Tesis Doctoral

τ–Lop: Scalably and Accurately Modeling Contention and Mapping Effects in Multi-core Clusters

Juan Antonio Rico Gallego

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

2015

Tesis Doctoral

τ–Lop: Scalably and Accurately Modeling Contention and Mapping Effects in Multi-core Clusters

Juan Antonio Rico Gallego

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

2015

El director, Juan Carlos Díaz Martín, conforme:

UNIVERSIDAD DE EXTREMADURA

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

Abstract

τ–Lop: Scalably and Accurately Modeling Contention and Mapping Effects in Multi-core Clusters

by Juan Antonio Rico Gallego

Modern HPC multi-core platforms are complex systems composed of heterogeneous processors and a hierarchy of shared communication channels. Achieving optimal performance of MPI applications on that platforms is not trivial. Formal analysis using *parallel performance models* contributes to depict algorithms behavior and communication complexities, with the goal of predicting their cost and improving their performance.

Current accepted communication models, as the representative *LogGP*, were initially conceived to predict the cost of algorithms in mono-processor clusters as a sequence of point-to-point transmissions characterized by network latency and bandwidth parameters. Although multiple extensions have been proposed for covering issues derived from current platforms complexities, as contention and channels hierarchy, such specific extensions are not enough to meaningfully and accurately model more than simple algorithms. As modern supercomputers are built upon cheap commodity boards with a growing number of cores, intra-node communication becomes progressively more relevant, as well as the derived contention in the communication channels. These heterogeneous high performance computing platforms need new approaches for the communication performance modeling to address their complexities.

This work unveils the reasons for the poor fit of the cited representative models in this domain, and proposes a new model named τ–Lop, which addresses the challenge of accurately modeling MPI communications on heterogeneous multi-core clusters. τ–Lop is based on the concept of *concurrent transfers*, and applies it to meaningfully represent the behavior of algorithms in platforms with hierarchical shared communication channels, taking into account the effects of contention and deployment of processes on the processors. It demonstrates the ability to predict the cost of advanced algorithms and communication mechanisms used by mainstream MPI implementations, such as MPICH or Open MPI, with a high accuracy. In addition, an exhaustive and reproducible methodology for measuring the parameters of the model is described.

Acknowledgements

The author would like to thanks the following people and institutions which, in different ways, have contributed to the development of this work:

The Extremadura Supercomputing Center (CenitS), specially to Dr. José Luis González Sánchez and César Gómez Martín, for their help and providing with some of the computing resources used in this work.

The High Performance computing Center in Stuttgart (HLRS), specially to Dr. Rolf Rabenseifner, for his hospitality and orientation in my visits to the center, providing computing resources, and to the HPC-Europe2 program of the European Commission which funded the visits.

The Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF), for providing with computing resources, in special to Dr. Abel Paz Gallardo for his kindly help.

The EU eCOST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS), in special to Prof. Alexey L. Lastovetsky from the University College Dublin (UCD), for its hospitality and decisive contributions to the development of this work.

To Dr. Javier García Blas from the University Carlos III in Spain, for their help in several aspect during last years.

Finally, my special thanks to Dr. Juan Carlos Díaz Martín, my PhD supervisor and my friend. Without his help, orientation and hard work, this thesis would not have been written.

Contents

List of Figures

List of Tables

Abbreviations

CPU	**C**entral **P**rocessing **U**nit
CTS	**C**lear **T**o **S**end
FPM	**F**unctional **P**erformance **M**odel
GPU	**G**raphic **P**rocessing **U**nit
HPC	**H**igh **P**erformance **C**omputing
KB	**K**ilo**B**ytes
MB	**M**ega**B**ytes
MPI	**M**essage **P**assing **I**nterface
NUMA	**N**on-**U**niform **M**emory **A**ccess
RDA	**R**ecursive **D**oubling **A**llgather
RR	**R**ound **R**obin
RTS	**R**equest **T**o **S**end
RTT	**R**ound **T**rip **T**ime
SEQ	**SEQ**uential
SHM	**SH**ared **M**emory
SMP	**S**ymmetric **M**ulti-**P**rocessing
SUMMA	**S**calable **U**niversal **M**atrix **M**ultiplication **A**lgorithm
UMA	**U**niform **M**emory **A**ccess

Symbols

Next, a list of symbols commonly used in the document is included. Usually, uppercase symbols are used to represent constant values, and lowercase symbols are used to represent variables. Original symbols used in related works are kept when possible.

c: number of communication channel departing from 0.
M: number of nodes in a cluster.
m: size of a message.
N: number of communication channels in the system.
n: number of hops a message or segment needs to reach the destination.
P: number of processes involved in a collective operation.
Q: number of processes in a node of the multi-core cluster.
S: size of a message segment in bytes.
s: number of stages in a collective operation.
T: cost of a point-to-point transmission of a message.
Θ: cost of a collective operation.
τ: number of concurrent transfers accessing a communication channel.

Dedicated to Isa and my parents, everything in my life I owe to them.

Chapter 1

Introduction

Twenty years ago parallel machines were converging to an emerging architecture, clusters of workstations, now known as multicomputers and, more recently, as multi-core clusters. Cost models such as *Hockney* [1] and *LogP* ([2, 3]) addressed the performance of the network with the goal of guiding the design of a parallel algorithm, i.e., organizing the computing loads and the communicating events (point-to-point messages) to optimize its execution time in the cluster. LogP, for instance, characterizes the network performance based on the parameters *network delay*, *overhead* or time invested by the processor in sending or receiving a message, and *gap per message* or minimum time interval between message transmissions. The execution time of a parallel algorithm is now formulated in terms of these parameters with the goal of analyzing the performance of the algorithm in the cluster.

LogGP [4] is an extension of LogP with an additional parameter, *gap per byte*, which captures the network bandwidth for long messages. LogGP is a representative message-passing performance model. It assumes that collective algorithms are built upon point-to-point message transmissions deployed over a network of mono-processors and models an algorithm as a sequence of stages, each one with the cost of a single point-to-point transmission. For instance, a Broadcast operation on P processes based on the *Binomial Tree* algorithm has a cost that equals the height of the tree ($\lceil \log P \rceil$) times the cost of a single transmission between two processes (see Figure 1.1). In modern HPC platforms this is an over-simplification that leads to important errors in the cost estimation of an algorithm, as it will be discussed in this thesis.

