Journal of Parallel and Distributed Computing 165 (2022) 1–16

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Model-based selection of optimal MPI broadcast algorithms for multi-core clusters [☆]

Emin Nuriyev [a,*], Juan-Antonio Rico-Gallego [b], Alexey Lastovetsky [a]

[a] University College Dublin, Belfield, Dublin 4, Ireland
[b] University of Extremadura, Avd. Universidad s/n, 10003, Cáceres, Spain

ARTICLE INFO

Article history:
Received 22 March 2021
Received in revised form 11 December 2021
Accepted 11 March 2022
Available online 23 March 2022

Keywords:
Message passing
Collective communication algorithms
Communication performance modeling
MPI
Multi-core clusters

ABSTRACT

The performance of collective communication operations determines the overall performance of MPI applications. Different algorithms have been developed and implemented for each MPI collective operation, but none proved superior in all situations. Therefore, MPI implementations have to solve the problem of selecting the optimal algorithm for the collective operation depending on the platform, the number of processes involved, the message size(s), etc. The current solution method is purely empirical. Recently, an alternative solution method using analytical performance models of collective algorithms has been proposed and proved both accurate and efficient for one-process-per-CPU configurations. The method derives the analytical performance models of algorithms from their code implementation rather than from high-level mathematical definitions, and estimates the parameters of the models separately for each algorithm. The method is network and topology oblivious and uses the Hockney model for point-to-point communications. In this paper, we extend that selection method to the case of clusters of multi-core processors, where each core of the platform runs a process of the MPI application.

We present the proposed approach using Open MPI broadcast algorithms, and experimentally validate it on three different clusters of multi-core processors, Grisou, Gros and MareNostrum4.

1. Introduction

The Message Passing Interface [1] (MPI) offers portable and scalable performance on high performance computing (HPC) platforms. Therefore, it has been dominantly used since its invention in HPC applications. MPI proposes an execution model based on processes deployed on the hardware resources of the HPC platform and communicating using message passing primitives. Both point-to-point and *collective routines* are defined in the MPI standard with different semantics, including non-blocking, buffered and persistent communication.

Collective routines involve a group of processes communicating in an isolated context, and those collectives rely on the semantics of *collective operations* such as *broadcast*, *gather*, *reduce* and so forth. A profiling study [34] reports that in average 80% of the total execution time of MPI applications is consumed by MPI collective operations. That is why significant research efforts have been invested in the design and implementation of efficient collective algorithms aimed to improve the performance of collective operations [44,35,5,4]. For example, Open MPI 3.1 [14] employs six different algorithms to implement *MPI_Bcast* and five algorithms to implement *MPI_Allreduce*. On a given platform, different algorithms will be optimal depending on many factors including the physical topology of the network, number of processes, message sizes and so forth. Unfortunately, there is no single collective algorithm optimal in all situations. Therefore, there exists a problem of selection of the optimal algorithm for each call of a collective routine, which normally depends on the platform, the number of processes, the message size and so forth.

There are two ways how this selection can be made in the MPI program. The first one, MPI_T interface [1], is provided by the MPI standard and allows the MPI programmer to explicitly select the collective algorithm from the list of available algorithms for each collective call at runtime. It does not however solve the problem of optimal selection but delegates its solution to the programmer.

[☆] This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474. This work has been partially supported by HPC-Europa3 Transnational Access programme under Grant Number HPC17M60Z5. Experiments presented in this paper 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.

[*] Corresponding author.
E-mail addresses: emin.nuriyev@ucdconnect.ie (E. Nuriyev), jarico@unex.es (J.-A. Rico-Gallego), alexey.lastovetsky@ucd.ie (A. Lastovetsky).

https://doi.org/10.1016/j.jpdc.2022.03.012
0743-7315/

The second one is transparent to the MPI programmer and provided by MPI implementations. It uses a simple *decision function* in each collective routine, which is used to select the algorithm at runtime. The decision function is empirically derived from extensive testing on a dedicated system. For example, the most popular MPI implementations, MPICH and Open MPI, for each collective operation both use a simple decision routine selecting the algorithm depending on the message size and number of processes [40,14,13]. Listing 1 illustrates such a decision routine, showing the Open MPI decision code for MPI_Bcast.

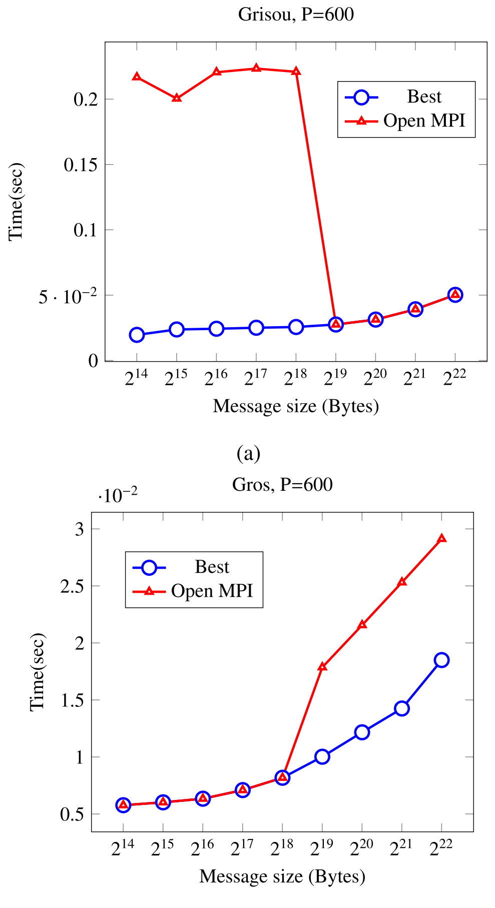
Listing 1 Open MPI decision function for MPI_Bcast.

```
int bcast_intra_dec_fixed(void *buff,int count,MPI_Datatype
*datatype,int root,MPI_comm *comm)
{
const size_t small_message_size = 2048;
const size_t intermediate_message_size = 370728;
const double a_p16 = 3.2118e-6;
const double b_p16 = 8.7936;
const double a_p64 = 2.3679e-6;
const double b_p64 = 1.1787;
const double a_p128 = 1.6134e-6;
const double b_p128 = 2.1102;

int communicator_size;
size_t message_size, dsize;

communicator_size = MPI_comm_size(comm);
MPI_Type_size(datatype, &dsize);
message_size = dsize * (unsigned long)count;

if ((message_size < small_message_size) || (count <= 1)) {
return binomial_tree_bcast(. . .);
} else if (message_size < intermediate_message_size) {
return split_binary_tree_bcast(. . .);
} else if (communicator_size < (a_p128 * message_size + b_p128))
     {
return chain_bcast(. . . );
} else if (communicator_size < 13) {
return split_binary_tree_bcast(. . .);
} else if (communicator_size < (a_p64 * message_size + b_p64)) {
return chain_bcast(. . .);
} else if (communicator_size < (a_p16 * message_size + b_p16)) {
return chain_bcast(. . .);
}
return chain_bcast(. . .);
}
```



(a)

(b)

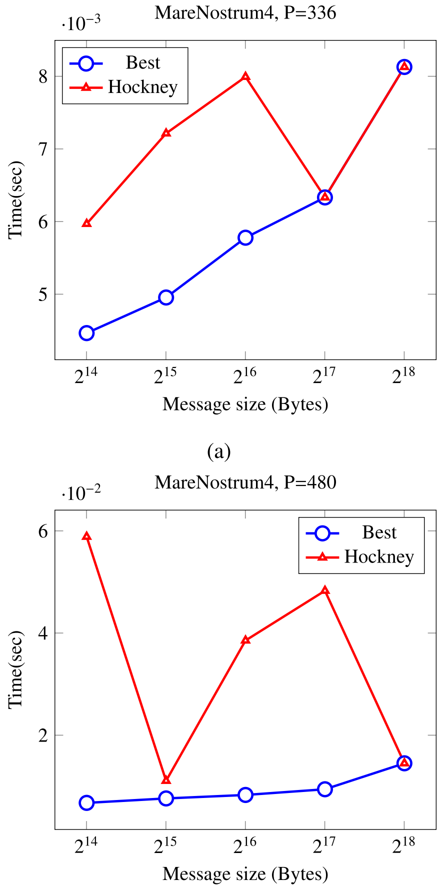
Fig. 1. The selection accuracy of the Open MPI decision routine for the MPI_Bcast collective operation on two Grid5000 clusters: Grisou (1a) and Gros (1b). Each data point on the blue graphs gives the execution time of the best broadcast algorithm for a given number of processes, P, and message size, m, available for selection in Open MPI. Each data point on the red graphs gives the execution time of the MPI_Bcast operation (in this case, the broadcast algorithm is automatically selected at runtime using the Open MPI decision function). The number of processes, P, executing broadcasts, is fixed to 600. The message size varies from 16 KB to 4 MB. The one-process-per-core configuration is used in the experiments.

The main advantage of this solution is its efficiency. The process of selection of the algorithm is very fast and does not affect the performance of the program. The main disadvantage of the state-of-the-art decision functions is that they do not guarantee the optimal selection in all situations. This is illustrated in Fig. 1 showing results of experiments with Open MPI on two clusters. On one of the clusters, the Open MPI broadcast routine selects non-optimal algorithms for messages smaller than 512 KB. On the other cluster, non-optimal algorithms are selected for messages larger than 256 KB. In both cases, this results in significant, multi-fold, performance degradation of the MPI_Bcast operation for a wide range of message sizes.

As a more accurate but equally efficient alternative to the use of empirical decision functions, the use of analytical performance models of collective algorithms for the selection process has been considered. The first detailed study of this approach was conducted by Pjevsivac-Grbovic et al. in [31]. However, this work as well as other early works in that direction [8,40,7] were not successful. The analytical models derived in these works were not able to accurately compare the relative performance of collective algorithms.

Recently, the model-based approach was revisited, and a novel method using analytical performance models for selection of optimal collective algorithms was proposed and proved both accurate and efficient for the one-process-per-CPU configuration of MPI applications [28]. The method proposes two innovations: (i) it derives the analytical performance models of algorithms from their implementation rather than from high-level mathematical definitions, and (ii) it estimates the parameters of the models separately for each algorithm. The first innovation results in much more detailed and realistic models, while the second one further improves their accuracy by tuning the model parameters, including the parameters of point-to-point communications, to the context of each algorithm.

While proved to be accurate for one-process-per-CPU MPI applications, this method fails for one-process-per-core configurations of MPI applications on modern multicore clusters. This is illustrated in Fig. 2. The reason is that this method is network-topology oblivious and uses the Hockney model [16] for point-to-point communication modeling, resulting in analytical models of collective algorithms that do not account for network congestion. While the effects of network congestion are not very significant for one-process-per-CPU MPI programs on modern platforms, they become much more impactful for one-process-per-core configurations. Therefore, to improve the selective accuracy of analytical models of collective algorithms in this case, the network topology



(a)

(b)

Fig. 2. The selection accuracy of the method [28] in the case of one-process-per-core Open MPI programs on the MareNostrum4 cluster. Each point on the blue graphs gives the execution time of the best broadcast algorithm available for selection in Open MPI. Each point on the red graphs gives the execution time of the broadcast algorithm selected using the method [28].

and more accurate point-to-point communication models must be considered.

In this paper, we revisit the model-based approach [28] and propose several innovations significantly improving the selective accuracy of analytical models to the extent that allows them to be used for accurate selection of optimal broadcast algorithms for one-process-per-core MPI applications on multi-core clusters. The contributions of this work are as follows:

- A novel method for runtime selection of optimal collective algorithms for collective communication operations in MPI applications running on multi-core clusters, based on analytical performance models of the collective algorithms, and application of this method to the Open MPI broadcast operation.
- Novel analytical performance models of the broadcast algorithms implemented in Open MPI. The models are derived from their implementation code and take into account the structure of the target multi-core cluster by representing each point-to-point transmission as a sequence of transfers via shared memory and network channels.
- A novel method for estimation of the model parameters, which finds them separately for each broadcast algorithm. The method is based on a careful design of the communication experiments, resulting in a system of linear equations with model parameters as unknowns. According to the method, the execution time of each experiment must be dominated by the execution time of the corresponding broadcast algorithm.
- Experimental validation of the selection accuracy of the proposed method on the MareNostrum4 and Grid5000 platforms.

