

Hierarchical redesign of classic MPI reduction algorithms

Khalid Hasanov¹ · Alexey Lastovetsky²

© Springer Science+Business Media New York 2016

Abstract Optimization of MPI collective communication operations has been an active research topic since the advent of MPI in 1990s. Many general and architecture-specific collective algorithms have been proposed and implemented in the state-of-the-art MPI implementations. Hierarchical topology-oblivious transformation of existing communication algorithms has been recently proposed as a new promising approach to optimization of MPI collective communication algorithms and MPI-based applications. This approach has been successfully applied to the most popular parallel matrix multiplication algorithm, SUMMA, and the state-of-the-art MPI broadcast algorithms, demonstrating significant multifold performance gains, especially for large-scale HPC systems. In this paper, we apply this approach to optimization of the MPI Reduce and Allreduce operations. Theoretical analysis and experimental results on a cluster of Grid⁵000 platform are presented.

Keywords MPI collectives · Reduction · Hierarchical MPI

1 Introduction

The message passing interface (MPI) [1] is widely accepted as the de facto standard for programming distributed-memory systems. The communication in the MPI is performed by message passing between processes either using point-to-point or collective

✉ Alexey Lastovetsky
alexey.lastovetsky@ucd.ie

Khalid Hasanov
khasanov@ie.ibm.com

¹ IBM Research Ireland, Dublin, Ireland

² University College Dublin, Dublin, Ireland

operations. The point-to-point operations are expressed as a set of send and receive functions that allow transmitting a message of specified size and type between two processes. The collective communication functions usually involve more than two processes and provide higher abstraction of parallel processing than point-to-point operations. Some collective communication operations offer collective computation and synchronization features besides transmitting data among the processes. The high level of abstraction of the collectives makes it possible to express an appropriate problem in an elegant declarative way. This in turn improves their portability across different platforms while hiding the implementation details. Therefore, the use of collectives rather than point-to-point operations is preferable where applicable.

According to research studies over the past two decades [2,3], MPI reduction operations, particularly MPI reduce and allreduce, are the most used collective operations in scientific applications. In the reduce operation, each node i owns a vector x_i of n elements. After completion of the operation all the vectors are reduced element-wise to a single n -element vector which is owned by a specified root process. In the allreduce operation, the result vector will be accumulated on all the processes in the same way as it happens in the reduce. Another widely used reduction operation is reduce-scatter, which can be seen as a reduce operation followed by a scatter operation. This paper focuses only on the reduce and allreduce operations.

Optimization of MPI collective operations has been an active research topic since the advent of MPI in 1994. Many general and architecture-specific collective algorithms have been proposed and implemented in the state-of-the-art MPI implementations. Hierarchical topology-oblivious transformation of existing communication algorithms has been recently proposed as a new promising approach to optimization of MPI collective communication algorithms and MPI-based applications [4,5,17]. This approach has been successfully applied to the most popular parallel matrix multiplication algorithm, SUMMA [6], and the state-of-the-art MPI broadcast and reduce algorithms, demonstrating significant multifold performance gains, especially on large-scale HPC systems [18]. In this article, we extend our conference paper [7] on topology-oblivious hierarchical optimization of MPI reduce operation by including a study of optimization of MPI allreduce operation using the same technique.

1.1 Contributions

We propose a topology oblivious hierarchical technique to optimize legacy MPI collective communication operations and show its applicability in the context of MPI reduce and allreduce algorithms. The approach is simple and general, allowing for application of the proposed optimization to any existing MPI collective communication algorithm. As by design the original algorithm is a particular case of its hierarchically transformed counterpart, the performance of the algorithm will either improve or stay the same in the worst case scenario. Theoretical study of the hierarchical transformation of six reduce and five allreduce algorithms, which are implemented in MPICH [19] and Open MPI [8], is presented. The theoretical results have been experimentally validated on a cluster of Grid'5000 (see <http://www.grid5000.fr>) infrastructure.

1.2 Outline

The rest of the paper is structured as follows. Section 2 discusses related work. The hierarchical optimization of MPI reduction algorithms is introduced in Sect. 3. The experimental results are presented in Sect. 4. Finally, Sect. 5 concludes the presented work and discusses future directions.

