structures constitute the data written to disk during the I/O phase. The number of elements that this structure has for each mesh entry is given by the *load* parameter. This means that, given a mesh and a load, the amount of data written to file is calculated as the product of the number of mesh elements and the load. In this work, we have evaluated our method using two different loads, 100 and 500.

5.2 HPC Clusters and MPI Implementations

We have performed our experiments on two different High-Performance Clusters in order to demonstrate how PRAcTICal-MPI adapts adapts itself to each architecture. In each cluster, a different MPI implementation is used. The main features of the clusters and MPI implementations used for our evaluation are:

1. HECToR is a Cray XT6 machine with contains 1856 nodes. Each node consists of two 12 core 2.1 GHz AMD opteron processors with 32 Gbytes of memory. The network used is Gemini interconnection. The MPI implementation used to perform our evaluation in this architecture is XT MPI 3.0.

2. EDDIE consists of 130 IBM dx360M2 iDataPlex servers with two Intel Westmere E5620 quad core processors and 24 GB of RAM, all connected through Gigabit ethernet. MPICH2.3 is the MPI implementation used for our experiments on EDDIE. We chose this implementation as it is one of the most popular MPI implementations.

5.3 Evaluation Results

We studied the performance of *PRAcTICaL-MPI* technique using the BIPS3D application and two different clusters, HECToR and EDDIE. Figures 4 and 5 show the overall speedup achieved using *PRAcTICaL-MPI* for $mesh1$, $mesh2$, $mesh3$ with two loads 100 and 500, and with 8, 16, 32, 64 and 128 processes, respectively.

Each speedup shown in these diagrams is calculated by comparing the original MPI implementation (MPICH2.3 in Figure 4 and XT MPI 3.0 in Figure 5) with the same MPI implementation wrapped with *PRAcTICaL-MPI*. Then, equation 1 is applied to these values. Values greater than one imply a reduction of the overall execution time using *PRAcTICaL-MPI*.

$$Speedup = \frac{Execution\_time\_MPI\_Implementation}{Execution\_time\_MPI\_Implementation\_with\_PRAcTICaL} \quad (1)$$

In general, the speedups achieved in 90% of the scenarios showed in Figures 4 and 5 are greater than or equal to one. These results are due to *PRAcTICaL-MPI* applying run-time compression to reduce the volume of the messages with the best algorithm per message, thus reducing execution time. Moreover, it deactivates the compression when it is not worth while applying any compression. The original MPI distribution performs better only in 10% of all cases, but even in those cases, the loss is nearly one in all of them.

The difference between the speedups achieved in the two scenarios is due to the cluster architecture, i.e. network speed and number of cores per node. On one hand, the EDDIE cluster (Figure 4) uses a Gigabit ethernet. This network is slower than the Gemini network, used in HECToR (Figure 5). Due to the fact that the network in HECToR is very fast, the compression is deactivated more often, because is less worthwhile sending the message compressed and decompressing it later than sending the message without compression. This behavior can be observed in Figure 5(a) for a load of 500 and 8, 16, and 32 processes. In these cases, the speedup is nearly one, because the compression is deactivated. On the other hand, the cluster architecture affects also the results, too. EDDIE has 8 cores per node, and HECToR has 12 cores per node. When the 12-core architecture is used, the number of processes in the same node increases, and therefore the number of communications between different nodes is lower than in the 8-core architecture. This means that in HECToR, compression is deactivated more times than in EDDIE. Therefore, we can observe how *PRAcTICaL-MPI* is able to adapt to different architectures at run-time.