The rest of the paper is organized as follows. Section 2 presents related work on analytical performance modeling of the collective algorithms, measurement of model parameters, and selection of optimal collective algorithms. Section 3 describes broadcast algorithms implemented in Open MPI. In section 4 we derive performance models of broadcast algorithms implemented in Open MPI. Section 5 describes the methodology of measurement of model parameters. Section 6 presents experimental validation on Grid5000 and MareNostrum4 clusters. Section 7 discusses limitations of the work and how they can be mitigated. Section 8 concludes the paper.

2. Related work

In this section, we first overview the state-of-the-art in analytical communication performance modeling and measurement of model parameters (a more detailed survey can be found in [39]). Then, we briefly discuss the use of analytical models in the selection problem.

2.1. Analytical performance models of MPI collective algorithms

All analytical models of collective algorithms use point-to-point communication models as building blocks. The most popular point-to-point communication models used in collective models are the Hockney model [16], LogP [9], LogGP [3], PLogP [21] and τ-Lop [38,36].

Hockney model represents a point-to-point message transmission in a homogeneous platform as $T(m) = \alpha + \beta \cdot m$, where m is the size of the message, and α and β are the latency and inverse bandwidth of the network respectively. The model parameters, α and β, are assumed to have the same value for all algorithms, message sizes and numbers of processes. This simple model has been extensively used in modeling collective algorithms. Thakur et al. [40] propose analytical performance models of several collective algorithms for *MPI_Allgather, MPI_Bcast, MPI_Alltoall, MPI_Reduce_scatter, MPI_Reduce*, and *MPI_Allreduce* routines. Chan et al. [8] build analytical performance models of *Minimum-spanning tree* algorithms and *Bucket* algorithms for MPI_Bcast, MPI_Reduce, MPI_Scatter, MPI_Gather, MPI_Allgather, MPI_Reduce_scatter, MPI_Allreduce collectives and later extend this work for multi-dimensional mesh architecture in [7]. An analytical performance model of a new *reduction* algorithm is proposed for a non-power-of-two number of processes by Rabenseifner et al. [35].

Culler et al. [10] propose the LogP model with the parameters L, the upper bound on the network latency, o_s, the overhead of processor involving sending a message, o_r, the overhead of processor involving receiving a message, and g, the gap between consecutive message transmission. LogGP model extends LogP with the parameter G representing the gap per byte in sending a message. Kielmann et al. [21] propose the PLogP (Parametrized LogP) model. PLogP defines model parameters, except for latency L, as functions of message size, in order to improve accuracy.

A detailed study of the performance of collective operations using the above analytical performance models (Hockney, LogP/LogGP, and PLogP) is conducted by Pjevsivac-Grbovic et al. [31]. They study the feasibility of selection of optimal collective algorithms for *barrier*, *broadcast*, *reduce* and *alltoall* using their analytical performance models. Additionally, the splitted-binary broadcast algorithm has been designed and analyzed with different performance models in this work. The models used in the study were built following the traditional approach using high-level mathematical definitions of the collective algorithms. After experimental validation of their modeling approach, the authors conclude that the proposed models are not accurate enough for selection of optimal algorithms.

A general analytical performance model for *tree-based* broadcast algorithms with message segmentation has been proposed by Patarasuk et al. [29]. Unlike traditional models, this model introduces a new parameter, *Maximum nodal degree* of the tree. The purpose of this model is restricted to theoretical comparison of different tree-based broadcast algorithms. Accurate prediction of the execution time of the broadcast algorithms and methods for measurement of the model parameters, including the maximal nodal degree of the tree, are out of the scope of this work.

All the above models of collective algorithms are built using their high-level theoretical description. The overall conclusion is that, while these models can be used for analysis of theoretical complexity of the algorithms, they are not accurate enough for the task of selection of optimal collective algorithms [40], [31]. The authors of [40], [31] also conclude that in order to improve the accuracy of their analytical models, we have to assume that the model parameters depend on the message size and the number of processes.

Recently, analytical communication performance models, adapted to current heterogeneous platforms, have been proposed. Lastovetsky et al. [26] propose the LMO point-to-point communication model based on Hockney model, Cameron et al. [6] analyze the performance of collective algorithms using a hardware-parameterized model, $log_n P$, which is based on the LogGP homogeneous model. The $mLog_n P$ model is a further extension of $log_n P$ proposed by Tu et al. [41]. The τ-Lop model proposed by Rico-Gallego et al. [37,38,36] includes the representation for concurrent transmissions and hence allows for a more accurate modeling of collective operations. These models include additional parameters to represent concurrency in communication channels, memory hierarchy and heterogeneity of communication channels. In this work, we use τ-Lop to model the contribution of point-to-point communications in the execution time of collective algorithms. The τ-Lop model is briefly introduced in section 4.

In our approach, we stick to the assumption of independence of model parameters on the message size and the number of processes. Instead, we improve the accuracy of the models of collective algorithms by deriving them from the implementation code. This work is based on our previous research [28], where a novel approach to the modeling of collective algorithms and to the measurement of model parameters is proposed. Unlike the traditional approaches, deriving analytical models of collective algorithms from their high-level mathematical definitions, it derives the models from their implementation code, taking into account all important implementation details affecting the performance. In addition, the approach relies on an elaborate method to measure the model parameters using carefully designed communication experiments involving all collective algorithms. Experimental results showed that analytical models taking into account implementation details of collective algorithms were able to compare the performance of collective algorithms accurately [28]. However, that work was limited to one-process-per-CPU configurations of MPI applications.

2.2. Measurement of model parameters

One use of analytical communication performance models is for theoretical analysis of the complexity of collective algorithms. In such purely theoretical studies, the authors do not pay much attention to methods of measurement of model parameters. However if a model is intended for accurate prediction of the execution time of the communication algorithm on each particular platform, and if it aims to be reproducible, a well-defined experimental measurement method of the model parameters will be as important as the theoretical formulation of the model. Different measurement methods may give significantly different values of the model parameters and therefore either degrade or improve the model's prediction accuracy.

Existing measurement methods predominantly rely on *point-to-point* communication experiments, which are used to obtain a system of *linear* equations with model parameters as unknowns. Hockney [16] presents a measurement method to find the α and β parameters of the Hockney model based on a set of communication experiments consisting of point-to-point round-trips. The sender sends a message of size m to the receiver, which immediately returns the message to the sender upon its receipt. The time $RTT(m)$ of this experiment is measured on the sender side and estimated as $RTT(m) = 2 \cdot (\alpha + m \cdot \beta)$. These round-trip communication experiments for a wide range of message size m produce a system of linear equations with α and β as unknowns. To find α and β from this system, the linear least-squares regression is used.

Culler et al. [10] propose a method of measurement of parameters of the LogP model that relies on the Active Messages (AM) protocol [12]. The method consists of four individual communication experiments for the four parameters of the model based on sending/receiving zero-sized transmissions and round-trip time messages. Kielmann et al. [21] extend that method of measurement of parameters for their Parametrized LogP (PlogP) model. As most of the parameters of PLogP are defined as functions of message size, some communication experiments are repeated a range of message sizes.

Hoefler et al. [17] develop a method to measure parameters of the LogGP model. The building block of the method is a parametrized round-trip function $PRTT(N, d, m)$, which depends on parameters N, the number of messages to send, d, the delay in sending the messages, and m, the size of the messages. Properly varying these parameters, the method obtains equations with the LogGP model parameters as unknowns.

From this overview, we can conclude that the state-of-the-art methods for measurement of parameters of communication performance models are all based on *point-to-point communication experiments*, which are used to derive a system of equations involving model parameters as unknowns. An exception from this rule is a method for measurement of parameters of the LMO heterogeneous communication model [24,25]. LMO is a communication model of a heterogeneous cluster, and the total number of its parameters is significantly larger than the maximum number of independent point-to-point communication experiments that can be designed to derive a system of independent linear equations. To address this problem and obtain the sufficient number of independent linear equations involving model parameters, the method additionally introduces simple collective communication experiments, each using three processors and consisting of a one-to-two communication operation (scatter) followed by a two-to-one communication operation (gather). The experiments are implemented using the MPIBlib library [27]. This method however is not designed to improve the accuracy of predictive analytical models of communication algorithms.

Based on the idea developed in the LMO model, Rico-Gallego et al. [38] propose a detailed method for measurement of parameters of the τ-Lop model on a multi-core cluster. For each communication channel, *shared memory* or *network*, experimental measurement of both $o^c(m)$ and $L^c(m, \tau)$ are designed separately. For the transfer time (L), the method includes measurements obtained from collective operations, such as broadcast, to improve the accuracy of the resultant linear equation system.

In this work, we use carefully designed *collective communication experiments* in the measurement method in order to improve the predictive accuracy of analytical models of collective algorithms.

2.3. Selection of collective algorithms using machine learning algorithms

Machine learning (ML) techniques have also been tried to solve the problem of selection of optimal MPI algorithms.

In [30], applicability of the quadtree encoding method to this problem is studied. The goal of this work is to select the best performing algorithm and segment size for a particular collective on a particular platform. The approach is based on the following steps. (1) Collective algorithms are executed on a particular platform to collect detailed performance data. (2) The decision map is built for the collective on a particular platform by analyzing the performance data. It is assumed that the decision map covers all message and communicator sizes. (3) The quadtree is initialized using the decision map. (4) The decision function source code is generated from the initialized quadtree. For example, Linear tree, Binary tree, Binomial tree, Split-Binary, and Chain tree broadcast algorithms are profiled with a maximum of 50 processes. The experimental results show that mean performance penalty reaches 74% and 37% and maximum performance penalty reaches 391% and 743% on different platforms respectively. While the study shows some level of applicability of the quadtree encoding algorithm to the problem, collection of detailed profiling data of collectives for all message sizes and communicator sizes is a very expensive procedure. Besides, for some message sizes and communicator sizes the penalty of the decision function is too high. Taking into account that decision trees are considered weak learners [11], the decision function will perform poorly on unseen data.

Applicability of the C4.5 algorithm to the MPI collective selection problem is explored in [32]. The C4.5 algorithm [33] is a decision tree classifier, which is employed to generate a decision function, based on a detailed profiling data of MPI collectives. The same steps are followed to build the decision tree using the C4.5 algorithm as in the quadtree encoding method presented above. The same weaknesses are shared by the decision trees built by the quadtree encoding algorithm and by the C4.5 algorithm. While the accuracy of the decision function built by the C4.5 classification algorithm is higher than that of the decision function built by quadtree encoding algorithm, still, the performance penalty is higher than 50%.

Most recently Hunold et al. [19] studied the applicability of six different ML algorithms for selection of optimal MPI collective algorithms. The basic idea of their approach is to create a regression model for every collective algorithm that is available for a given collective operation, predicting the execution time of the collective algorithm. The constructed regression models are then used at run time to select the algorithm that minimizes the execution time for unseen configurations. The ML algorithms employed to build the regression models are Random Forests, Neural Networks, Linear Regressions, XGBoost, K-nearest Neighbor, and generalized additive models (GAM). The configuration is characterized by the message size, the number of nodes, and the number of processes per node. The approach is evaluated using MPI_Bcast, MPI_Allreduce and MPI_Alltoall collectives. In the experimental evaluation, the number of nodes varies between 4 and 36, and the number of processes per node varies between 1 and 32. The experimental results show two things. First, it is very expansive and difficult to build a regression model even for a relatively small cluster. There is no clear guidance on how to do it to achieve better results. Second, even the best regression models do not accurately predict the fastest collective algorithm in most of the reported cases. Moreover, in many cases the selected algorithm performs worse than the default algorithm, that is, the one selected by a simple native decision function.

To the best of the authors' knowledge, the works outlined in this subsection are the only research done in MPI collective algorithm selection using ML algorithms. The results show that the selection of the optimal algorithm without any information about the semantics of the algorithm yields inaccurate results. While the ML-based methods treat a collective algorithm as a black box, we derive its performance model from the implementation code and estimate the model parameters using statistical techniques. The limitations of the application of the statistical techniques (AI/ML) to collective performance modeling and selection problem can be found in a detailed survey [43].

3. Broadcast algorithms implemented in Open MPI

Open MPI architecture is based on software components, plugged into the library kernel. A component provides functionality with specific implementation features. For instance, a collective component known as *Tuned* implements different algorithms for each collective operation defined in MPI as a sequence of point-to-point transmissions between the involved processes. A *communicator* provides an isolated communication context for the group of processes executing the collective operation. Processes in a communicator are identified by an assigned *rank* integer number, starting at 0.