Modern high performance computing systems are composed of nodes with a significant number of cores per node and deep memory hierarchies, connected by high performance networks. Scientific applications face the challenge of obtaining as much performance as possible from such complex platforms. Hence, another branch of interest in LogP/LogGP

is its ability to model the underlying message-passing algorithms that implement the primitives of the MPI standard [5]. MPI is, and will continue to be, the de facto standard that defines the communication interface used by this type of applications, despite of others programming models have appeared to supplement its scalability problems ([6, 7]). MPI modeling is a critical issue, because it allows a formal machine-independent performance comparison and analysis of different implementations of the standard [8–11].

The MPI execution model is based on processes communicating by message passing, including point-to-point and collective operations, which involve a group of processes. Frequently used, collective operations are a key issue in achieving performance and scalability in parallel applications, and their costs have an important influence in the global performance of MPI applications. A well known profiling study by Rabenseifner [12] shows that MPI applications spend more than eighty per cent of communication time in collective operations; hence, efficient algorithms for the *Broadcast* or *Reduce* have attracted special interest [13].

Communication performance models contribute to understand the collective algorithms behavior and aid in the optimization of applications. Performance of the underlying algorithms implementing MPI collectives vary significantly with parameters as the message size, the number of involved processes and the platform characteristics. Hence, a collective operation can be implemented by algorithms from a preset menu, with one of them chosen at runtime based on the value of such parameters. In general, an algorithm is executed as a sequence of steps, and defines a point-to-point communication graph between the processes of the group. Algorithms are designed with the goals of minimizing the number of such steps, optimizing the use of the underlying communication channels available in the system, or fitting a particular network technology or topology. Besides, achieving these goals in modern complex multi-core clusters requires to consider the hierarchy of communication channels (imposed by the difference in channels performance), the implications of process distribution over the cores of the platform (virtual topology), and the effects of the contention due to the use of shared communication resources.

Modeling and formal analysis of such algorithms is important to gain insights into their performance. Due to the complexity and the lack of accuracy of the communication modeling in modern heterogeneous HPC platforms, performance models have been partially left behind in favor of intensive experimental measurements, more expensive in computational time, system dependent, and which not ensure higher accuracy [14].

To get a feel of collective cost modeling, the well known Binomial tree broadcasting algorithm is used here. A process named *root* sends data to the rest of the processes in the *communicator*, an isolated group of P processes identified by their *rank* numbers,

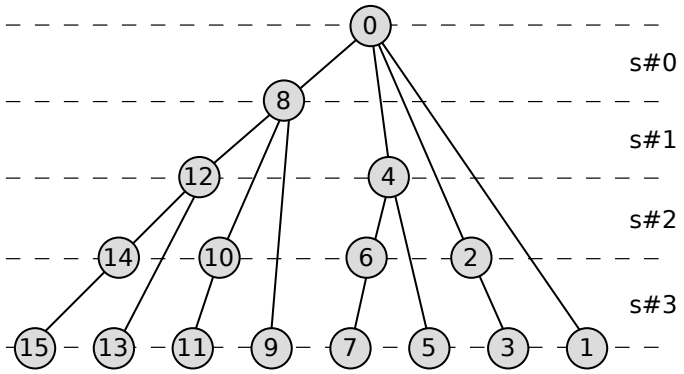


FIGURE 1.1: The broadcasting algorithm as a Binomial tree for $P = 16$ processes and $\lceil \log_2(P) \rceil = 4$ stages. An arc is a message transmission between two processes. Processes are identified by rank, 0 being rank of the root.

defined in MPI as a consecutive integer number in the range $0 \ldots P-1$. Figure 1.1 shows the algorithm deployment. The root process sends a message to the process with rank $root + P/2$. The algorithm continues recursively, with these processes as *roots* of their own trees with $P/2$ processes. It completes in $\lceil \log_2 P \rceil$ stages, the height of the tree; so, the cost of the algorithm is $T = \lceil \log_2 P \rceil \times (\alpha + m\beta)$ under the Hockney model, where m is the size of the message, and α is the latency and β is the inverse of the bandwidth for the specific network.

Current MPI implementations adopt common well established point-to-point algorithms in the inter-node[1] communication, as the Binomial tree formerly examined. However, each implementation develops its own techniques in the intra-node domain. Two popular MPI implementations, *MPICH* [15] and *Open MPI* [16], are mainly considered in this work. MPICH builds intra-node collectives upon point-to-point message communications. Open MPI promotes a software architecture based on components providing different communication mechanisms, as the *SM* (shared memory) collective component, which communicates processes in shared memory through a common memory zone mapped to processes involved. As a consequence, point-to-point Hockney and LogP/LogGP are not the indicated modeling tools for Open MPI on shared memory, as it will be demonstrated in this thesis. Besides, their difficulties in modeling shared memory performance seem to have been tacitly recognized by the HPC community. In fact, to the best of the author's knowledge, no report exists on the practical ability of the LogP family of algorithms to model the performance of MPI in shared memory. Yet, representative model LogGP has been incorporated in the discussions from a formal point of view and its parameters were measured in the studied platforms.

[1] The term *intra-node* refers to the communication between two processes by means of shared memory and the term *inter-node* to communication by means of a network.

The more recent model $\log_n P$ [17–19] has nothing to do with LogP, although they are similar in name. $\log_n P$ was also conceived, just as LogP, to model point-to-point communication, but with a new key feature, each individual data transfer that occurs in a message transmission. $\log_n P$ introduces the concept of *transfer*, and it describes a point-to-point transmission as a sequence of transfers or movements of data between memory entities, that may vary in number depending on the communication channel. The cost of a transfer is not directly related to hardware, but to the *middleware*, i.e., the software providing services of data movement to the applications, and might include costs from packing/unpacking non contiguous data. Nevertheless, $\log_n P$ is still simplistic and it considers a parallel algorithm as a set of point-to-point message transmissions. Indeed, it has not demonstrated its capabilities to model complex algorithms nor even broadly used basic techniques as the segmentation of messages in shared memory communication. In the author's view, the *transfer* is the building block that conveniently suits the purpose of modeling the behavior of MPI algorithms in shared memory.