2 Related work

The reduce operation was implemented as an inverse broadcast in the CCL [9] library and was not optimized for different message sizes. The InterComm [10] library proposes a more advanced reduce implementation as a combination of reduce–scatter and gather, and tries to be efficient for small and large message sizes. The allreduce operation was implemented in a similar way as a reduce–scatter followed by an all-gather operation. The MagPIe [11] library provides optimized collective algorithms, including algorithms for reduction operations for wide area systems.

Design and high-performance implementation of collective communication operations and commonly used algorithms, such as minimum-spanning tree reduce algorithm, are discussed in [12]. The authors also discuss the lower bounds of the reduction operations. The lower bounds on the latency and computation costs are the same in both the reduce and allreduce operations, and equal to $\lceil \log_2(p) \rceil$ and $\frac{p-1}{p} \times m \times \gamma$, respectively. Here, p is the total number of processes in the operation, m is the message size, and γ are the flops. On the other hand, the lower bound on the bandwidth cost is $m \times \beta$ for the reduce, while it is $2 \times \frac{p-1}{p} \times m \times \beta$ for the allreduce operation.

Automatic tuning of collectives for a given system by conducting a series of experiments on the system was discussed in [13]. Rabenseifner [2] proposes five reduction algorithms optimized for different message sizes and numbers of processes. Cheetah framework [3] implements MPI reduction operations in a hierarchical way on multi-core systems, which supports multiple communication mechanisms. Unlike that work, our optimization is topology oblivious, and our hierarchical algorithms have not been designed as new algorithms from scratch, but rather employs the existing reduction algorithms underneath.

2.1 MPI reduce and allreduce algorithms

We assume that the time to send a message of size m between any two MPI processes is modeled with Hockney model [14] as $\alpha + m \times \beta$, where α is the latency per message and β is the reciprocal bandwidth per byte. It is also assumed that the computation cost per byte in the reduction operation is γ on any MPI process. Unless otherwise noted, in the rest of the paper we will call MPI process just process.

This section outlines general purpose MPI reduce and allreduce algorithms, which present in the state-of-the-art MPI implementations, such as MPICH and Open MPI. We call them general purpose as by design they do not assume any knowledge of the underlying topology and platform. The general purpose reduce algorithms have the

Table 1 Reduce algorithms and their theoretical cost within the Hockney model

Algorithm	Theoretical cost
Flat tree	$(p - 1) \times (\alpha + m \times \beta + m \times \gamma)$
Linear tree	$(p - 1) \times (\alpha + m \times \beta + m \times \gamma)$
Pipeline	$(p + X - 2) \times (\alpha + \frac{m}{X} \times \beta + \frac{m}{X} \times \gamma)$
Binary tree	$2 (\log_2 (p + 1) + X - 2) \times (\alpha + \frac{m}{X} \times \beta + \frac{m}{X} \times \gamma)$
Binomial tree	$\log_2 (p) \times (\alpha + m \times \beta + m \times \gamma)$
Rabenseifner's reduce [2]	$2 \log_2 (p) \times \alpha + 2 \frac{p-1}{p} \times m \times \beta + \frac{p-1}{p} \times m \times \gamma$

theoretical costs given in the Table 1. We do not provide theoretical analysis of those algorithms, as this work is an extension of our conference paper [7] which already provides detailed discussion of them. Instead, we provide a broader discussion of the MPI allreduce algorithms.

– Linear allreduce algorithm

Open MPI implements the linear allreduce algorithm as a linear reduce to a specified root followed by a linear broadcast from the same root. Despite that the root process faces the communication and computation overhead, the linear algorithm can be a preferred algorithm for small messages on a small number of processes. The time for the allreduce can be derived as the sum of the linear reduce (Table 1) and linear broadcast [5] times:

$$2 \times (p - 1) \times (\alpha + m \times \beta) + (p - 1) \times m \times \gamma. \quad (1)$$

– Recursive doubling allreduce algorithm

In each step of the recursive doubling allreduce algorithm, the distance between the communicating processes is doubled. For a power-of-two number of processes, the algorithm consists of $\log_2 p$ steps. The amount of data exchanged by each process doubles in each step as well. The total execution time of the algorithm is as follows: $\log_2 p \times (\alpha + m \times \beta + m \times \gamma)$.