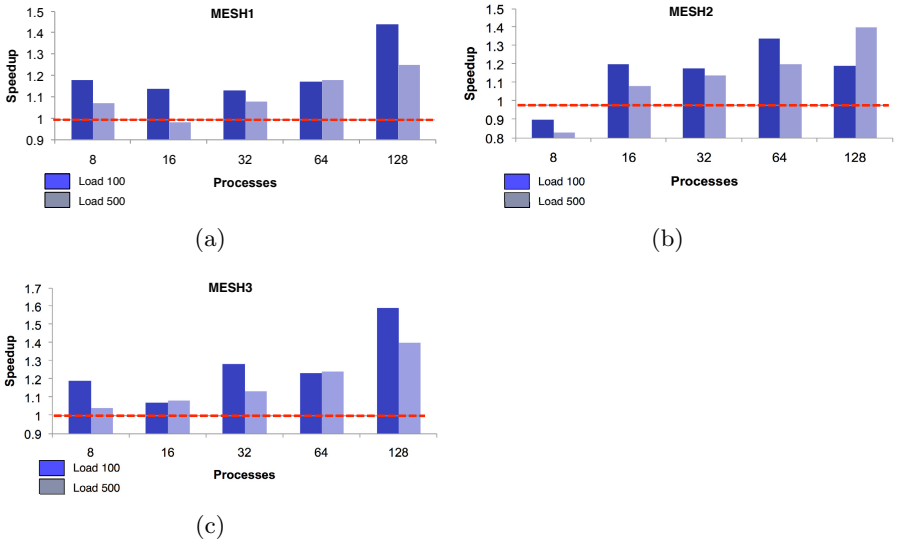

Fig. 4. Execution time improvement of BIPS3D on the EDDIE cluster: (a) Mesh1 (b) Mesh2 (c) Mesh3

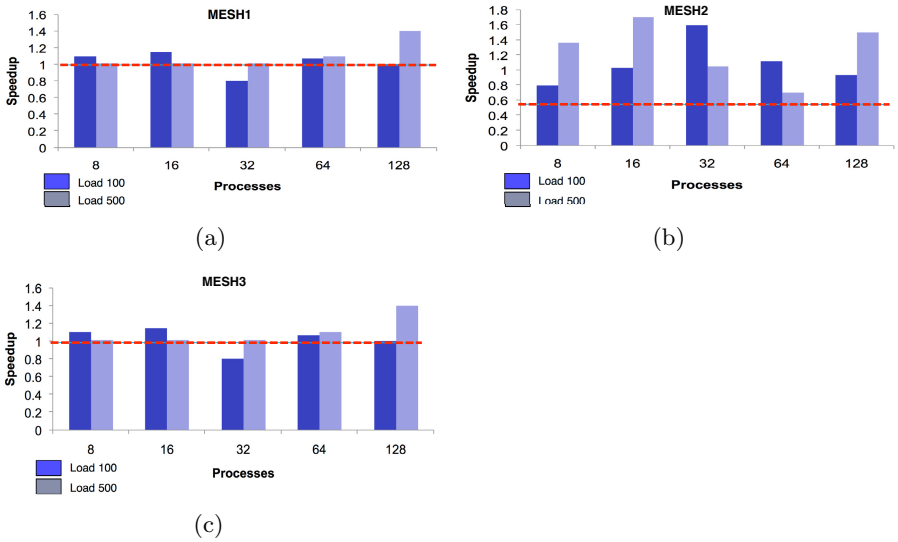

Fig. 5. Execution time improvement of BIPS3D in HECToR cluster: (a) Mesh1 (b) Mesh2 (c) Mesh3

Finally, we can notice that, the greater the number of processes, the bigger the application speedup achieved by *PRAcTICAL-MPI*. This behavior is due to the increasing number of communications. Therefore, the improvement of the communication performance has a bigger impact on the overall application performance. Thus, we can conclude that overal scalability is enhanced with *PRAcTICAL-MPI*.

6 Conclusions and Future Work

In this paper we have presented a portable optimization of MPI communications, called *PRAcTICAL-MPI*. The main goal of *PRAcTICaL-MPI* is to enhance the performance and scalability of MPI-based applications reducing the volume of communications by applying adaptive run-time lossless compression. Furthermore, *PRAcTICaL-MPI* is fully portable and transparent for both applications and MPI Implementations.

The evaluation results show that *PRAcTICAL-MPI* improves the speedup of BIPS3D for most of the scenarios considered, because the volume of communications is reduced by using the best compression algorithm per message. It also demonstrates that, even when compression is deactivated, application performance speedup is close to one. Furthermore, the run-time performance gain is bigger in most of the cases when more processes are employed, which increases scalability, and illustrates that our method will be most useful when utilised for massively parallel systems.

In future work, we want to evaluate the performance of *PRAcTICAL-MPI* technique with new compression algorithms like Snappy or PFOR. Furthermore, we want to apply *PRAcTICAL-MPI* to more MPI routines, such as collective IO, non-contiguous communications.