1.1 Motivations

Communication performance models provide a theoretical framework for the analytic representation of the communications and their associated costs based on system parameters. Every model operates on a set of input platform-related parameters. An adequate election of such parameters is essential to achieve an accurate cost estimation and a meaningful representation. Processes communicate through different physical channels with very different performance, and usually sharing the available bandwidth. Direct transmissions are possible in the same node using operating system modules, and even in different nodes through high performance networks as *Infiniband*. On the other hand, that platforms execute applications based on parallel programming models different from message passing, as RMA (Remote Memory Access). Performance models have to deal with all these complexities, something hard to represent with the close to hardware LogGP parameters. Important features including advanced transmission mechanisms (such as RDMA and OS bypass), middleware related costs, or network technology, should be captured by an accurate and scalable cost estimation model.

The type of parameters divides the known models in two broad groups: hardware and software. Hardware models, such as Hockney or LogGP, represent communication costs with hardware related parameters, such as network latency or bandwidth. They were initially created for homogeneous mono-processor clusters, and the increasing complexity of modern platforms limits their accuracy. In addition, they show weakness in representing different mechanisms provided by the software such as communication protocols,

or in estimating the impact of the communication middleware on the communication cost. These issues are partially addressed by adding new parameters to include additional costs in the communications (see chapter 2). Nevertheless, these extensions are oriented to very specific platforms and topics. By contrast, software models, such as $\log_n P$, abstract the hardware complexities by the adoption of parameters related to the communication middleware, with the drawback of a possible loss of network technology details.

1.2 Goals

The main goal of this work is to propose a new performance model named τ–*Lop*. It is a software parametrized parallel model aimed to represent parallel algorithms and accurately and scalably predict their cost. Just as $\log_n P$, it is rooted on the concept of *transfer*. It will be shown that it is also capable of capturing issues that are specific to shared communication channels, e.g., the formerly mentioned simultaneous access to shared memory by all the processes involved in a collective, or message segmentation, as discussed by Rico and Díaz in ([20, 21]).

τ–Lop provides a simple and powerful abstraction: to consider a message transmission composed of a sequence of, possibly concurrent, transfers progressing through the multicore cluster hierarchy of communication channels. The transfers reflect data movements between hardware or software agents, and are modeled based on the underlying architecture characteristics. The decomposition of a message transmission into a sequence of transfers allows for an incremental analysis of the algorithm, from a high level representation to low platform-specific details. An atomic point-to-point message transmission could be used as a building block to model a collective operation, but deeper insight can be obtained by further considering the transmission as a sequence of data movements through a communication channel.

Although the decomposition of transmissions into transfers is not a new concept ([17]), τ–Lop goes beyond in natively representing the concurrency in the access to the communication resources shared by parallel transfers. This contention effect has a significant impact on the algorithm performance, so that its consideration in the model results in a substantial improvement in the accuracy of the predicted cost. In stage 3 of Figure 1.1, for instance, eight concurrent transfers contend for the bandwidth of the shared channel, with a cost usually higher than the cost of stage 0. Furthermore, the model considers that a transfer does not have to happen just between a sender and receiver, a restriction of previous models, but also in the so-called *collective transfers*. In this way, τ–Lop provides with an expressive, straightforward and hierarchical representation of algorithms,

covering the point-to-point approach of MPICH, and also the alternative mechanisms adopted by Open MPI shared memory (SM) collectives and the cost of algorithms built under parallel programming models as RMA.

The τ–Lop model addresses the influence of the deployment of processes over the system processors on contention. In a multi-core cluster with hierarchical organization of communication channels, the mapping of processes can improve or aggravate the contention effect. τ–Lop takes into account the way the virtual topology defined by the algorithm is mapped onto the physical topology of the machine. This ability provides a more realistic representation of the algorithm, leading to a better cost prediction, as introduced by Rico and Díaz in [22] and demonstrated by Rico et al. in [23].

1.3 Organization of the Thesis

Each chapter of this document introduces aspects of the proposed model incrementally, which are discussed, evaluated and compared to other model approaches.

Chapter 2 gives a survey of the most important approaches to the parallel analytical modeling of algorithms and applications on current and past platforms, including the MPI communication libraries used in scientific applications. It revisits the LogGP and $\log_n P$ models and examines their weaknesses to model complex algorithms in current platforms.

Chapter 3 introduces the basic aspects of the τ–Lop model, the parameters and their meanings in comparison with other models, and the methodology to build a model from them. Point-to-point communication modeling is introduced, including different types such as Send-receive or Exchange of messages. A general expression describing communication between two processes is developed. This chapter shows the impact of the contention on a shared communication channel, and how τ–Lop manages that contention, taking as an example shared memory. As well, the *concurrency operator* is defined, an operator that allows to expressively represent concurrent point-to-point message transmissions sharing a communication channel.

Chapter 4 explains τ–Lop application to key algorithms of MPI collectives, besides introducing its potential as an analytical inference model for the prediction of the communication cost. One shared channel is considered for simplicity, and some complex algorithms are experimentally evaluated against the actual measured cost and the estimations of other models.

Chapter 5 extends the model to cover the issue of the communication channels hierarchy. Processes in multi-core clusters communicate through shared communication channels with uneven performance depending on their placement in the system, called the mapping of the processes. The performance of the collective algorithms depends highly on such mapping, and the models have to consider this fact in the estimations.

Chapter 6 exhaustively shows the capabilities of the τ–Lop model through a two cases of study. Their complexity exceeds that commonly addressed in previous related works, which allows to state that the model can be used and adapted to broadly available platforms and systems.