– Rabenseifner's allreduce algorithm

Rabenseifner's allreduce algorithm consists of a reduce–scatter followed by an allgather. The cost of the algorithm is: $2 \times \log_2 p \times \alpha + 2 \times \frac{p-1}{p} \times m \times \beta + \frac{p-1}{p} \times m \times \gamma$. The main limitation of this algorithm is that it cannot be applied to user-defined operations due to the difficulty of using reduce–scatter with user-defined operations.

– Ring allreduce algorithm

The ring algorithm for allreduce uses a nearest-neighbour communication pattern and is used for commutative operations; it consists of computation and distribution phases. The send buffer is divided into p blocks of size $\frac{\text{send_count}}{p}$. The algorithm can be quite easily modified to support a use case where p does not divide the send count [15]. In each iteration i of the computation phase, rank $(r - 1 + p) \% p$ sends block $(r - i + p) \% p$ to rank r , which in turn receives the data using a non-blocking

receive and performs the reduction operation on the block before sending the result to rank $(r + 1) \% p$. In the data distribution phase, rank r receives the reduced data from its left neighbour and sends it to its right neighbour. The algorithm continues this way in $2 \times p - 1$ iterations. Its total cost will be the following:

$$2 \times (p - 1) \times \left(\alpha + \left\lceil \frac{m}{p} \right\rceil \times \beta \right) + (p - 1) \frac{m}{p} \times \gamma. \tag{2}$$

The algorithm assumes that $send_count > p$.

– Segmented ring allreduce algorithm

In the segmented ring allreduce algorithm, all blocks are divided into segments of X size. Then the computation phase is performed in a block-cyclic way for each of the segment groups. The distribution phase is executed similarly to that of the non-segmented ring algorithm. The main limitation of the algorithm is that it can be applied only if the send count is greater than $p \times \frac{block_size}{X}$. The cost of the algorithm is given below:

$$(p + X - 2) \times \left(\alpha + \frac{m}{X} \times \beta + \frac{m}{X} \times \gamma \right) + (p - 1) \times \left(\alpha + \left\lceil \frac{m}{p} \right\rceil \times \beta \right). \tag{3}$$

3 Hierarchical optimization of MPI reduce and allreduce algorithms

This section introduces a topology-oblivious optimization of MPI reduce and allreduce algorithms. The idea is inspired by our previous study on the optimization of the communication cost of parallel matrix multiplication [4] and MPI broadcast [5] on large-scale distributed memory platforms.

3.1 Hierarchical optimization of MPI reduce algorithms

The proposed optimization technique is based on the arrangement of the p processes participating in the reduce into logical groups. For simplicity, it is assumed that the number of groups divides the number of MPI processes and can change between one and p . Let G be the number of groups. Then there will be $\frac{p}{G}$ MPI processes per group. Figure 1 shows an arrangement of eight processes in the original MPI reduce operation (left), and an arrangement in a hierarchical reduce operation (right) with two groups of four processes. The hierarchical reduce consists of two phases: in the first phase, a group leader is selected for each group and the leaders start reduce operation inside their own group in parallel (in this example, among 4 processes). In the next phase, the reduce is performed among the group leaders (in this example, between 2 processes). The grouping can be done by taking the topology into account as well. However, in this work the grouping is topology oblivious and the first process in each group is selected as the group leader. In general, different algorithms can be used for reduce operations among the group leaders and within each group. This work focuses on the case where the same algorithm is employed at both levels of hierarchy. Algorithm 1 shows the pseudocode of the hierarchically transformed MPI reduce operation. Line 4 calculates

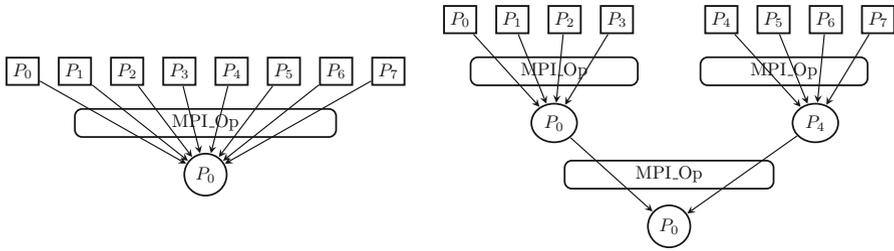


Fig. 1 Logical arrangement of processes in MPI reduce (left) and hierarchical MPI reduce (right)

the root for the reduce between the groups. Then line 5 creates a sub-communicator of G processes between the groups, and line 6 creates a sub-communicator of $\frac{p}{G}$ processes inside the groups. We utilize the `MPI_Comm_split` routine to create new sub-communicators.