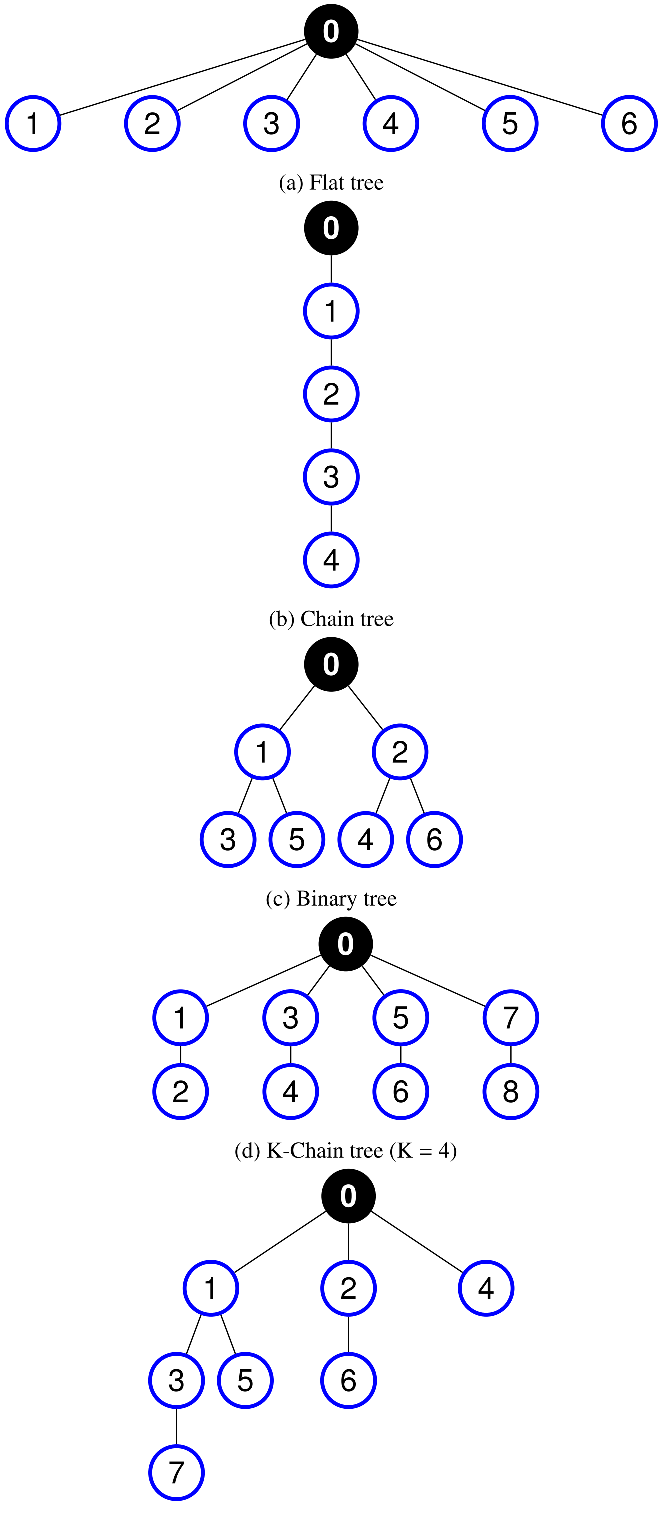
In the broadcast operation (`MPI_Bcast`) a process called *root* sends a message with the same data to all processes in the communicator. Messages can be segmented in transmissions. *Segmentation* of messages is a common technique used for increasing the communication parallelism by avoiding the *rendezvous* protocol, and hence, improving the performance. It consists of dividing up the message into smaller fragments called segments and sending them in sequence.

Every algorithm implementing the broadcast in the *Tuned* component defines a communication graph with a specific topology between the P ranks in the communicator. Ranks are the nodes in the graph, and they are mapped to the processes of the parallel machine. The features and topology of the broadcast algorithms implemented in Open MPI *Tuned* component are listed below:

- **Flat tree algorithm**. The algorithm employs a single level tree topology shown in Fig. 3a where the root node has $P-1$ children. The message is transmitted to child nodes without segmentation.
- **Chain tree algorithm**. Each internal node in the topology has one child (see Fig. 3b). The message is split into segments and transmission of segments continues in a pipeline until the last node gets the broadcast message. ith process receives the message from the $(i-1)$th process, and sends it to $(i+1)$th process.
- **Binary tree algorithm**. Unlike the chain tree, each internal process has two children, and hence data is transmitted from each node to both children (Fig. 3c). Segmentation technique is employed in this algorithm. For simplicity we assume that the binary tree is complete, then $P=2^H-1$ where H is the height of the tree, $H=\log_2(P+1)$.
- **Split binary tree algorithm**. The split binary tree algorithm employs the same virtual topology as the binary tree (Fig. 3c). As the name implies, the difference from the binary tree algorithm is that the message is split into two halves before transmission. After splitting the message, the right and left halves of the message are pushed down into the right and left sub-trees respectively. In an additional last phase, the left and right nodes exchange in pairs their halves of the message to complete the broadcast operation.

- **K-Chain tree algorithm**. The *K-Chain* virtual topology is employed in the algorithm (Fig. 3d). The root broadcasts the message using segmentation to the child processes, and then the child processes broadcast the message to their children in parallel. As the name implies, the virtual topology consists of K *chain* tree virtual topology each of which is connected to *root*. The height of *K-chain tree* is estimated as $H = \lfloor \frac{P-1}{K} \rfloor$. Last process must wait for $H_{k-chain}$ steps until it gets the broadcast message. Rank of processes is mapped into K-Chain tree virtual topology using following formula, $\sum_{k=0}^{K-1} \sum_{i=0}^{H-1} (H \cdot k + i + 1)$
- **Binomial tree algorithm**. The binomial tree topology is determined according to the binomial tree definition [20]. The algorithm employs *balanced* binomial tree (Fig. 3e). Unlike the binary tree, the maximum nodal degree of the binomial tree decreases from the root down to the leaves as follows: $\lceil \log_2 P \rceil$, $\lceil \log_2 P \rceil - 1$, $\lceil \log_2 P \rceil - 2, \ldots$. The height of the binomial tree is the order of the tree, $H = \lfloor \log_2 P \rfloor$.

In the following section, we build performance models for the broadcast algorithms described above.



(a) Flat tree

(b) Chain tree

(c) Binary tree

(d) K-Chain tree (K = 4)

(e) Balanced binomial tree

Fig. 3. Virtual topologies used by broadcast algorithms in Open MPI.

4. Modeling of Open MPI broadcast algorithms on multi-core cluster

We build new analytical performance models of broadcast algorithms described in Section 3 for multi-core clusters. For point-to-point communication modeling, we use the τ-Lop model.

τ-Lop takes into account the distinctive features of the communication channels to represent a point-to-point transmission as a sequence of transfers via shared memory or network. In this approach, we use the τ-Lop model, that it estimates the time of sending a point-to-point message in s transfers as $T^{c}_{p2p}(m) = o^{c}(m) + \sum_{i=1}^{s} L^{c_i}(m)$, where c represents the communication channel, and the o and L parameters represent the overhead of the communication protocol and the transfer time respectively. Hence, τ-Lop considers a different representation for a message transmitted through shared memory ($c = 0$) and network channel ($c = 1$). For instance, through shared memory, MPI libraries default transmission is through a *shared intermediate buffer* between sender and receiver processes, hence two identical transfers ($s = 2$) are needed to transmit the message, and previous expression reduces to $T^{0}_{p2p}(m) = o^{0}(m) + 2L^{0}(m)$. While, through a network, representation depends on the network capabilities. For instance, in an Ethernet network we consider two shared memory transfers, from sender memory to NIC and from receiver NIC to destination memory, and a network transfer between NICs, as $T^{1}_{p2p}(m) = o^{1}(m) + 2L^{0}(m) + L^{1}(m)$.

Most of the Open MPI broadcast algorithms are implemented using message segmentation, except for the flat tree broadcast algorithm. For segmented broadcast algorithms, we assume that $m = n_s \cdot m_s$, where n_s and m_s are the number of segments and the segment size respectively. In this paper, we assume the same fixed segment size in all segmented algorithms.

4.1. Flat tree algorithm

In Open MPI, the linear broadcast algorithm is implemented using non-blocking *send* and blocking *receive* operations. The algorithm transmits the whole message from the root to the leaves without message segmentation. Because of non-segmented message transmission and assuming the *rendezvous* protocol, each next *send* only starts after the previous one has been completed. Therefore, the execution time of the linear tree broadcast algorithm will be equal to the sum of execution times of $P - 1$ send operations:

$$T_{BFT}(P, m) = \sum_{i=1}^{P-1} T^{c_i}_{p2p}(m), \tag{1}$$

where $T^{c_i}_{p2p}$ is the point-to-point communication time through a channel c_i, connecting the root and the i-th process, estimated using the τ-Lop model. In the rest of the paper, we use BFT to refer to the blocking flat tree broadcast algorithm.

Algorithm 1 Tree-based segmented broadcast algorithm.

```
if $(rank == root)$ then
    // Send segments to all children
    for $i \in 0..n_s - 1$ do
        for $child \in list\ of\ children$ do
            MPI_Isend(segment[$i$], child, ... )
        end for
        MPI_Waitall(. . .)
    end for
else if $(intermediate\ nodes)$ then
    for $i \in 0..n_s - 1$ do
        // Post receive and wait
        MPI_Irecv($segment[i]$)
        MPI_Wait(. . .)
        // Send data to children
        for $child \in list\ of\ children$ do
            MPI_Isend($segment[i]$, $child$, ... )
        end for
        MPI_Waitall($children$)
    end for
else if $(leaf\ nodes)$ then
    // Receive all segments from parent in a loop
    for $i \in 0..n_s - 1$ do
        MPI_Irecv($segment[i]$, ... )
        MPI_Wait(. . .)
    end for
end if
```

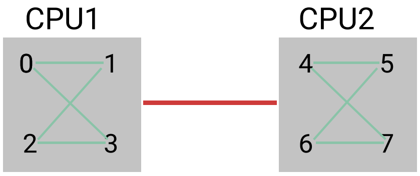


Fig. 4. Cluster of two quad-core processors. Processes running on the same processor use the shared-memory for point-to-point communications (green links). Processes running on different processors use the network for point-to-point communications (the red link).

Every time the Open MPI *MPI_Bcast* operation is invoked with a specific root, an internal tree with the specific virtual topology for the chosen algorithm is built, and then, the algorithm is executed. This internal tree is used as a building block in tree-based segmented broadcast algorithms implementing *MPI_Bcast*, namely, in the *binomial tree*, *binary tree*, *split binary tree*, *k-chain tree*, and *chain tree* broadcast algorithms (see Algorithm 1 for more details). That tree algorithm is composed of flat trees using *non-blocking* send and receive operations, which we refer to as Non-Blocking Flat Trees (NBFTs).

As illustrated in Fig. 5, NBFT can use either one of the two available channels for all point-to-point communications or both of them. The number of network point-to-point communications, C, in an NBFT can be calculated as follows,

$$C = \sum_{i=1}^{P-1} c_i \tag{2}$$

Obviously, $0 \le C \le P - 1$. Therefore, the number of point-to-point communications through shared memory in the NBFT will be equal to $P - C - 1$. The time of message transmission through a network channel is longer than through shared memory. In our model, we assume that

$$T^1_{p2p}(m) = Q(m) \cdot T^0_{p2p}(m), \tag{3}$$

where $Q(m)$ is a platform-dependent parameter representing the ratio of delays of the communication channels ($Q(m) > 1$). We denote $T^c_{NBFT}(P, m)$ the execution time of an NBFT, which uses only one channel, c, for all message transmissions. The execution time of an arbitrary NBFT, $T_{NBFT}(P, C, m)$, which can use both available channels, is modeled as follows,

$$T_{NBFT}(P, C, m) = \begin{cases} T^0_{NBFT}(P, m), \text{ if } C = 0 \\ T^1_{NBFT}(C + \lfloor \frac{P-C-1}{Q(m)} \rfloor + 1, m), \text{otherwise.} \end{cases} \tag{4}$$

Thus, in our model the execution time of any NBFT **A** using two channels (shared memory ($c = 0$) and network ($c = 1$)) will be calculated as the execution time of the NBFT **B**, which only uses the network channel and is obtained from **A** by formal replacement of each group of $Q(m)$ shared-memory transmissions by one network transmission.