Chapter 7 covers a critical aspect in a model, the parameter measurement methodology. The accuracy in the cost estimations of a model highly depends in the correct assessment of its parameters. The methodology has to measure the behavior of the channels under contention, hence a detailed approach based on a set of simple operations is described, which can be extended for accuracy in a concrete platform by adding experiment costs and applying a linear regression method. This chapter also describes the approach to report the error in the cost estimation of the measurements along this work.

Finally, chapter 8 gives the conclusions of the work.

This thesis also includes an appendix for on-work extensions to the model exploring its capacities in the field of the heterogeneous computing systems. Appendix A describes the application of the model to the evaluation of the communications of hybrid scientific applications executing on heterogeneous platforms. Current platforms trend is to include different processing elements with different characteristics of performance, such a combination of CPU multi-core nodes and GPUs. Each process running in these platforms has to be provided with an unbalance computation load in order to balance the global system workload. However, the communications in this type of applications are not addressed for optimization, and it is usually done experimentally. τ–Lop has been extended to achieve the goal of helping in the modeling of these application communications and providing a framework to facilitate their optimization.

Chapter 2

Related work

Next, a summarized survey of the large amount of available literature about performance modeling is provided. Models are classified in *hardware* and *software*, based on the type of parameters each model is composed. Then, extensions for that models, as well as new proposals, covering different features of modern architectures, are analyzed and their limitations discussed. Through this document, a new proposed model is compared with the representative mainstream models of each type, and when necessary, the discussion includes extensions to the models which cover aspects of the specific platforms considered. Finally, libraries implementing the MPI standard used in this document to evaluate the model discussed are introduced.

2.1 Hardware models

Most of the current communication performance models derive from Hockney [1] and LogP [3], oriented to support the message passing model on mono-processor clusters. They model an algorithm as a sequence of point-to-point transmissions, with the goal of organizing the computing loads and the communication events to optimize its execution time in the cluster. Both are linear models, in the sense that they represent the point-to-point cost by a linear function that involves the size of the message and a set of network-related parameters. Together with benchmarks measurements, these simple models have been used for comparison of different MPI collective algorithms in a given distributed system, for ranges of message sizes and numbers of involved processes. Chan et al. [24, 25] analyze some MPI collective algorithms and propose optimization for their implementation in distributed memory systems. Thakur et al. [26, 27] use the Hockney model to guide the selection of one among multiple algorithms for a collective

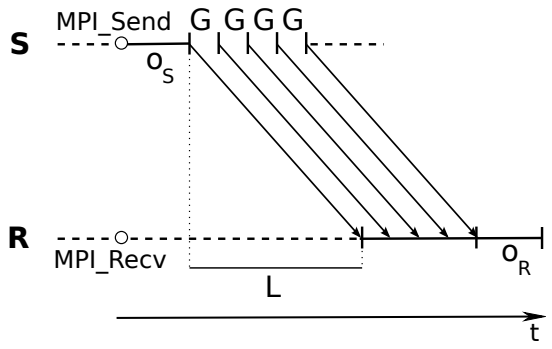


FIGURE 2.1: A point-to-point message transmission cost represented as the overhead in the sender (o_S) plus the gap per byte (G) multiplied with the number of bytes of the message ($m = 5$), and the latency of the network L, and finally the receiver overhead o_R. In the transmission of an isolated message, g parameter is not present.

operation depending on the message size, with the goal of improving the overall performance in clusters with switched networks running the MPICH library. Other works address the development and optimization of efficient algorithms for a specific range of MPI collectives: a Hockney based optimal Broadcast is developed by Träff [28] and evaluated in clusters of SMP nodes, algorithms for the *reduction* operations are proposed by Rabenseifner et al. in [29], and the *Alltoall* collective is discussed by Kale et al. in [30].

LogP characterizes the cost of a point-to-point message transmission according to the four parameters that compose its name: network delay (L), the overhead or time invested by the processor in sending or receiving a message (o), the minimum time interval between message transmissions (g) and the number of processors (P) in the cluster. With respect to Hockney, the more advanced LogP cost model splits the network latency in the network delay (L) and the processor overhead (o), what allows to model the computation and communication overlap. It supposes that a long message is sent as a sequence of shorter messages, using the minimum time interval between message transmissions (g). Culler et al. [2] present and analyzes some example algorithms commonly used in applications, such as Broadcast, summation, or FFT.

Alexandrov et al. LogGP model [4] appears as a new model that extends LogP with an additional parameter, gap per byte G, the time between two consecutive byte transmissions. Following the LogP convention, LogGP characterizes the communication cost according to the following parameters: the network latency (L), the overhead or time a processor is engaged in the transmission or reception of a message (o), the time interval between consecutive message transmissions (g), and the time interval between consecutive byte transmissions (G), which captures the network bandwidth for long messages. P is the number of processors in the cluster. Figure 2.1 shows the point-to-point transmission of a message between two processes. The cost of the transmission is formulated in terms of the parameters and the size in bytes of the message (m), as

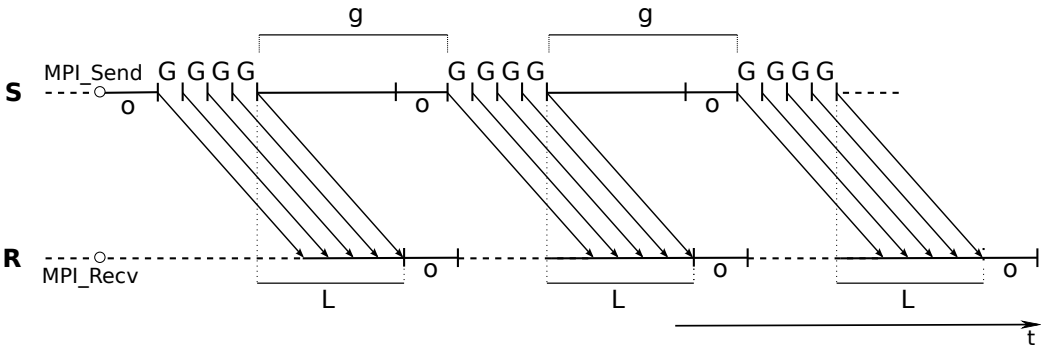


FIGURE 2.2: Transmission of a point-to-point message in $k = 3$ segments of $S = 5$ bytes in LogGP.