Algorithm 1: Hierarchical optimization of MPI reduce operation.

Data: p - Number of processes

Data: G - Number of groups

Data: *sendbuf* - Send buffer

Data: *recvbuf* - Receive buffer

Data: *count* - Number of entries in send buffer (integer)

Data: *datatype* - Data type of elements in send buffer

Data: *op* - MPI reduce operation handle

Data: *root* - Rank of reduce root

Data: *comm* - MPI communicator handle

Result: The root process has the reduced message

begin

```

1  MPI_Comm comm_outer          /* communicator between the groups */
2  MPI_Comm comm_inner         /* communicator inside the groups */
3  int root_outer              /* root of reduce between the groups */
4  root_outer = Calculate_Root_Outer(G, p, root, comm)
5  comm_outer = Create_Comm_Between_Groups(G, p, root_outer, comm)
6  comm_inner = Create_Comm_Inside_Groups(G, p, root, comm)
7  MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm_inner)
8  MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root_outer, comm_outer)

```

The reduce algorithms which have flat design in their arrangement of processes, such as flat tree, pipeline reduce algorithms, can be improved by the hierarchical transformation. On the other hand, the reduce algorithms which have some kind of hierarchy in their design cannot be improved by this transformation. The binary, binomial and Rabenseifner's algorithms belong to the second group where we could not improve their performance. However, in such cases, the parameterization of the hierarchy lets the hierarchical algorithms fall back to the original underlying algorithm and be equally fast. In addition, the state-of-the-art MPI implementations, such as Open MPI employs different algorithms and switches between them depending on the message size, the segment size, and the number of MPI processes. That means, this kind of hierarchical

transformation can also improve the native reduction operations in the MPI implementation. Our experimental studies cover this case as well.

Because of the limited space, we only show theoretical analysis of the hierarchical flat tree reduce algorithm here. A detailed theoretical analysis of the other hierarchical reduce algorithms can be found in our previous work [7].

3.1.1 Hierarchical transformation of flat tree reduce algorithm

If we assume that p processes are organized into G groups, then each group will contain $\frac{p}{G}$ number of processes. The reduce operations inside each group are independent of each other and can happen in parallel. The second phase of the algorithm starts among G groups, or equivalently among G MPI processes, as soon as the reduce operations finish inside each groups. Therefore, if we apply the cost function of the flat tree reduce algorithm (see Table 1) in each phase, the cost of the reduce operations among groups and inside groups will be $(G-1) \times (\alpha + m \times \beta + m \times \gamma)$ and $(\frac{p}{G}-1) \times (\alpha + m \times \beta + m \times \gamma)$, respectively. Thus, the overall run time of the hierarchical flat tree reduce algorithm can be seen as a function of G : $F(G) = (G + \frac{p}{G} - 2) \times (\alpha + m \times \beta + m \times \gamma)$. The derivative of the function is $(1 - \frac{p}{G^2}) \times (\alpha + m \times \beta + m \times \gamma)$, and $p = \sqrt{G}$ is the minimum point of the function in the interval $(1, p)$. Thus, the optimal value of the function will be as follows: $F(\sqrt{p}) = (2\sqrt{p} - 2) \times (\alpha + m \times \beta + m \times \gamma)$.

3.2 Hierarchical optimization of MPI allreduce algorithms

The allreduce operation has a more complex communication pattern than the broadcast and reduce operations. Therefore, its hierarchical transformation is not as trivial as it was in those cases. The main difficulty comes from the fact that in our design we are trying not to introduce a new allreduce algorithm, but rather use existing algorithms. To be more clear, in the case of allreduce we would only like to use the allreduce communication operation both inside and between groups. If it was not the case, it would be possible to design a hierarchical allreduce in very different ways. One example of hierarchical allreduce implementation could be using a hierarchical reduce followed by a hierarchical broadcast.