It is evident from Algorithm 1 that NBFTs are only used in Open MPI to transmit segments of the same fixed size, m_s, which is therefore not a variable in our model.

The execution time of the NBFT broadcasting a message of size m_s through channel c, $T^c_{NBFT}(P, m_s)$, can be bounded as follows,

$$T^c_{p2p}(m_s) \le T^c_{NBFT}(P, m_s) \le T^c_{BFT}(P, m_s). \tag{5}$$

The lower bound represents the case of purely parallel point-to-point communications, while the upper bound - the case of purely serial point-to-point communications.

From formula (1), we can derive

$$T^c_{BFT}(P, m_s) = \sum_{i=1}^{P-1} T^c_{p2p}(m_s) = (P - 1) \cdot T^c_{p2p}(m_s). \tag{6}$$

Hence,

$$T^c_{p2p}(m_s) \le T^c_{NBFT}(P, m_s) \le (P - 1) \cdot T^c_{p2p}(m_s). \tag{7}$$

Therefore, we approximate $T^c_{NBFT}(P, m_s)$ as follows,

$$T^c_{NBFT}(P, m_s) = \gamma^c(P) \cdot T^c_{p2p}(m_s), \tag{8}$$

where $\gamma^c(P)$ is a *parallelization factor*, representing the increase in the cost of $P - 1$ overlapping non-blocking transmissions, originating from the same root, of a segment of size m_s through the channel c in the NBFT, with respect to a single point-to-point transmission ($1 \le \gamma^c(P) \le P - 1$).

4.2. Binomial tree algorithm

In Open MPI, the binomial tree broadcast algorithm is segmentation-based and implemented as a combination of flat tree broadcast algorithms using non-blocking *send* and *receive* operations, NBFTs. Fig. 5 shows the broadcast binomial tree with eight processes ($P = 8$) mapped to the cores of the cluster of two quad-core processors shown in Fig. 4. It also details the stages of execution of the binomial tree algorithm when the number of segments, $n_s = \frac{m}{m_s}$, is equal to 3. Each stage consists of parallel execution of several NBFTs. Therefore, the execution time of each stage will be equal to the maximum execution time of its NBFTs. We assume that the stages of the algorithm are executed serially. Therefore, the total execution time of the algorithm will be equal to the sum of the execution times of its stages.

Open MPI employs the *balanced* binomial tree for the binomial tree broadcast algorithm. The height of the balanced binomial tree is equal to $\lfloor log_2 P \rfloor$. Therefore, the algorithm will be completed in $\lfloor log_2 P \rfloor + \frac{m}{m_s} - 1$ stages. Thus, the time to complete the binomial tree broadcast algorithm can be estimated as follows,

$$T_{binomial}(P, m, m_s) = \sum_{i=1}^{\lfloor log_2 P \rfloor + \frac{m}{m_s} - 1} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right), \tag{9}$$

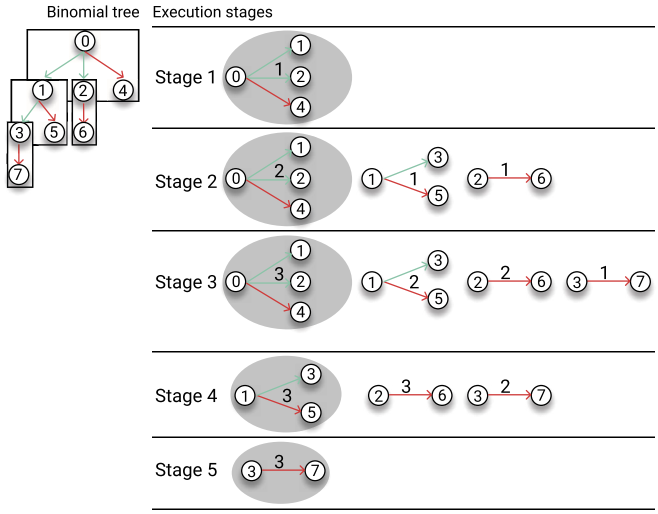


Fig. 5. The topology and execution stages of the binomial broadcast algorithm employing the non-blocking flat tree (NBFT) broadcasts on a cluster of two quad-core processors ($P = 8$) shown in Fig. 4. The message is split up into $n_s = 3$ segments. Each arrow in an NBFT represents transmission of a segment through a shared memory (green) or network (red) channel. The number over the arrow gives the index of the broadcast segment and ranges from 1 to 3. The execution time of each stage is equal to the execution time of its longest NBFT, which is encircled in a gray-colored oval.

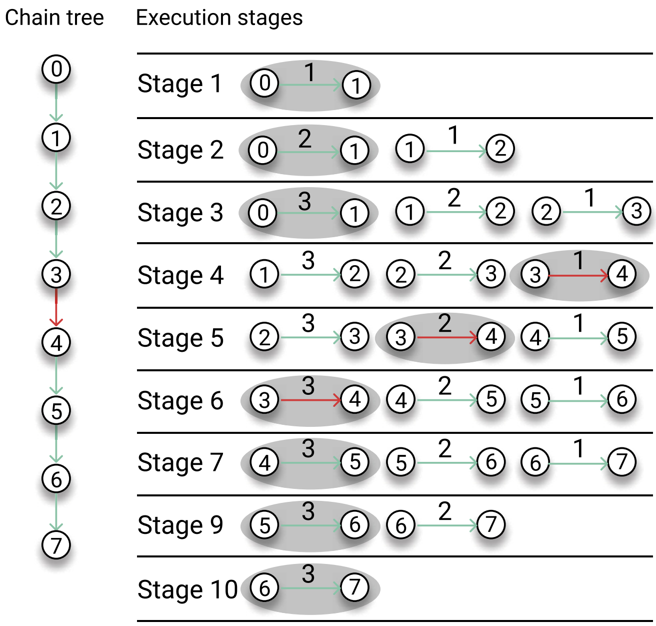


Fig. 6. The topology and execution stages of the chain broadcast algorithm employing NBFT broadcasts on the cluster of two quad-core processors ($P = 8$) from Fig. 4. The message is split up into $n_s = 3$ segments. Each arrow in an NBFT represents transmission of a segment through a shared memory (green) or network (red) channel. The number over the arrow is the index of the broadcast segment. The execution time of each stage is equal to the execution time of its longest NBFT, which is encircled in a gray-colored oval.

where P_{ij_i} $(1 \le P_{ij_i} \le \lceil log_2 P \rceil)$ is the number of nodes in the j_i-th $NBFT$ running at i-th stage. The number of parallel NBFTs is varying from stage to stage but upper bounded by $2 + (\lfloor log_2 P \rfloor - 1) \cdot (\lfloor log_2 P \rfloor - 2)$.

4.3. Chain tree algorithm

In Open MPI, the chain tree algorithm is segmentation-based and implemented using non-blocking point-to-point communication. While the height of the chain tree equals $P - 1$, the algorithm will be completed in $P + \frac{m}{m_s} - 2$ stages, each consisting of a varying number of concurrent NBFTs. All the NBFTs will have exactly two nodes and therefore be equivalent to non-blocking point-to-point communications. Fig. 6 illustrates the stages of execution of the chain tree algorithm on the cluster of two quad-core processors (Fig. 4) when the broadcast message is split into three segments ($n_s = \frac{m}{m_s} = 3$). Each stage consists of parallel execution of several NBFTs. Therefore, the execution time of each stage will be equal to the maximum execution time of its NBFTs. As we assume sequential execution of the stages, the total execution time of the algorithm will be equal to the sum of the execution times of its stages. Thus, the execution time of the chain tree algorithm can be estimated as

$$T_{chain}(P, m, m_s) = \sum_{i=1}^{P+\frac{m}{m_s}-2} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right), \quad (10)$$

where the number of nodes in the j_i-th $NBFT$ running at i-th stage, P_{ij_i}, is always equal to 2. The number of parallel NBFTs is varying from stage to stage, starting from 1 for stage 1, incrementally growing to $P - 1$ for middle stages, and then incrementally decreasing to 1 for the last, $(P + \frac{m}{m_s} - 2)$-th, stage.

4.4. Binary tree algorithm

In Open MPI, the binary tree broadcast algorithm is segmentation-based and uses the balanced binary tree topology (see Fig. 3c). The root broadcasts each segment to its children using the NBFT. Upon receipt of the next segment, each internal node acts similarly. As the height of the balanced binary tree is equal to $\lfloor \log_2 P \rfloor$, the algorithm will be completed in $\lfloor \log_2 P \rfloor + \frac{m}{m_s} - 1$ stages, each consisting of a varying number of concurrent NBFTs. Fig. 7 illustrates the stages of execution of the binary tree algorithm on the cluster of two quad-core processors (Fig. 4) when the broadcast message is split into three segments ($n_s = \frac{m}{m_s} = 3$). Each stage consists of parallel execution of several NBFTs. Therefore, the execution time of each stage will be equal to the execution time of its longest NBFT. As we assume sequential execution of the stages, the execution time of the algorithm will be equal to the sum of the execution times of the stages. Therefore,

$$T_{binary}(P, m, m_s) = \sum_{i=1}^{\lfloor \log_2 P \rfloor + \frac{m}{m_s} - 1} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right), \quad (11)$$

where the number of nodes in the j_i-th $NBFT$ running at the i-th stage, P_{ij_i}, is either 3 or 2. The number of parallel NBFTs is varying from stage to stage, starting from 1 for stage 1 and reaching a maximum of $2^{\lfloor log_2 P \rfloor}$ for middle stages.

4.5. K-chain tree algorithm

In Open MPI, the K-chain tree algorithm is implemented using non-blocking communication and message segmentation. In the K-chain tree, the root node has $K(K > 1)$ children, while the internal nodes have a single child each (Fig. 3d). As the height of the tree is $\lfloor \frac{P-1}{K} \rfloor$, the algorithm takes $\lfloor \frac{P-1}{K} \rfloor + \frac{m}{m_s} - 1$ stages to complete. As we assume sequential execution of the stages, the execution time of the algorithm will be equal to the sum of the execution times of the stages. Therefore,

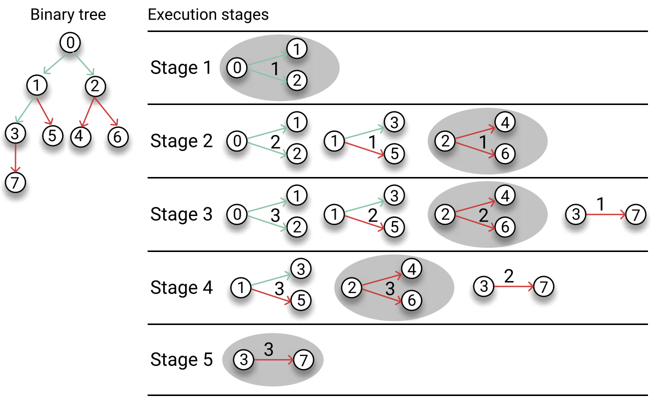


Fig. 7. The topology and execution stages of the binary broadcast algorithm employing the non-blocking flat trees (NBFTs) broadcasts on the cluster of two quad-core processors ($P = 8$) from Fig. 4. The message is split up into $n_s = 3$ segments. Each arrow in an NBFT represents transmission of a segment through the shared memory (green) or network (red) channel. The number over the arrow gives the index of the broadcast segment. The execution time of each stage is equal to the execution time of its longest NBFT, which is encircled in a gray-colored oval.

$$T_{kchain}(P, m, m_s) = \sum_{i=1}^{\lfloor \frac{P-1}{K} \rfloor + \frac{m}{m_s} - 1} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right), \tag{12}$$

where the number of nodes in the j_i-th $NBFT$ running at the i-th stage, P_{ij_i}, is either $K + 1$ or 2. The number of parallel NBFTs is varying from stage to stage, starting from 1 for stage 1 and reaching a maximum of $P - K - 1$ for middle stages.