$o_S + L + (m-1)\,G + o_R$, i.e., the overhead in the sender processor, the latency in the network of the first byte, the time of sending the rest $m-1$ bytes of the message, and the overhead in the reception processor. For simplicity, it is usually assumed that $o = (o_S + o_R)/2$ and $g \leq L + o$. Conclusions are not affected by these simplifications which are consistent with measured parameter values. Hence, the cost of a point-to-point message transmission of size m is:

$$T_{p2p} = 2\,o + L + (m-1)\,G \tag{2.1}$$

Segmentation of messages is a technique commonly used by MPICH and Open MPI in shared memory transmissions. It operates by breaking up the message into smaller chunks known as segments and sending them in sequence. Figure 2.2 represents a point-to-point transmission of a message, split into k segments ($k = 3$) of size S. The cost is modeled as:

$$T_{p2p_s} = 2\,o + L + (S-1)\,G + (k-1)\,\big(g + (S-1)\,G\big) \tag{2.2}$$

As will be explained later, the segments allow overlapping of the sender copy (to the shared buffers) with the receiver copy (from the shared buffers), accelerating the communication. These buffers are not explicitly represented in the LogGP model.

LogGP, together with the aforementioned models, claim their ability to model the point-to-point message transmission primitives of the MPI standard, as shown by Al-Tawil et al. in [31], and upon them, the message passing algorithms underlying the collective primitives of MPI, on mono-processor clusters. For instance, LogGP understands the Binomial tree algorithm supporting the *MPI_Bcast* collective as a sequence of parallel point-to-point transmissions (as with Hockney). It estimates its cost by simply multiplying the cost of a stage with the height of the tree:

$$T_{bcast} = \lceil \log_2(P) \rceil \times T_{p2p} \tag{2.3}$$

As a result, for instance, the cost for $P = 65$ and that for $P = 128$ in a multi-core will be the same, which is far from being a correct prediction.

Thorough studies of known collective algorithms and their performance evaluation under the LogGP model can be found in Pješivac-Grbović et al. [13] and Hoefler [9].

Parallel algorithms are also modeled in [13] using PLogP [32, 33]. The *Parametrized* LogP model slightly changes the meaning of some parameters of LogP and makes them (except latency) dependent on the message size. This model is also applied to a wide range of collective algorithms in [34] by Estefanel and Mounié. The cost of a point-to-point transmission modeled under PLogP is $T_{p2p} = L + g(m)$, with L the end-to-end latency from process to process, and $g(m)$ the gap, i.e. the minimum time interval between consecutive message transmissions. The gap contains the overheads $o_s(m)$ and $o_r(m)$, the time intervals the CPUs on both sides are busy sending and receiving. The definition of the parameters as a function of the message size m, main contribution of the PLogP model onto LogP, improves the accuracy of the model.

2.2 Software models

The increasing complexity of multi-core cluster architectures and the implementation of different intra-node communication techniques by current MPI libraries, lead to the necessity of new approaches to performance modeling. Albeit similar in name to LogP, Cameron et. al $\log_n P$ [17], [19], [18] is a different model addressing the *middleware* costs of the communication. The $\log_n P$ model assumes that a message transmission takes place between two processes as a sequence of *implicit* transfers (copies) between the source and the destination buffers. The cost of a transmission is the sum of the costs of its n individual transfers:

$$T_{p2p} = \sum_{i=0}^{n-1} \left(max\{g_i, o_i\} + l_i \right), \tag{2.4}$$

where o (*overhead*) is the amount of time the processor is engaged in each transfer for a contiguous message, which is represented by g (*gap*) when contention time is considered, and l (*latency*) is the additional processor time when the message is not contiguous in memory. Figure 2.3 shows an example of a message transmission in shared memory. Here $n = 2$, because the transmission needs just two transfers going through an intermediate buffer. On contiguous messages $l_i = 0, \forall\, i$ and, if the next reasonable simplifications $max\{o_i, g_i\} = o_i$ and $o_i = o\ \forall\, i$ apply, then (2.4) becomes

$$T_{p2p} = 2\, o \tag{2.5}$$

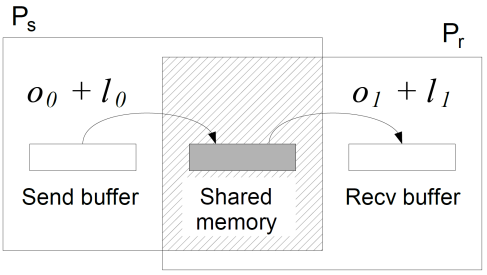


FIGURE 2.3: Concept of *message transmission* in $\log_2 P$. P_s and P_r are the sender and receiver processes respectively. The message transmission occurs in two *transfers* through the shared memory channel. Each transfer has different cost parameters, but equaling them is a reasonable approach in this context.

These foundations lead to simplistic modeling expressions of algorithms. For instance, the cost of the Binomial tree broadcast with contiguous messages is modeled, similarly to LogGP, as:

$$T_{bcast} = \lceil \log_2(P) \rceil \times T_{p2p} = \lceil \log_2 P \rceil \, (n\, o) \tag{2.6}$$

The *transfer* concept conveniently suits the purpose of modeling MPI communication algorithms. Nevertheless, this model lacks of several features commonly found in the current middleware implementing the MPI standard. First, although segmentation has a strong impact on the MPI intra-node performance, it is not explicitly considered in $\log_n P$, even though the segmented and non-segmented transmissions of a message give different values of g and o. It considers a message cost as a half of the actual measured cost of a point-to-point message transmission in shared memory. Second, works introducing the model do not define neither the contention meaning nor its measurement[1] (g parameter). Third, although the model is recent, it does not represent the mixture of communications through the different channels present in current systems. Last, the packing cost is usually not attributable to a point-to-point transmission when it is executed in the context of a collective operation, but the cost is included at the beginning and end of the collective.