The design of the hierarchical allreduce follows a similar design philosophy to the hierarchical broadcast and hierarchical reduce operations. Namely, the main idea is to organize the processes into logical groups. The grouping results in a two-level hierarchy and the allreduce operations are performed in two phases. In the first phase, the operation is operated over the processes inside each group independently. Later on, as soon as this phase finishes, the processes at the same index position from different groups start the allreduce operation among them on the partially reduced value. However, unlike the hierarchical reduce, more sub-communicators are needed to perform the reduction operation further from all the processes at the same index position inside each group. The communication pattern of the hierarchical allreduce is demonstrated in Fig. 2.

The hierarchical approach does not improve the recursive doubling allreduce and Rabenseifner's allreduce algorithms. However, we theoretically and experimentally

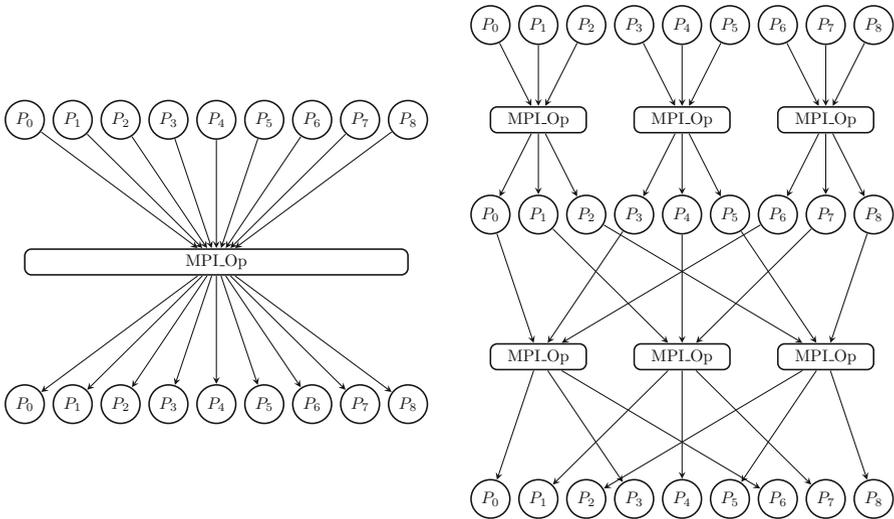


Fig. 2 Logical arrangement of processes in MPI allreduce (*left*) and hierarchical MPI allreduce (*right*)

demonstrate that the linear, ring, and segmented ring algorithms can be improved after transforming them hierarchically.

3.2.1 Theoretical analysis of the allreduce algorithms

The theoretical cost of the hierarchical linear allreduce can be derived by applying linear allreduce (see formula 1) among G groups and $\frac{p}{G}$ group members. The cost of these two allreduce together will be as follows:

$$F(G) = 2(\alpha + m \times \beta) \times \left(1 - \frac{p}{G^2}\right) + \left(1 - \frac{p}{G^2}\right) \times m\gamma. \tag{4}$$

For a fixed $p, \alpha, \beta,$ and $\gamma,$ it can be shown that $F(G)$ function attains its minimum at $G = \sqrt{p}$ and the minimum of the function at this point is given as below:

$$F(\sqrt{p}) = 4(\sqrt{p} - 1) \times (\alpha + m\beta) + 2(\sqrt{p} - 1) \times m\gamma. \tag{5}$$

In the same way, we can show that the optimal value of the hierarchical ring allreduce algorithms is attained at $G = \sqrt{p}$ and given as follows:

$$F(\sqrt{p}) = 4 \times (\sqrt{p} - 1) \times \alpha + 4 \times \left(1 - \frac{1}{\sqrt{p}}\right) \times m \times \beta + 2 \times \left(1 - \frac{1}{\sqrt{p}}\right) \times m \times \gamma. \tag{6}$$

On the other hand, the cost function of the segmented ring algorithm attains its minimum at $G = \sqrt{p}$ only if $2 \times p \times X \times \alpha + m \times (p - X) \times \beta + m \times p \times \gamma > 0.$ In this case, the optimal value will be:

$$F(\sqrt{p}) = (2 \times \sqrt{p} + 2 \times X - 4) \times \left(\alpha + \frac{m}{X} \times \beta + \frac{m}{X} \times \gamma \right) + (2 \times \sqrt{p} - 2) \times \alpha + 2 \times \left(1 - \frac{1}{\sqrt{p}} \right) \times m \times \beta. \quad (7)$$