4.6. Split-binary tree algorithm

In Open MPI, the split-binary tree algorithm is segmentation-based and implemented using blocking *send* and non-blocking *receive* routines. This is the difference from other segmented tree-based broadcast algorithms which all use non-blocking standard-mode send. However, because in Open MPI the segment size m_s is selected so that the blocking sends in the split binary tree will be executed in the buffered mode, we approximate the execution time of all flat tree broadcast algorithms in the split binary tree algorithm by the execution time of an NBFT.

The split binary tree algorithm consists of two phases – *forwarding* and *exchange*. In the first phase, the message of size m is split into two equal parts in the root, which are then sent to the left and right subtrees respectively using message segmentation.

After completion of the first phase, each node in the left subtree contains the first half of the message and each node in the right subtree – the second half of the message. Because of segmentation, each node will receive $\frac{m}{2 \cdot m_s}$ segments during the first phase.

As the balanced binary tree virtual topology is employed in the split-binary tree algorithm, each node in the left subtree will have a matching pair in the right subtree and vice versa. In the second phase, each pair of matching nodes in the left and right subtrees exchange their halves of the message. The execution time of the split-binary tree broadcast will be equal to the sum of the execution times of the first and second phases. As the height of the balanced binary tree is equal to $\lfloor \log_2 P \rfloor$, the first phase will be completed in $\lfloor log_2 P \rfloor + \frac{m}{2m_s} - 1$ stages. While in the second phase each pair of matching nodes sends/receives message at the same time, the execution time of the second phase will be equal to $\max_{c_k} T^{c_k}_{p2p}\left(\frac{m}{2}\right)$ where $1 \le k \le \lfloor \frac{P-1}{2} \rfloor$ and $c_k \in \{0, 1\}$. Thus, as we assume sequential execution of stages at the forwarding phase, the time to complete the split-binary tree algorithm can be estimated as follows:

$$T_{split\_binary}(P, m, m_s) = \sum_{i=1}^{\lfloor \log_2 P \rfloor + \frac{m}{2m_s} - 1} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right) + \max_{c_k} \left\{ T^{c_k}_{p2p}\left(\frac{m}{2}\right) \right\}, \tag{13}$$

where the number of nodes in the j_i-th $NBFT$ running at the i-th stage of the forwarding phase, P_{ij_i}, is either 3 or 2. The number of parallel NBFTs at the forwarding phase is varying from stage to stage, starting from 1 for stage 1 and reaching a maximum of $2^{\lfloor log_2 P \rfloor}$ for middle stages.

5. Estimation of algorithm specific and platform dependent model parameters

As presented in Section 4, the analytical models of the Open MPI broadcast algorithms use the τ-Lop point-to-point model parameters, the ratio of delays of the communication channels $Q(m)$, and the parallelization factor $\gamma^c(P)$ as the model parameters. The traditional state-of-the-art approach to estimation of model parameters would be to find these parameters from a number of point-to-point communication experiments. Namely, the time of a round-trip of a message of size m, $RTT(m)$, is measured for a wide range of m. From these experiments, a system of linear equations with unknown model parameters is derived. Then, linear regression is applied to find model parameters.

This approach yields a unique single set of parameters for each target platform. Unfortunately, with model parameters found this way, not all our analytical formulas will be accurate enough to be used for accurate selection of the best performing broadcast algorithm. Using non-linear regression does not improve the situation as the function $RTT(m)$ is typically near linear. Therefore, we propose to estimate the model parameters separately for each broadcast algorithm. More specifically, we propose to design a specific communication experiment for each broadcast algorithm, so that the algorithm itself would be involved in the execution of the experiment. Moreover, the execution time of this experiment must be dominated by the execution time of this broadcast algorithm. Then, we conduct a number of experiments on the target platform for a range of numbers of processors and message sizes. From those experiments, we can derive a sufficiently large number of equations with unknown model parameters, and then use an appropriate solver to find their values.

Our approach to this problem is the following. We consider $\gamma^c(P)$ and $Q(m)$ platform-specific but algorithm-independent parameters and design a separate communication experiment for their estimation. The values of $\gamma^c(P)$ and $Q(m)$ found from this experiment are then used as known constants in the algorithm-specific systems of equations for the τ-Lop model parameters. We present this approach in Sections 5.1 and 5.2. The motivation behind assuming $\gamma^c(P)$ and $Q(m)$ algorithm-independent is that otherwise they would appear in the derived equations as unknowns and make the equations non-linear.

5.1. Estimation of $\gamma^c(P)$ and $Q(m)$

The model parameter $\gamma^c(P)$ represents the increase in the cost of the overlapping $P - 1$ non-blocking transmissions through the NBFT with respect to a single point-to-point message transmission. The NBFT is only used for broadcasting of a segment in the tree-based segmented broadcast algorithms. Thus, in the context of

$$\begin{cases} \lambda_{11} \cdot o^0 + \lambda_{12} \cdot o^1 + \lambda_{13} \cdot L^0 + \lambda_{14} \cdot L^1 = T_1 \\ \lambda_{21} \cdot o^0 + \lambda_{22} \cdot o^1 + \lambda_{23} \cdot L^0 + \lambda_{24} \cdot L^1 = T_2 \\ \dots \\ \lambda_{M1} \cdot o^0 + \lambda_{M2} \cdot o^1 + \lambda_{M3} \cdot L^0 + \lambda_{M4} \cdot L^1 = T_M \end{cases}$$

Fig. 8. A system of M linear equations with o^0, o^1, L^0 and L^1 as unknowns, derived from M communication experiments, each consisting of the execution of the binomial tree broadcast algorithm, broadcasting a message of size m_i ($i = 1, ..., M$) from the root to the remaining $P-1$ processes, followed by the flat-without-synchronization gather algorithm, gathering messages of size m_s (segment size) on the root. The execution times, T_i, of these experiments are measured on the root.

Open MPI, the NBFT will always broadcast a message of size m_s to a relatively small number of processes.

According to Formula (8),

$$\gamma^c(P) = \frac{T^c_{NBFT}(P, m_s)}{T^c_{p2p}(m_s)} = \frac{T^c_{NBFT}(P, m_s)}{T^c_{NBFT}(2, m_s)}, \tag{14}$$

where $P \in \{2, ..., P^{max(c)}_{NBFT}\}$. $P^{max(1)}_{NBFT}$ is the maximum number of the processes communicating through the network channel in NBFTs. $P^{max(0)}_{NBFT}$ is the maximum number of the processes communicating through the shared memory channel where the NBFT only uses the shared memory channel. Therefore, in order to estimate $\gamma^c(P)$, we need a method for estimation of $T^c_{NBFT}(P, m_s)$. We use the following method:

- For each $2 \le P \le P^{max(c)}_{NBFT}$, we measure on the root the execution time $\theta^c(P, N)$ of N successive calls of the NBFT separated by barriers. The routine broadcasts a message of size m_s.
- We estimate $T^c_{NBFT}(P, m_s)$ as

$$T^c_{NBFT}(P, m_s) = \frac{\theta^c(P, N)}{N}.$$

The experimentally obtained discrete function $\frac{T^c_{NBFT}(P, m_s)}{T^c_{NBFT}(2, m_s)}$ is used as a platform-specific but algorithm-independent estimation of $\gamma^c(P)$.

Parameter $Q(m)$ only appears in the analytical models of the Open MPI broadcast algorithms in the context of NBFTs broadcasting a segment of fixed size m_s (see Section 4). Therefore, we only need to estimate $Q(m_s)$. By definition $T^c_{NBFT}(2, m_s) = T^c_{p2p}(m_s)$, and using formula (3) we estimate $Q(m_s)$ as

$$Q(m_s) = \frac{T^1_{NBFT}(2, m_s)}{T^0_{NBFT}(2, m_s)} \tag{15}$$

5.2. Estimation of algorithm specific model parameters

To estimate the model parameters for a given broadcast algorithm, we design a communication experiment, which starts and finishes on the root (in order to accurately measure its execution time using the root clock), and involves the execution of the modeled broadcast algorithm so that the total time of the experiment would be dominated by the time of its execution. In this section, we present the estimation of τ-Lop model parameters.

For all broadcast algorithms, the communication experiment consists of a broadcast of a message of size m (where m is a multiple of segment size m_s), using the modeled broadcast algorithm, followed by the *flat-without-synchronization* gather algorithm. This algorithm works by gathering messages of size m_s on the root. The execution time of this experiment on P nodes, $T_{comm\_experiment}(P, m)$, can be estimated as follows,

$$T_{comm\_experiment}(P, m) = \\ T_{bcast}(P, m) + T_{flat\_gather}(P, m_s). \tag{16}$$

The execution time of the flat-without-synchronization gather algorithm, gathering a segment of size m_s on the root from $P-1$ processes, is estimated as follows [28],

$$T_{flat\_gather}(P, m) = \sum_{i=1}^{P-1} T^{c_i}_{p2p}(m_s) = \\ \sum_{i=1}^{P-1} \begin{cases} T^0_{p2p}(m_s), \text{ if } c_i = 0 \\ T^1_{p2p}(m_s), \text{ if } c_i = 1 \end{cases} = \\ \sum_{i=1}^{P-1} \begin{cases} o^0 + 2L^0, \text{ if } c_i = 0 \\ o^1 + 2L^0 + L^1, \text{ if } c_i = 1 \end{cases} \tag{17}$$

where o^0, o^1, L^0 and L^1 denote $o^0(m_s)$, $o^1(m_s)$, $L^0(m_s)$ and $L^1(m_s)$ respectively. Thus, as $\{c_i\}_{i=1}^{P-1}$ are all knowns for the experimental setup, $T_{flat\_gather}(P, m)$ is estimated as a linear function of unknown τ-Lop model parameters o^0, o^1, L^0 and L^1.

To explain in detail the contribution of the broadcast algorithm in the estimated time of the experiment, we assume the binomial tree broadcast algorithm. Therefore, according to formulas (3), (4) and (9), the execution time of the broadcast algorithm will be expressed as follows,

$$T_{bcast}(P, m) = T_{binomial}(P, m) = \\ \sum_{i=1}^{N} \max_{j_i} T_{NBFT}\left(P_{ij_i}, C_{ij_i}, m_s\right) = \\ \sum_{i=1}^{N} \max_{j_i} \begin{cases} \gamma^0(P_{ij_i}) \cdot T^0_{p2p}(m_s), \text{if } C_{ij_i} = 0 \\ \gamma^1(P^\circ_{ij}) \cdot T^1_{p2p}(m_s), \text{otherwise} \end{cases} = \\ \sum_{i=1}^{N} \max_{j_i} \begin{cases} \frac{\gamma^0(P_{ij_i})}{Q(m_s)} \cdot T^1_{p2p}(m_s), \text{if } C_{ij_i} = 0 \\ \gamma^1(P^\circ_{ij_i}) \cdot T^1_{p2p}(m_s), \text{otherwise} \end{cases} = \\ T^1_{p2p}(m_s) \cdot \sum_{i=1}^{N} \max_{j_i} \begin{cases} \frac{\gamma^0(P_{ij_i})}{Q(m_s)}, \text{if } C_{ij_i} = 0 \\ \gamma^1(P^\circ_{ij_i}), \text{otherwise} \end{cases} = \\ \left(o^1 + 2L^0 + L^1\right) \cdot \sum_{i=1}^{N} \max_{j_i} \begin{cases} \frac{\gamma^0(P_{ij_i})}{Q(m_s)}, \text{if } C_{ij_i} = 0 \\ \gamma^1(P^\circ_{ij_i}), \text{otherwise} \end{cases} \tag{18}$$

where N is the number of execution stages of the algorithm, $N = \lfloor log_2 P \rfloor + \frac{m}{m_s} - 1$; P_{ij_i} and C_{ij_i} are the number of nodes and the number of network point-to-point communications in the j_i-th NBFT at i-th stage respectively; $P^\circ_{ij_i} = C_{ij_i} + \lfloor \frac{P_{ij_i} - C_{ij_i} - 1}{Q(m_s)} \rfloor$.