2.3 Other models and extensions

In addition to new model proposals, extensions of the mainstreams models contribute with a variety of add-ons to represent different aspects of the communication in the complex modern HPC platforms. Amongst these, specific mention can be made of costs derived from specific network technologies, different types of communication and synchronization, and heterogeneous architectures. For instance, extensions of the LogGP model addressing the former issues include the following. Ino et al. LogGPS [35] covers

[1] In fact, except enumeration of the parameter, the contention is not dealt with this works.

aspects of synchronization, providing the rendezvous cost as a new parameter S, which represents the cost of the communication protocols used in the data transmission. Chen et al. LogGPO [36] accurately captures the overlap of communication and computation in non-blocking transmissions. More specific cost models fitting different network technologies also exist. Hoefler et al. LogfP [37] models short messages in Infiniband networks. The authors use the model to study the effects of multi-stage switches on that kind of networks in [38]. Paper [39] by Kim and Lee analytically estimates the communication delays in Myrinet networks.

Next, some of the proposals are summarized, with a particular focus on those facing the *hierarchy* of the communication channels, the *contention*, and the *heterogeneity* of clusters.

2.3.1 Hierarchy

Modern HPC clusters are composed of multi-core nodes connected by high performance networks, which allows a hierarchical classification of the communication channels based on their different performance. A process communicates with the rest using different channels depending on its placement, for instance, using shared memory if the destination executes in the same node, or through the network if the destination is in a different node. The most known model covering communication hierarchy derives from LogGP. Yuan et al. LogGPH [40] supports representation for hierarchical architectures using a different set of parameter values for each communication channel. The parameters are measured separately and used to represent the cost of an algorithm as a composition of point-to-point transmissions progressing through different channels. Unfortunately, algorithms usually show a complex behavior that usually makes difficult to separate simultaneous communications through different communication channels, as it will be shown in chapter 5. In the other hand, LogGPH can analytically model a kind of algorithms developed to perform in a hierarchical way on multi-core clusters ([41]). In a broadcast, for instance, first, an inter-node broadcast is executed between the representative process in each node, and then a local shared memory Broadcast runs inside the node with the representative acting as root. These *hierarchical* algorithms are modeled in phases, involving different communication channels.

In the software models side, Tu et al. $m\log_n P$ [42] extends $\log_n P$ by distinguishing the variety of communication channels[2] in a system. In multi-core clusters, for instance, intra-socket, inter-socket and inter-node communication channels can be found, performing in a hierarchy. The locality of a process determines the communication channel used

[2]We will use the term communication *channel*, though the term communication *level* is used in the original definition.

for communicating to another process, and so the communication cost. When only one communication channel is considered, $m\log_n P$ and $\log_n P$ become equivalent.

2.3.2 Contention

Concurrent access to a shared communication channel shrinks its available bandwidth and degrades the overall performance. Modeling the cost of those common situations is required to provide with an accurate estimation, for instance, of collective operations.

Sound works directly related to the contention modeling issue in high performance networks are carried out by Bravetti et al. in [43], Martinasso et al. in [44] and by Jerome et al. in [45] for Ethernet, Infiniband and Myrinet networks respectively. The authors generate specific models for each type of network based on the study of network flow control mechanisms and experimentally deduce *penalty coefficients*, derived from resource sharing experiments, and categorize different types of conflicts. Zhu et al. [46] show that a model needs to address the contention derived from bidirectional TCP communication in a full-duplex Ethernet network, which diminishes the maximum reachable network bandwidth. Another LogGP derivative, the LoGPC model [47] proposed by Moritz and Frank, introduces a large set of parameters to evaluate the contention in mesh networks with wormhole routing. Work by Steffenel in [48] introduces the *contention factor*, represented as a parameter exclusively affecting the modeling of the network performance, which is incremented linearly from the contend-free communication cost. This paper evaluates the *Total Exchange* collective under this assumption.

In general, contention-aware studies provide hardware models very close to the specific network technology. They usually deal with a limited number of nodes and, even more important, with a short number of cores per node. These works do not address the shared memory effects in the overall communication cost and, hence, omit the impact of the virtual topology of processes on the cost of the collective algorithms. Besides, only a limited set of simple collectives have been evaluated against the generated models.

2.3.3 Heterogeneity

In addition to the different performance of the communication channels in an hierarchical system, an heterogeneous system includes processing elements with differences in performance and features. Currently, most common HPC heterogeneous systems are composed of multi-core CPUs and GPUs.

Bhat et al. work [49] proposes a model-based framework for developing efficient scheduling algorithms for the Broadcast and multicast collective communication patterns in

heterogeneous platforms, including processor and network heterogeneity, and provides performance evaluation mechanisms.

Scheduling algorithms for efficient Gather operation in distributed heterogeneous systems and their performance evaluation using a communication performance model are addressed by Hatta and Shibusawa in [50], while Ooshita et al. [51] found an optimal Gather scheduling time for an heterogeneous cluster system characterized by the number of the processors and the number of processors of distinct types.

HLogGP [52] is a model based on LogGP that takes into account both the processor and network heterogeneity of a system. The scalar parameters of LogGP are expanded to represent the o_s, o_r and g values of the p processors as vectors of p components, and the L and G values of each pair of processors as matrices of $p \times p$. HLogGP provides with a methodology to measure the parameters and validate the estimations in a small cluster.

Lastovetsky et al. work in [53] addresses hierarchical communications including an Ethernet network connecting a set of heterogeneous processors. The model proposed, LMO, includes the impact of the heterogeneous processors on the global communication time for a set of point-to-point, one-to-many (Scatter and Gather) and broadcasting operations. LMO and its parameter measurement procedure is deeply discussed by Lastovetsky et al. in [54] and [55]. LMO is a communication performance model aimed to estimate the cost of algorithms in heterogeneous, in addition to homogeneous, systems. Similar to LogGP, it carefully separates the cost related to the processors and the network in order to gain more accurate communication cost predictions. The cost of a message transmission between the processors i and j is:

$$T_{i \to j} = C_i + m\,t_i + C_j + m\,t_j + \frac{m}{\beta_{ij}}$$

C_i and C_j are fixed processing delays, and reflect the processor heterogeneity. t_i and t_j are delays of processing a byte, and reflect the communication channels heterogeneity. β_{ij} is the transmission rate of the channel connecting the processors. A disadvantage of this model is the amount of parameters to be measured, which tends to p^2 in a platform with p processors.