Acknowledgments. This work has been performed by using the facilities of HECToR, the UKs national high performance computing service, which is provided by UoE HPCx Ltd at the University of Edinburgh, Cray Inc and NAG Ltd, and funded by the Office of Science and Technology through EPSRCs High End Computing Programme.

References

1. Message Passing Interface Forum, MPI: A message-passing interface standard. International Journal of Supercomputer Applications 8, 165–414 (1994)
2. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing 22(6), 789–828 (1996)
3. 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: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004)

4. Burns, G., Daoud, R., Vaigl, J.: LAM: An open cluster environment for MPI. In: Proceedings of Supercomputing Symposium 1994 (1994)
5. Zigon, R.: Run length encoding. Dr. Dobb's Journal of Software Tools 14(2) (February 1989)
6. Knuth, D.E.: Dynamic huffman coding. J. Algorithms 6(2), 163–180 (1985)
7. Salvatore Coco, D.G., D'Arrigo, V.: A Rice-based Lossless Data Compression System For Space. In: Proceedings of the 2000 IEEE Nordic Signal Processing Symposium, pp. 133–142 (2000)
8. Burtscher, M., Ratanaworabhan, P.: FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. IEEE Transactions on Computers 58(1), 18–31 (2009)
9. Oberhumer, M.F.X.J.: Lzo real-time data compression library (2005)
10. Filgueira, R., Carretero, J., Singh, D.E., Calderon, A., Garcia, F.: Adpative-compi: Enhancing mpi based applications performance and scalability by using adaptive compression. International Journal of High Performance Computing and Applications (April 2010)
11. Schulz, M., de Supinski, B.R.: A flexible and dynamic infrastructure for mpi tool interoperability. In: Proceedings of the 2006 International Conference on Parallel Processing, ICPP 2006, pp. 193–202. IEEE Computer Society, Washington, DC (2006), http://dx.doi.org/10.1109/ICPP.2006.6
12. Zaki, O., Lusk, E., Swider, D.: Toward scalable performance visualization with Jumpshot. High Performance Computing Applications 13, 277–288 (1999)
13. Balkanski, D., Trams, M., Rehm, W.: Heterogeneous Computing With MPICH/Madeleine and PACX MPI: A Critical Comparison (2003)
14. Keller, M.L.R.: Using PACX-MPI in metacomputing applications. In: 18th Symposium Simulationstechnique, Erlangen, September 12-15 (2005)
15. Ratanaworabhan, P., Ke, J., Burtscher, M.: Fast Lossless Compression of Scientific Floating-Point Data. In: DCC 2006: Proceedings of the Data Compression Conference, pp. 133–142. IEEE Computer Society, Washington, DC (2006)
16. Ke, J., Burtscher, M., Speight, E.: Runtime Compression of MPI Messages to Improve the Performance and Scalability of Parallel Applications. In: SC 2004: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, p. 59. IEEE Computer Society, Washington, DC (2004)
17. Carretero, J., No, J., Park, S.-S., Choudhary, A., Chen, P.: COMPASSION: a Parallel I/O Runtime System Including Chunking and Compression for Irregular Applications. In: Sloot, P., Bubak, M., Hertzberger, B. (eds.) HPCN-Europe 1998. LNCS, vol. 1401, pp. 668–677. Springer, Heidelberg (1998)
18. Garíca, F., Galderón, A., Carretero, J.: MiMPI: A Multithread-Safe Implementation of MPI. In: Margalef, T., Dongarra, J., Luque, E. (eds.) PVM/MPI 1999. LNCS, vol. 1697, pp. 207–214. Springer, Heidelberg (1999)
19. Filgueira, R., Singh, D.E., Calderón, A., Carretero, J.: CoMPI: Enhancing MPI Based Applications Performance and Scalability Using Run-Time Compression. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) PVM/MPI. LNCS, vol. 5759, pp. 207–218. Springer, Heidelberg (2009)
20. Loureiro, A., González, J., Pena, T.F.: A parallel 3D semiconductor device simulator for gradual heterojunction bipolar transistors. Int. Journal of Numerical Modelling: Electronic Networks, Devices and Fields 16, 53–66 (2003)
21. Karypis, G., Kumar, V.: Metis a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Tech. Rep. (1998)