In this experiment, $\{P_{ij_i}\}$ and $\{C_{ij_i}\}$ are all knowns. As presented in Section 5.1, $\gamma^c(P)$ and $Q(m_s)$ are algorithm-independent parameters, which are estimated separately, before the estimation of algorithm-specific τ-Lop parameters. Therefore, $P^\circ_{ij_i} = C_{ij_i} + \lfloor \frac{P_{ij_i} - C_{ij_i} - 1}{Q(m_s)} \rfloor$, $\frac{\gamma^0(P_{ij_i})}{Q(m_s)}$ and $\gamma^1(P^\circ_{ij_i})$ are also all knowns for all i and j_i. Thus, like $T_{flat\_gather}(P, m)$, $T_{bcast}(P, m)$ is also estimated as a linear function of unknown τ-Lop model parameters o^0, o^1, L^0 and L^1. Therefore, for each pair of $(P,\ m)$, formula (16) will yield one linear equation with unknown τ-Lop model parameters of the form

$$T_{comm\_experiment}(P, m) = \\ \lambda_1 \cdot o^0 + \lambda_2 \cdot o^1 + \lambda_3 \cdot L^0 + \lambda_4 \cdot L^1 \tag{19}$$

where λ_1, λ_2, λ_3 and λ_4 are constants. By repeating this experiment with different m, we obtain a system of linear equations for

Table 1
Estimated values of $P_{NBFT}^{max(1)}$, $P_{NBFT}^{max(0)}$ and $\gamma^1(P)$ on Grisou, Gros and MareNostrum4 clusters.

(a)

Cluster	$P_{NBFT}^{max(1)}$	$P_{NBFT}^{max(0)}$
Gros	7	3
Grisou	7	3
MareNostrum4	5	3

(b)

Number of processes (P)	$\gamma^1(P)$		
	Grisou	Gros	MareNostrum4
3	1.114	1.084	1.145
4	1.219	1.170	1.290
5	1.283	1.254	1.435
6	1.451	1.339	
7	1.540	1.424	

Table 2
Estimated values of o^1, L^0 and L^1 on the Grisou, MareNostrum4 and Gros clusters for Open MPI broadcast algorithms. o^0 is equal to 0 in all clusters.

Broadcast algorithm	$o^1, L^0, L^1 (sec)$		
	Grisou	MareNostrum4	Gros
Linear tree	2.1×10^{-4}, 2.1×10^{-4}, 2.1×10^{-4}	6.8×10^{-4}, 6.7×10^{-4}, 6.8×10^{-4}	8.2×10^{-5}, 8.2×10^{-5}, 8.2×10^{-5}
K-Chain tree	2.2×10^{-4}, 4.1×10^{-5}, 2.2×10^{-4}	6.2×10^{-5}, 1.2×10^{-5}, 6.2×10^{-5}	8.0×10^{-6}, 1.1×10^{-5}, 8.0×10^{-6}
Chain tree	1.2×10^{-5}, 2.0×10^{-5}, 1.2×10^{-5}	5.4×10^{-5}, 5.8×10^{-6}, 5.4×10^{-5}	8.1×10^{-6}, 1.3×10^{-5}, 8.1×10^{-6}
Split-binary tree	7.4×10^{-5}, 3.5×10^{-4}, 7.4×10^{-5}	1.7×10^{-4}, 2.1×10^{-5}, 1.7×10^{-4}	3.2×10^{-5}, 9.3×10^{-6}, 3.2×10^{-5}
Binary tree	3.3×10^{-4}, 3.0×10^{-4}, 3.3×10^{-4}	3.0×10^{-4}, 7.5×10^{-5}, 3.0×10^{-4}	5.6×10^{-5}, 9.7×10^{-6}, 5.6×10^{-5}
Binomial tree	9.7×10^{-5}, 2.1×10^{-4}, 9.7×10^{-5}	7.9×10^{-5}, 7.9×10^{-6}, 7.9×10^{-5}	1.9×10^{-5}, 8.0×10^{-6}, 1.9×10^{-5}

o^0, o^1, L^0 and L^1 (Fig. 8). Each equation in this system is represented in the canonical form, $\lambda_{i1} \cdot o^0 + \lambda_{i2} \cdot o^1 + \lambda_{i3} \cdot L^0 + \lambda_{i4} \cdot L^1 = T_i, (i = 1, ..., M)$. Finally, we use the least-square regression to find unknown model parameters. Similarly, we build systems of linear equations for other broadcast algorithms implemented in Open MPI.

6. Experimental results and analysis

In this section, we present experimental validation of the proposed approach to selection of optimal broadcast algorithms on multi-core clusters.

6.1. Experimental setup and methodology

We validate our approach on three large scale clusters. Two clusters, Grisou and Gros, are located in France and belong to the Grid5000 experimental infrastructure. The third cluster, MareNostrum4, is hosted by Barcelona Supercomputing Center.

Grid5000 is the large-scale testbed with seven sites in Grenoble, Luxembourg, Lyon, Nancy, Nantes, Rennes and Sophia. We run our experiments on Grisou and Gros clusters of the Nancy site using Open MPI 3.1. Grisou consists of 51 nodes each with 2 Intel Xeon E5-2630 v3 CPUs (8 cores/CPU), 128 GiB RAM, interconnected via 10 Gbps Ethernet. Gros consists of 124 nodes each with Intel Xeon Gold 5220 CPU (18 cores/CPU), 96 GiB RAM, interconnected via 25 Gbps Ethernet.

MareNostrum4 is a cluster based on Intel Xeon Platinum processors from the Skylake generation in the Barcelona Supercomputing Center. It consists of 3456 nodes each with 2-socket Intel Xeon Platinum 8160 CPU with 24 cores per socket, 96 GiB of main memory 1.880 GB/core, interconnected via 10 Gbit Ethernet.

In our collective experiments, we use up to 38 nodes in Grisou, up to 56 nodes in Gros, and up to 10 nodes in MareNostrum4. MPI programs use the one-process-per-CPU-core configuration, and the maximal total number of processes is 600 for Grisou, 1000 for Gros and 480 for MareNostrum4. They utilize all CPU-cores in the nodes used in experiments. The default serial mapping of MPI processes to cores is used in all programs. The message segment size, m_s, for segmented broadcast algorithms is set to 8 KB and is the same in all experiments. This very segment size is commonly used for segmented broadcast algorithms in Open MPI. Selection of optimal segment size is out of the scope of this paper.

We follow a detailed methodology to make sure that the experimental results are reliable: 1) We make sure that the cluster is fully reserved and dedicated to our experiments. 2) For each data point in the execution time of collective algorithms, the sample mean is used, which is calculated by executing the application repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. We also check that the individual observations are independent and their population follows the normal distribution. For this purpose, MPIBlib tool [27] is used. As presented in Section 5.2, we design the communication experiments using broadcast algorithms and the flat-without-synchronization gather algorithm. Gathering the message in the root enables us to measure the execution time of the experiment using the root clock where it is started. The estimation of the execution time is synchronized by the three MPI_Barrier. The first MPI_Barrier is used before the loop where repetitions start. The second is used directly before MPI_Bcast to make sure that processes start the message transmission at the same time. The last one is used directly after MPI_Bcast to make sure that all processes received the message, then gather starts.

All results presented in the paper are reproducible. The MPI code to run communication experiments and Python scripts to train models is freely available from the UCD GitLab server [2].

$$\begin{cases} 14.31 \cdot o^1 + 1012.31 \cdot L^0 + 14.31 \cdot L^1 = 0.0103169514 \\ 17.15 \cdot o^1 + 1015.15 \cdot L^0 + 17.15 \cdot L^1 = 0.010932707 \\ 22.85 \cdot o^1 + 1020.85 \cdot L^0 + 22.85 \cdot L^1 = 0.0121768391 \\ 34.24 \cdot o^1 + 1032.24 \cdot L^0 + 34.24 \cdot L^1 = 0.015099681 \\ 57.03 \cdot o^1 + 1055.03 \cdot L^0 + 57.03 \cdot L^1 = 0.0199503694 \\ 102.59 \cdot o^1 + 1100.59 \cdot L^0 + 102.59 \cdot L^1 = 0.0293413861 \\ 193.73 \cdot o^1 + 1191.73 \cdot L^0 + 193.73 \cdot L^1 = 0.0479402791 \\ 376.00 \cdot o^1 + 1374.00 \cdot L^0 + 376.00 \cdot L^1 = 0.0479402791 \\ 740.55 \cdot o^1 + 1738.55 \cdot L^0 + 740.55 \cdot L^1 = 0.1595142696 \end{cases} \tag{20}$$

A system of linear equations built in Gros cluster using binomial tree broadcast algorithm where $P = 1000$ and $m \in [16KB, 4MB]$.

Table 3
Comparison of the model-based and Open MPI selections with the best performing MPI_Bcast algorithm. For each selected algorithm, its performance degradation against the optimal one is given in braces.

P=450, Grisou

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*flat*	*chain* (24)	*split_binary* (1568)
32	*chain*	*chain* (0)	*split_binary* (1211)
64	*chain*	*chain* (0)	*split_binary* (702)
128	*chain*	*chain* (0)	*split_binary* (1153)
256	*chain*	*chain* (0)	*split_binary* (1106)
512	*chain*	*chain* (0)	*chain* (0)
1024	*chain*	*chain* (0)	*chain* (0)
2048	*chain*	*chain* (0)	*chain* (0)
4096	*chain*	*chain* (0)	*chain* (0)

P=600, Grisou

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*flat*	*chain* (20)	*split_binary* (1006)
32	*chain*	*chain* (0)	*split_binary* (744)
64	*chain*	*chain* (0)	*split_binary* (808)
128	*chain*	*chain* (0)	*split_binary* (792)
256	*chain*	*chain* (0)	*split_binary* (762)
512	*chain*	*chain* (0)	*chain* (0)
1024	*chain*	*chain* (0)	*chain* (0)
2048	*chain*	*chain* (0)	*chain* (0)
4096	*chain*	*chain* (0)	*chain* (0)

P=400, Gros

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*split_binary*	*binary* (1)	*split_binary* (0)
32	*split_binary*	*binary* (6)	*split_binary* (0)
64	*split_binary*	*split_binary* (0)	*split_binary* (0)
128	*split_binary*	*split_binary* (0)	*split_binary* (0)
256	*k − chain*	*k − chain* (0)	*split_binary* (4)
512	*k − chain*	*k − chain* (0)	*chain* (67)
1024	*k − chain*	*k − chain* (0)	*chain* (76)
2048	*k − chain*	*k − chain* (0)	*chain* (62)
4096	*k − chain*	*k − chain* (0)	*chain* (42)

P=1000, Gros

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*split_binary*	*binomial* (1)	*split_binary* (0)
32	*split_binary*	*split_binary* (0)	*split_binary* (0)
64	*split_binary*	*split_binary* (0)	*split_binary* (0)
128	*split_binary*	*split_binary* (0)	*split_binary* (0)
256	*split_binary*	*split_binary* (0)	*split_binary* (0)
512	*split_binary*	*split_binary* (0)	*chain* (97)
1024	*split_binary*	*k − chain* (2)	*chain* (570)
2048	*k − chain*	*k − chain* (0)	*chain* (73)
4096	*k − chain*	*k − chain* (0)	*chain* (66)

P=96, MareNostrum4

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*split_binary*	*k − chain* (3)	*split_binary* (0)
32	*k − chain*	*k − chain* (0)	*split_binary* (4)
64	*k − chain*	*k − chain* (0)	*split_binary* (19)
128	*k − chain*	*k − chain* (0)	*split_binary* (40)
256	*k − chain*	*k − chain* (0)	*split_binary* (74)
512	*k − chain*	*k − chain* (0)	*chain* (9)
1024	*chain*	*k − chain* (1)	*chain* (0)
2048	*chain*	*chain* (0)	*chain* (0)
4096	*chain*	*chain* (0)	*chain* (0)

P=480, MareNostrum4

m (KB)	Best	Model-based (%)	Open MPI (%)
16	*split_binary*	*k − chain* (84)	*split_binary* (0)
32	*k − chain*	*k − chain* (0)	*split_binary* (92)
64	*k − chain*	*k − chain* (0)	*split_binary* (385)
128	*k − chain*	*k − chain* (0)	*split_binary* (719)
256	*k − chain*	*k − chain* (0)	*split_binary* (209)
512	*k − chain*	*k − chain* (0)	*chain* (31)
1024	*k − chain*	*k − chain* (0)	*chain* (21)
2048	*k − chain*	*k − chain* (0)	*chain* (49)
4096	*k − chain*	*k − chain* (0)	*chain* (49)

6.2. Experimental estimation of model parameters

Platform-specific but algorithm-independent model parameters $\gamma^c(P)$ and $Q(m_s)$ are estimated first (and separately for each experimental cluster) following the methodology described in Section 5.1.