2.4 Libraries

Three implementations of the MPI Standard have been considered to evaluate the models described in this document: the well known and broadly used MPICH [15] and Open MPI [16] libraries, based on classic processes, and AzequiaMPI, a complete implementation

of the MPI 1.1 Standard based on threads, described by Díaz and Rico in [56, 57]. The Intel IMB benchmark suite ([58]) is the tool used for performance measurements of the point-to-point and collective communications in the implementations

2.4.1 MPICH

MPICH builds collectives upon the point-to-point communication library *Nemesis* [59]. Nemesis communicates two processes in the same node by lock-free message queues of, by default, 64 KB memory blocks called *cells*[3]. Queues and cells are allocated in a memory area mapped by each local pair of processes.

MPICH-Nemesis uses other communication protocols for improving the performance of communications in shared memory. For instance, to improve the short message transmission latency, MPICH-Nemesis sends small messages using a technique known as "fast box", an intermediate buffer per local pair of processes with boolean flag that determines if data is ready to be read or written.

2.4.2 Open MPI

Open MPI promotes a software architecture based on components (*MCA*, for Modular Component Architecture). Every functionality is paid by a well defined interface known as *framework*. An MCA framework uses the MCA's services to find and load *components* at run time. An MCA component is a standalone collection of code that can be inserted into the Open MPI code base, either at run-time and/or compile-time. *COLL* is a framework for collectives including the *Tuned* and *SM* (shared memory) components. Tuned component implements collectives similarly to MPICH, as a sequence of point-to-point transmissions between the involved processes. SM component implements collectives based on a different approach. It does not use point-to-point communication to build collectives, but it maps a memory zone which is common to all the participant processes for interchanging messages in shared memory.

2.4.3 Operating system modules

Operating system modules can provide services of data movement between processes without an intermediate copy in shared memory. This technique improves the transmission bandwidth of large messages. However, it has proved to be inefficient in short

[3]The size of a *cell* can be changed in the implementation, and so it is done in most of the examples of this thesis (to $8\,KB$). The reason is to increment the number of segments fitting in the message sizes considered in the measurements.

messages due to the heavy overhead introduced by the system calls. Examples of this approach are *LiMIC* [60] and *KNEM* [61]. KNEM transfers data from one process to another through a single copy within the Linux kernel. It is used by MPICH-Nemesis and some components of Open MPI to improve point-to-point messages. Paper [62] presents a KNEM extension that implements collective operations directly in a Linux kernel module, and achieves bandwidth improvements in the IMB benchmark between 40% and 75% with respect to Open MPI. A similar approach is achieved by Rico and Díaz in the implementation of an Open MPI component for executing collective operations in [63].

Operating system modules suppose a meaningful example of the importance of the middleware costs, which a model needs to represent to accurately predict the cost of applications in current HPC platforms.

2.4.4 Thread-based MPI

Thread-based MPI implementations are inherently more efficient in shared memory than their process-based counterparts. Threads executing in the same node share the same address space, which allows message transmissions in a single transfer without a significant overhead, and precludes reserving additional common mapped memory. This issue is becoming more and more important given the current trend to reduce the memory per core rate in modern multi-core platforms. *MPC-MPI* [64] and AzequiaMPI are MPI thread-based implementations. They use the same techniques as MPICH and Open MPI for inter-node communication, and, in the case of AzequiaMPI, it uses *fast boxes* to improve the latency of the very short messages in shared memory.

Chapter 3

Proposal: The τ–Lop model

This chapter describes the goals and features of the proposed model, named τ–Lop. Parameters of the model are introduced, and the modeling of point-to-point messages in shared memory and network communication channels are discussed, including commonly used techniques as message segmentation. Furthermore, the concurrency operator is introduced to represent the contending transmissions through a communication channel. Finally, the chapter shows a generic expression representing the cost of a a point-to-point transmission, which supposes the basis of the model.

3.1 Motivations and goals

Main τ–Lop key challenge is to represent and accurately model the cost of MPI operations, both point-to-point and collectives, when deployed on multi-core clusters, composed of a hierarchy of communication channels that allow concurrent transfers.

The goals of the model are twofold:

- Representing the cost of the algorithms underlying the implementation of collective operations and applications. Although simple representation is important, it is necessary to model the techniques actually used in the point-to-point and collective operations in current MPI implementations. In order to grasp a realistic view that allows their optimization and tuning. Zero-copy transmissions with the support of the operating systems or the commonly used segmentation of large messages in shared memory are two examples of communication which need a more explicit modeling that provided by current models. Beyond the point-to-point transmissions, complex scenarios need to be modeled and analyzed providing increasing levels of abstraction, to extract useful information from their representation.

- Accurately and scalably estimate the cost of algorithms and applications. The choice of the parameters and their assessment method are critical to reach a good level of approximation to the actual cost of an algorithm running in an specific platform. As well, platform characteristics have to be captured in the parameter values in order to achieve scalability in the predictions.

τ–Lop achieves the first goal introducing the concept of concurrent transfer. A concurrent transfer is the building block of a point-to-point transmission. It allows to abstract from hardware complexities and enables a constructive modeling method to represent MPI communications in a more intuitive way. These characteristics of the model are covered in this and the next chapter, together with its accuracy in the estimation of algorithms costs.

The second goal is faced with an incremental methodology for estimating the values of the parameters. This methodology is based on benchmarking a simple *RING* operation, and can be extended for improving the accuracy with the addition of more operations. These issues are discussed in chapter 7. The obtained values are used in the prediction of communication costs along the examples.