4 Experiments

The experiments were carried out on the Graphene cluster of Grid'5000 infrastructure in France (see <http://www.grid5000.fr>). Almost all the sites are interconnected by 10 Gb/s high-speed network. The cluster is equipped with 144 nodes and each node has a disk of 320 GB storage, 16 GB of memory and 4 cores of CPU Intel Xeon X3440. The nodes in the graphene cluster are interconnected via 20 Gb/s Infiniband and Gigabyte Ethernet.

The experiments have been done with Open MPI 1.8.4, which provides several algorithms for MPI reduction operations. This includes linear, chain, pipeline, binary, binomial, and in-order binary algorithms for the reduce operation. The allreduce operation comes with basic linear allreduce, recursive doubling, ring, and segmented ring algorithms. In addition, the implementation provides platform-/architecture-specific algorithms, some of which are reduction algorithms for Infiniband networks, and the Cheetah framework for multicore architectures. In this work, we do not consider the platform-specific reduction implementations. We used the same approach as described in MPIBlib [16] to benchmark our experiments. During the experiments, the mentioned reduce and allreduce algorithms were selected using Open MPI MCA (Modular Component Architecture) parameters. MPI_MAX operation has been used in the experiments.

The theoretical and experimental results showed that the hierarchical approach mainly improves the algorithms which assume flat arrangements of the processes, such as linear, chain, pipeline, ring, and segmented ring. On the other hand, the native Open MPI reduce and allreduce operations switch among different algorithms depending on the requested message size, the count and the number of processes. This means that the hierarchical transformation can improve the native reduction operation as well. It is expected that [7] the overhead from the MPI_Comm_split operation should affect only reduce operations with smaller message sizes. Figure 3 validates this with experimental results. The hierarchical reduce operation of 1 KB message with the underlying native reduce achieved its best performance when the number of groups was one, as the overhead from the split operation itself was higher than the reduce.

Figure 4 presents experiments with the hierarchical pipeline reduce with a message size of 16 KB with 1 KB segmentation. The performance of the pipeline algorithm with larger messages and segment sizes of 32 and 64 KB can be found in [7]. Figure 5 shows the speedup of the hierarchical transformation of native Open MPI reduce operation, linear, chain, pipeline, binary, binomial, and in-order binary reduce algorithms with message sizes starting from 16 KB up to 16 MB. Except binary, binomial, and in-order binary reduce algorithms, there is a significant performance improvement. In the figure, NT is native Open MPI reduce operation, LN is linear, CH is chain, PL is pipeline with 32 KB segmentation, BR is binary, BL is binomial, and IBR denotes in-

Fig. 3 Time spent on MPI_Comm_split and hierarchical native reduce. $m = 1$ KB, $p = 512$

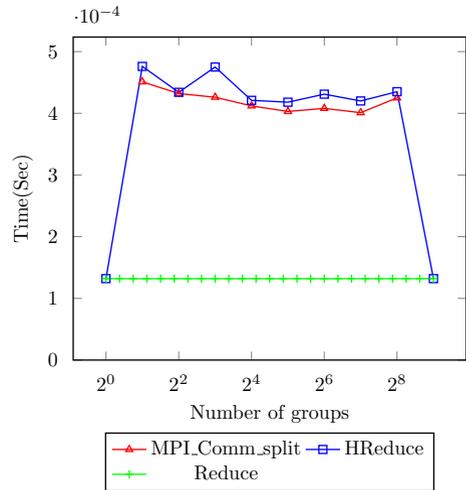
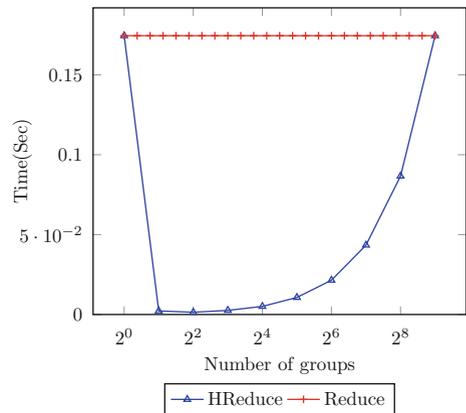