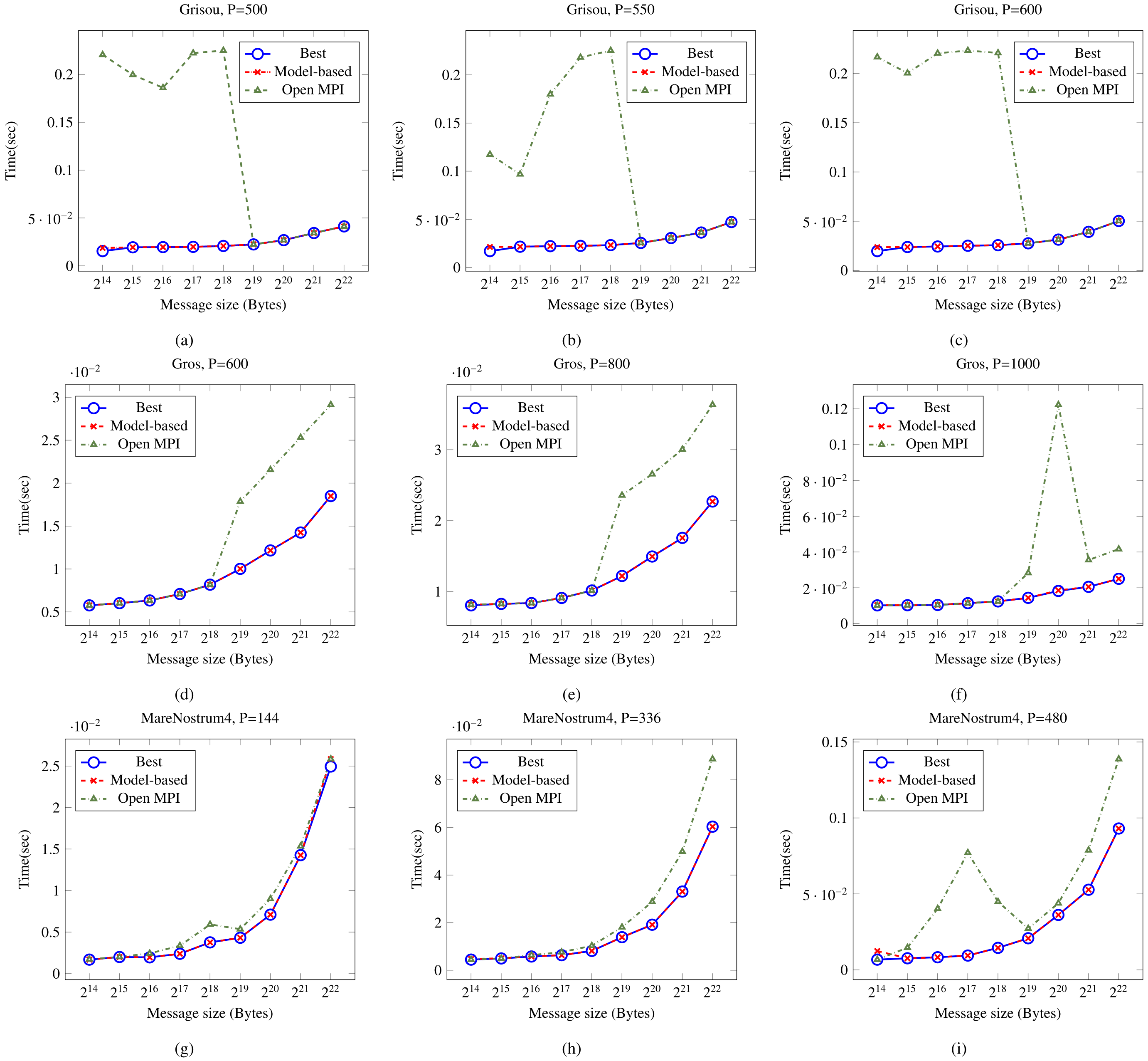
The calculated values of $P^{max(1)}_{NBFT}$ and $P^{max(0)}_{NBFT}$ for our experimental setups are given in Table 1(a). $\gamma^0(P^{max(0)}_{NBFT})$ is estimated 1 for all the clusters. $Q(m_s)$ is estimated 6 for Grisou, 12 for Gros and 9 for MareNostrum4. The estimated values of $\gamma^1(P)$ for P from 3 to 7 are given in Table 1.

Then, algorithm-specific τ-Lop model parameters are estimated for each platform and each algorithm, following the method described in Section 5.2. In the communication experiments, we use 600 processes on Grisou, 1000 processes on Gros, and 480 processes on MareNostrum4. The message size, m, varies in the range from 16 KB to 4 MB on all platforms. We use 9 different message sizes for Open MPI broadcast algorithms, $\{m_i\}_{i=1}^{9}$, separated by a constant step in the logarithmic scale, $\log_2 m_{i+1} - \log_2 m_i = 1$. Thus, for each broadcast algorithm, we obtain a system of 9 linear equations with τ-Lop model parameters as unknowns (See (20)). We use the Huber regressor [18] to find their values from the system. The values of the parameters for each platform can be found in Table 2. We can see that the values of model parameters do vary depending on the broadcast algorithm. The results support our original hypothesis that the average execution time of a point-to-point communication will very much depend on the context of the use of point-to-point communications in the algorithm. Therefore, the estimated values capture more than just sheer network characteristics. Despite the fact that the Split-binary tree and Binary tree broadcast algorithms use the same virtual topology, the estimated time of a point-to-point communication is smaller in the context of the Split-binary one. This can be explained by a higher level of parallelism of the Split-binary algorithm, where a significant part of point-to-point communications is performed in parallel by a large number of independent pairs of processes from the left and right subtrees. o^0 has been estimated 0 in all clusters because of the small size, 8 K, of the message segment transmitted through a shared memory channel.

6.3. Accuracy of selection of optimal collective algorithms using the constructed analytical performance models

The constructed analytical performance models of the Open MPI broadcast algorithms are designed for the use in the MPI_Bcast routines for efficient and accurate runtime selection of the optimal algorithm, depending on the number of processes and the message size. While the efficiency is evident from the low complexity of the analytical formulas derived in Section 4, the experimental results on the accuracy are presented in this section.

Fig. 9 shows the results of our experiments in three clusters for MPI_Bcast. We present results of experiments with $P \in \{450, 500, 550, 600\}$ in Grisou, with $P \in \{400, 600, 800, 1000\}$ in Gros and with $P \in \{96, 144, 336, 480\}$ in MareNostrum4. Again, the message size, m, varies in the range from 16 KB to 4 MB on all platforms, and we use 9 different sizes, $\{m_i\}_{i=1}^{9}$, separated by a constant step in the logarithmic scale, $\log_2 m_{i+1} - \log_2 m_i = 1$. The plots show the execution time of the broadcast operation as a



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

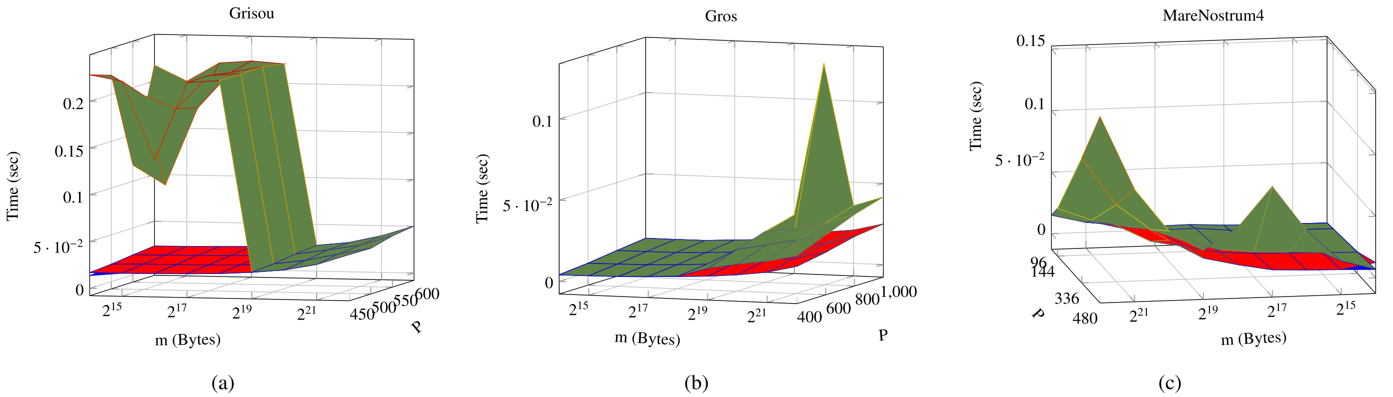
Fig. 9. Comparison of the selection accuracy of the Open MPI decision function and the proposed model-based method for MPI_Bcast. (9a - 9c), (9d - 9f) and (9g - 9i) present performance of collectives on Grisou, Gros and MareNostrum4 clusters respectively.

function of the message size. Each data point on a green line shows the performance of the algorithm selected by the Open MPI decision function for the given number of processes and message size. Each point on a red line shows the performance of the algorithm selected by our decision function, which uses the constructed analytical models. Each point on a blue line shows the performance of the best broadcast algorithm for MPI_Bcast. Fig. 10 demonstrates the accuracy of the model-based selection compared to the best performance and Open MPI decision function for all message sizes and the number of processes on three clusters.

Table 3 presents selections made for MPI_Bcast using the proposed model-based runtime procedure and the Open MPI decision function. For each message size m, the best performing algorithm, the model-based selected algorithm, and the Open MPI selected algorithm are given. For the latter two, the performance degradation in percentages in comparison with the best performing algorithm is also given.

It is evident from the results that for all message sizes but 16 KB, the model-based decision function selects either the optimal (45 cases) or near-optimal (3) algorithm. The performance of the near optimal algorithms is practically indistinguishable from that of the optimal (the difference is 1%, 2% and 6% correspondingly).

In the case of 16 KB, on the Gros cluster it selects the near optimal algorithms, which are practically as good as the optimal (the difference in performance is only 1%). On the Grisou cluster, the second best algorithm is selected with a moderate degradation of $20-24\%$. The second best algorithm is also selected on the MareNostrum4 cluster with a degradation of $3-84\%$. Thus, the inaccuracy of the model-based decision function for 16 KB experiments is either minor or quite moderate. It can be explained by the fact that we use linear regression to find the values of model



(a) (b) (c)

Fig. 10. (10a), (10b) and (10c) present performance of MPI_Bcast on Grisou, Gros and MareNostrum4 clusters respectively. Blue, red and green surfaces present the best performance of MPI_Bcast, model-based estimation and Open MPI decision function respectively.

parameters from the experimentally constructed systems of linear equations. Linear regression methods tend to prioritize larger values in experimental data points over smaller ones when minimizing the penalty of the fit. The Huber regressor [18], which we use in our work, tries to mitigate this problem by increasing the weight of smaller values but still follows the general trend. One possible solution is to break the whole range of message sizes into two segments, one for smaller messages and the other for larger ones, and find model parameters separately for each segment.

The Open MPI selection is near optimal in 46% cases and causes significant performance degradation in the remaining 54% cases (up to 1106%, 570% and 719% on Grisou, Gros and MareNostrum4 clusters respectively).

The Open MPI decision function only uses three broadcast algorithms (*chain tree, split-binary tree and binomial tree*) out of a total of six implemented and available for selection (see Listing 1). For example, the K-Chain tree broadcast algorithm, which is never used by the Open MPI decision function, outperforms all the algorithms on Gros in 38% cases, and on MareNostrum4 in 72% cases. In contrast to the Open MPI decision function, the model-based function selects the K-Chain broadcast algorithm on Gros and MareNostrum4 platforms, when this algorithm is either the best or the second best with a very small penalty of $1-2\%$.

7. Discussion

In this section, we briefly discuss some limitations of the presented work and their impact.

First, our approach assumes that collectives are implemented through calls to point-to-point communication operations. We do not consider MPI implementations that exploit hardware collective support to perform certain collectives, for example, multicast, in O(1).

Second, we assume that the segment size, m_s, is fixed and the same in all collective algorithms. This limitation can be eased by making the segment size another decision variable with values from a small discrete set, say, $\{8K, 16K, 32K, 64K, 128K\}$. For each collective algorithm, we can build a separate model for each value of the segment size and use the models at runtime to select the fastest combination of the algorithm and segment size for each collective operation.

Third, we assume that the values of model parameters, such as o^0, o^1, L^0 and L^1 do not depend on the number of processes, P, executing the algorithm. While this assumption did not negatively affect the selective accuracy of the models in our experimental setups, it may not be the case for larger platforms, able to run tens of thousands of MPI processes. For such platforms, one possible solution could be to break the total range of the number of processes into several segments and find the values of model parameters separately for each segment. There are other, more general possible solutions, but in order to study any possible solution, regular access to a large-scale platform is needed. Unfortunately, the authors do not have such access.

8. Conclusion

In this paper, we proposed a model-based approach to automatic selection of optimal algorithms for the MPI broadcast operation on multi-core clusters, which proved to be both efficient and accurate. We took into account the topology of the communication channels to build performance models. Communication experiments are designed to estimate algorithmic and channel-specific model parameters.

We also developed this approach into a detailed method and applied it to Open MPI 3.1 and its MPI_Bcast operations. We experimentally validated this method on three different clusters and demonstrated its accuracy and efficiency. These results suggest that the proposed approach, based on analytical performance modeling of collective algorithms, provides the solution of the problem of accurate and efficient runtime selection of optimal algorithms for MPI collective operations.

The target architecture for the presented method is a cluster of multi-core processors. While this is the most common architecture where MPI is used, there are other architectures such as heterogeneous clusters [23], [42] or Internet of Things [22], [15], where MPI either is used or can be used in the future. Such platforms are highly heterogeneous, represent a significant challenge for accurate modeling of MPI collective algorithms and are out of the scope of this paper. We consider this topic as future work.

CRediT authorship contribution statement

Emin Nuriyev: Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Juan-Antonio Rico-Gallego:** Formal analysis, Investigation, Writing – original draft. **Alexey Lastovetsky:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A Message-Passing Interface Standard, https://www.mpi-forum.org/.

[2] A tool to run MPI communication experiments and build performance models presented in this paper, https://csgitlab.ucd.ie/emin.nuri/mpicollmodelling.

[3] A. Alexandrov, M.F. Ionescu, K.E. Schauser, C. Scheiman LogGP, Incorporating long messages into the LogP model — one step closer towards a realistic model for parallel computation, Tech. Rep., USA, 1995.

[4] M. Bernaschi, G. Iannello, M. Lauria, Efficient implementation of reduce-scatter in MPI, J. Syst. Archit. 49 (3) (2003) 89–108, https://doi.org/10.1016/S1383-7621(03)00059-6, parallel, Distributed and Network-based Processing - selected papers from the 10th Euromicro Workshop, http://www.sciencedirect.com/science/article/pii/S1383762103000596.

[5] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, D. Weathersby, Efficient algorithms for all-to-all communications in multiport message-passing systems, IEEE Trans. Parallel Distrib. Syst. 8 (11) (1997) 1143–1156, https://doi.org/10.1109/71.642949.

[6] K.W. Cameron, R. Ge, Predicting and evaluating distributed communication performance, in: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, IEEE, 2004, p. 43.

[7] E. Chan, M. Heimlich, A. Purkayastha, R. van de Geijn, Collective communication: theory, practice, and experience: research articles, Concurr. Comput., Pract. Exp. 19 (13) (2007) 1749–1783.

[8] E.W. Chan, M.F. Heimlich, A. Purkayastha, R.A. van de Geijn, On optimizing collective communication, in: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935), 2004, pp. 145–155.

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: towards a realistic model of parallel computation, SIGPLAN Not. 28 (7) (1993) 1–12, https://doi.org/10.1145/173284.155333.

[10] D. Culler, L.T. Liu, R.P. Martin, C. Yoshikawa, LogP performance assessment of fast network interfaces, IEEE MICRO 16 (1) (1996) 35–43.

[11] T.G. Dietterich, E.B. Kong, Machine learning bias, statistical bias, and statistical variance of decision tree algorithms, Tech. Rep., Technical report, Department of Computer Science, Oregon State University, 1995.

[12] T.v. Eicken, D.E. Culler, S.C. Goldstein, K.E. Schauser, Active messages: a mechanism for integrated communication and computation, in: [1992] Proceedings the 19th Annual International Symposium on Computer Architecture, 1992, pp. 256–266.

[13] G.E. Fagg, J. Pjesivac-Grbovic, G. Bosilca, T. Angskun, J. Dongarra, E. Jeannot, Flexible collective communication tuning architecture applied to Open MPI, in: Euro PVM/MPI, 2006.

[14] 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: D. Kranzlmüller, P. Kacsuk, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 97–104.

[15] T. Guo, K. Yu, M. Aloqaily, S. Wan, Constructing a prior-dependent graph for data clustering and dimension reduction in the edge of AIoT, Future Gener. Comput. Syst. 128 (2022) 381–394.

[16] R.W. Hockney, The communication challenge for MPP: Intel Paragon and Meiko CS-2, Parallel Comput. 20 (3) (1994) 389–398, https://doi.org/10.1016/S0167-8191(06)80021-9, http://www.sciencedirect.com/science/article/pii/S0167819106800219.

[17] T. Hoefler, T. Schneider, A. Lumsdaine, LogGP in theory and practice – an in-depth analysis of modern interconnection networks and benchmarking methods for collective operations, Simul. Model. Pract. Theory 17 (9) (2009) 1511–1521, https://doi.org/10.1016/j.simpat.2009.06.007, Advances in System Performance Modelling, Analysis and Enhancement, http://www.sciencedirect.com/science/article/pii/S1569190X09000811.

[18] P.J. Huber, Robust estimation of a location parameter, in: Breakthroughs in Statistics, Springer, 1992, pp. 492–518.

[19] S. Hunold, A. Bhatele, G. Bosilca, P. Knees, Predicting MPI collective communication performance using machine learning, in: 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020, pp. 259–269.

[20] L.P. Huse, Collective communication on dedicated clusters of workstations, in: European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting, Springer, 1999, pp. 469–476.

[21] T. Kielmann, H.E. Bal, K. Verstoep, Fast measurement of LogP parameters for message passing platforms, in: J. Rolim (Ed.), Parallel and Distributed Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 1176–1183.

[22] E. Kristiani, C.-T. Yang, C.-Y. Huang, P.-C. Ko, H. Fathoni, On construction of sensors, edge, and cloud (iSEC) framework for smart system integration and applications, IEEE Int. Things J. 8 (1) (2020) 309–319.

[23] A. Lastovetsky, R. Reddy Heterompi, Towards a message-passing library for heterogeneous networks of computers, J. Parallel Distrib. Comput. 66 (2) (2006) 197–220.

[24] A. Lastovetsky, V. Rychkov, Building the communication performance model of heterogeneous clusters based on a switched network, in: 2007 IEEE International Conference on Cluster Computing, 2007, pp. 568–575.

[25] A. Lastovetsky, V. Rychkov, Accurate and efficient estimation of parameters of heterogeneous communication performance models, Int. J. High Perform. Comput. Appl. 23 (2) (2009) 123–139, https://doi.org/10.1177/1094342009103947.

[26] A. Lastovetsky, I. Mkwawa, M. O'Flynn, An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network, in: 12th International Conference on Parallel and Distributed Systems - (ICPADS'06), vol. 2, 2006.

[27] A. Lastovetsky, V. Rychkov, M. O'Flynn MPIBlib, Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters, in: A. Lastovetsky, T. Kechadi, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 227–238.

[28] E. Nuriyev, A. Lastovetsky, Efficient and accurate selection of optimal collective communication algorithms using analytical performance modeling, IEEE Access 9 (2021) 109355–109373, https://doi.org/10.1109/ACCESS.2021.3101689.

[29] P. Patarasuk, A. Faraj, Xin Yuan, Pipelined broadcast on Ethernet switched clusters, in: Proceedings 20th IEEE International Parallel Distributed Processing Symposium, 2006.

[30] J. Pješivac-Grbović, G.E. Fagg, T. Angskun, G. Bosilca, J.J. Dongarra, MPI collective algorithm selection and quadtree encoding, in: B. Mohr, J.L. Träff, J. Worringen, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 40–48.

[31] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G.E. Fagg, E. Gabriel, J.J. Dongarra, Performance analysis of MPI collective operations, Clust. Comput. 10 (2) (2007) 127–143, https://doi.org/10.1007/s10586-007-0012-0.

[32] J. Pješivac-Grbović, G. Bosilca, G.E. Fagg, T. Angskun, J.J. Dongarra, Decision trees and MPI collective algorithm selection problem, in: A.-M. Kermarrec, L. Bougé, T. Priol (Eds.), Euro-Par 2007 Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 107–117.

[33] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[34] R. Rabenseifner, Automatic profiling of MPI applications with hardware performance counters, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, in: Lecture Notes in Computer Science, vol. 1697, Springer Berlin Heidelberg, 1999, pp. 35–42.

[35] R. Rabenseifner, J.L. Träff, More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems, in: D. Kranzlmüller, P. Kacsuk, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 36–46.

[36] J. Rico-Gallego, A.L. Lastovetsky, J.C.D. Martín, Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 3215–3228, https://doi.org/10.1109/TPDS.2017.2715809.

[37] J.-A. Rico-Gallego, J.-C. Díaz-Martín, τ-Lop: modeling performance of shared memory MPI, Parallel Comput. 46 (2015) 14–31, https://doi.org/10.1016/j.parco.2015.02.006, http://www.sciencedirect.com/science/article/pii/S0167819115000447.

[38] J.-A. Rico-Gallego, J.-C. Díaz-Martín, A.L. Lastovetsky, Extending τ-lop to model concurrent MPI communications in multicore clusters, Future Gener. Comput. Syst. 61 (2016) 66–82, https://doi.org/10.1016/j.future.2016.02.021, http://www.sciencedirect.com/science/article/pii/S0167739X16300346.

[39] J.A. Rico-Gallego, J.C. Díaz-Martín, R.R. Manumachu, A.L. Lastovetsky, A survey of communication performance models for high-performance computing, ACM Comput. Surv. 51 (6) (Jan. 2019), https://doi.org/10.1145/3284358.

[40] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of collective communication operations in MPICH, Int. J. High Perform. Comput. Appl. 19 (1) (2005) 49–66, https://doi.org/10.1177/1094342005051521.

[41] B. Tu, J. Fan, J. Zhan, X. Zhao, Performance analysis and optimization of MPI collective operations on multi-core clusters, J. Supercomput. 60 (1) (2012) 141–162.

[42] D. Valencia, A. Lastovetsky, M. O'Flynn, A. Plaza, J. Plaza, Parallel processing of remotely sensed hyperspectral images on heterogeneous networks of workstations using HeteroMPI, Int. J. High Perform. Comput. Appl. 22 (4) (2008) 386–407.

[43] U. Wickramasinghe, A. Lumsdaine, A survey of methods for collective communication optimization and tuning, arXiv preprint, arXiv:1611.06334, 2016.

[44] J. Worringen, Pipelining and overlapping for MPI collective operations, in: 28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN '03. Proceedings, 2003, pp. 548–557.

Emin Nuriyev is a PhD researcher at Heterogeneous Computing Laboratory at the School of Computer Science, University College Dublin. He received a BSc and MSc degrees in Applied Mathematics from the Baku State University in 2005 and 2007 respectively. His main research interests include algorithms and models for High-Performance Computing.

Juan-Antonio Rico-Gallego received the Computer Science Engineering degree and the PhD degree on Computer Science from the University of Extremadura in 2002 and 2016 respectively. Formerly a software consultant, he is an associate professor at the Dept. of Computer Systems Engineering of the University of Extremadura (Spain). His research interests are in Performance Modeling and Analysis, MPI standard implementations and applications and optimization of application on heterogeneous high performance computing platforms with learning-based methods.

Alexey L. Lastovetsky received a Ph.D. degree from the Moscow Aviation Institute in 1986, and a Doctor of Science degree from the Russian Academy of Sciences in 1997. His main research interests include algorithms, models, and programming tools for high performance heterogeneous computing. He is currently Associate Professor in the School of Computer Science at University College Dublin (UCD). At UCD, he is also the founding Director of the Heterogeneous Computing Laboratory.