Fig. 4 Hierarchical pipeline reduce. $m = 16$ KB, segment 1 KB and $p = 512$



order binary tree reduce algorithm. We would like to highlight one important point that Fig. 5 does not compare the performance of different Open MPI reduce algorithms, but shows the speedup of their hierarchical transformations. Each of these algorithms can be better than the others in some specific settings depending on the message size, number of processes, underlying network, and so on. At the same time, the hierarchical transformation of these algorithms will either improve their performance or be equally fast.

Figure 6 demonstrates experiments with the hierarchical allreduce. On the left, we have the results with the hierarchical linear allreduce where any number of groups decrease the execution time of the original allreduce operation. The figure on the right shows the speedups of the hierarchical transformation of different allreduce algorithms that include the native allreduce operation, basic linear (reduce followed by broadcast without message segmentation), non-overlapping (reduce followed by broadcast with message segmentation), recursive-doubling, ring, and segmented-ring algorithms. As

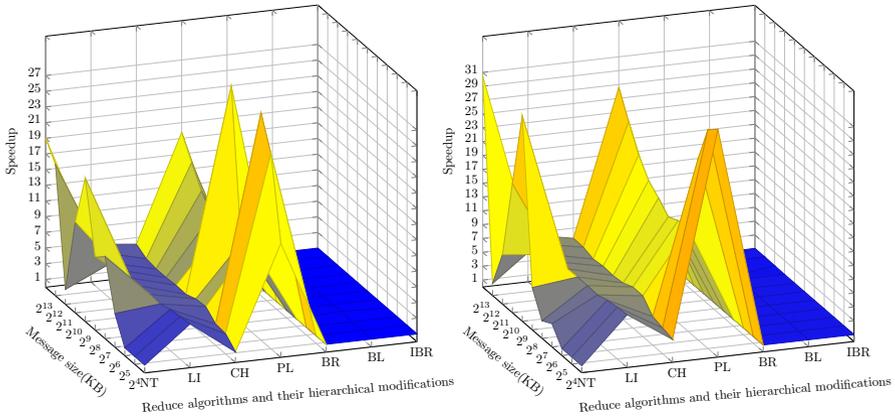


Fig. 5 Speedup on 256 (left) and 512 (right) cores, one process per core

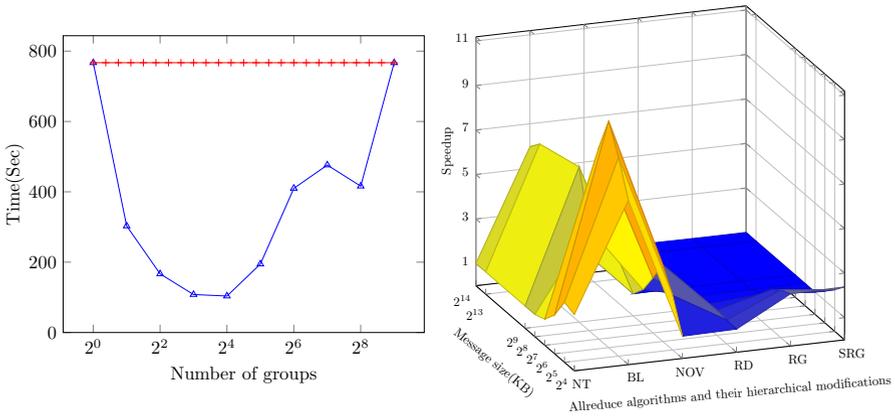


Fig. 6 Hierarchical linear allreduce performance with a message of size 16 KB (left). Speedup of hierarchical allreduce on 512 cores with different algorithms and message sizes (right). *NT* native open MPI allreduce operation, *BL* basic linear, *NOV* non-overlapping, *RD* recursive doubling, *RG* ring, and finally *SRG* segmented ring algorithm

seen, there is no improvement over the recursive-doubling algorithm; however, all the other algorithms get their execution time reduced after their hierarchical transformation. As expected, this improvement is multifold for basic linear and non-overlapping allreduce algorithms.

5 Conclusion

Despite that there has been a lot of research in MPI collective communications, this work shows that their performance is far from optimal and there is some room for improvement. Indeed, our simple hierarchical optimization, which transforms existing

MPI reduction algorithms into two-level hierarchy, gives significant improvement on small- and medium-scale platforms.

We plan to investigate the effect of using different reduction algorithms in each hierarchy and different number of processes per group.

Automatic estimation of the optimal number of groups is the main challenge of the hierarchical approach. To deal with that we are actively developing an MPI software library which will provide automatic estimation and selection of the optimal number of groups during run time.

Acknowledgments The experiments presented in this publication were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several universities as well as other funding bodies (see <https://www.grid5000.fr>). This work was also supported by Science Foundation Ireland under Grant Number 14/IA/2474.

References

1. Message passing interface forum. <http://www.mpi-forum.org/>. Accessed 20 Feb 2016
2. Rabenseifner R (2004) Optimization of collective reduction operations. In: 2004 International conference on computational science, pp 1–9
3. Venkata MG et al (2013) Optimizing blocking and nonblocking reduction operations for multicore systems: hierarchical design and implementation. In: 2013 IEEE international conference on cluster computing, pp 1–8
4. Hasanov K, Quintin JN, Lastovetsky A (2015) Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms. *J Supercomput* 71(11):3991–4014
5. Hasanov K, Quintin JN, Lastovetsky A (2014) High-level topology-oblivious optimization of MPI broadcast algorithms on extreme-scale platforms. In: Euro-Par 2014: parallel processing workshops, lecture notes in computer science, vol 8806, Springer, New York, pp 412–424
6. de Geijn RA, Jerrell W (1997) SUMMA: scalable universal matrix multiplication algorithm. *Concurr Pract Exp* 9(4):255–274
7. Hasanov K, Lastovetsky A (2015) Hierarchical optimization of MPI reduce algorithms. In: PaCT 2015, lecture notes in computer science, vol 9251, Springer, New York, pp 21–34
8. Gabriel E, Fagg G, Bosilca G, Angskun T, Dongarra J et al (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. In: EuroPVM/MPI 2004, lecture notes in computer science, vol 3241, Springer, New York, pp 97–104
9. Bala V, Bruck J, Cypher R, Elustondo P, Ho C-T, Ho C-T, Kipnis S, Snir M (1995) CCL: a portable and tunable collective communication library for scalable parallel computers. *IEEE Trans Parallel Distrib Syst* 6(2):154–164
10. Barnett M, Shuler L, van De Geijn R, Gupta S, Payne DG, Watts J (1994) Interprocessor collective communication library (InterCom). In: IEEE scalable high-performance computing conference, pp 357–364
11. Kielmann T, Hofman RF, Bal HE, Plaat A, Bhoedjang RA (1999) MagPIe: MPI's collective communication operations for clustered wide area systems. *ACM Sigplan Notices* 34(8):131–140
12. Chan EW, Heimlich MF, Purkayastha A, Van de Geijn RA (2004) On optimizing collective communication. In: 2004 IEEE international conference on cluster computing, pp 145–155
13. Vadhiyar SS, Fagg GE, Dongarra J (2000) Automatically tuned collective communications. In: ACM/IEEE conference on supercomputing, p 3
14. Hockney RW (1994) The communication challenge for MPP: intel paragon and Meiko CS-2. *Parallel Comput* 20(3):389–398
15. Pješivac-Grbović J (2007) Towards automatic and adaptive optimizations of MPI collective operations. PhD thesis, University of Tennessee, Knoxville
16. Lastovetsky A, Rychkov V, O'Flynn M (2008) MPIBlib: Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters. In: EuroPVM/MPI 2008, lecture notes in computer science, vol 5205, Springer, New York, pp 227–238

17. Hasanov K, Quintin JN, Lastovetsky A (2015) Topology-oblivious optimization of MPI broadcast algorithms on extreme-scale platforms. *Simul Model Pract Theory* 58:30–39
18. Hasanov K (2015) Hierarchical approach to optimization of MPI collective communication algorithms. PhD. thesis, University College Dublin
19. MPICH-A Portable Implementation of MPI. <http://www.mpich.org/>. Accessed 01 March 2016