3.2 Communication issues covered by τ–Lop

In order to fulfill goals, a model needs to accomplish some of the issues commonly presented in data communication on current platforms. τ–Lop has been designed to model the next ones:

Concurrent transfers: A cost model has to consider the fact that a communication channel bandwidth shrinks when transfers are concurrent. The cost of a transfer (a copy or data movement through a channel) is represented as $c(m, \tau)$, which is a function of the message size (m) and of the number of concurrent copies on the channel (τ). Figure 3.1 represents the cost of broadcasting a buffer by a process running in core 0, that is read by $P = 1 \ldots 128$ cores of a shared memory ccNUMA machine. If all the copies are simultaneous (concurrent), the memory bandwidth gets exhausted. As a result, the cost grows with the increase of the number of transfers. Current models do not consider this fact, which leads to a lack of scalability in their cost estimations.

Collective transfers: A transfer need not be just between a sender and a single receiver, a restriction of $\log_n P$ and LogGP and their derivatives. For instance, as will be shown later in chapter 4, the Open MPI COLL SM component implements the *MPI_Bcast*

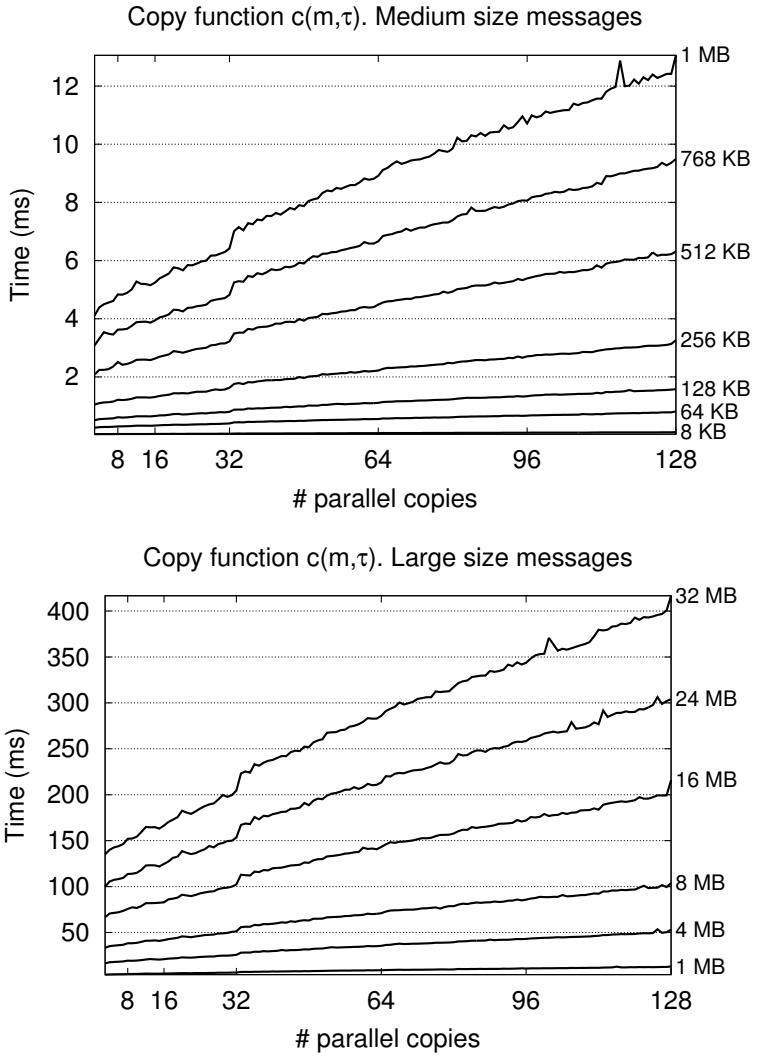


FIGURE 3.1: Cost of a copy as a function of message size (m) and the number of processes concurrently copying (τ) in *Lusitania*, a 128-core NUMA machine. All processes copy from the same memory buffer, stressing the communication channel

collective operation using shared memory regions that are written by a sender and read simultaneously by several receivers in a collective transfer.

Message segmentation: Transmissions that require an intermediate copy, as those between processes in shared memory, generally use message segmentation techniques for large messages. Segmentation demands less intermediate memory because storing the whole message is not needed. It comes with the additional cost generated by synchronization of the processes in the interchange of segments, as well as by the effect of concurrent copying by the sender and the receiver. Despite that, it speeds up the communication progress by overlapping the sending of segments with their reception [59]. Other techniques exist for improving message transmission in shared memory. Operating system modules, as KNEM, provide services of data movement between processes

without an intermediate copy, as well as other MPI implementations as MPC-MPI and AzequiaMPI make direct copies in shared memory between ranks address spaces, by implementing ranks as threads.

Costs attributable to protocol: The MPI standard includes a wide range of communication modes, including the synchronous send (*MPI_Ssend*), which enforces sender process to wait until receiver arrives. In addition, limitations on buffer space impose MPI libraries to set up communication protocols, such as *rendezvous*, so that the sender waits for the receiver, who on arrival notifies the sender to proceed. Protocols charge additional cost, mainly at the beginning of the communication, which must be considered in a model.

Computational cost: Collectives such as *MPI_Reduce*, which involve application of an algebraic operation to the message data, add cost to the communication. Their modeling is direct, and although is beyond the scope of this document, some examples of modeling and cost estimation of MPI collective algorithms including computation are provided.

3.3 The parameters of the model

To represent the foregoing communication issues, a set of six parameters are defined below for τ–Lop:

L: *Transfer time* (L) is the cost of a transfer and is represented as $L(m, \tau)$. As the transfer may flow concurrently with others (see Figure 1.1), its cost depends not only on the message size (m), but also on the number of such concurrent transfers (τ). The definition of L, hence, enforces the restriction that the cost of A concurrent transfers will be some value between the cost of a single transfer and that of A consecutive ones, $L(m, 1) \leq L(m, A) \leq A \times L(m, 1)$, an expression that is generalized for convenience as follows:

$$L(m, \tau) \leq L(m, A \times \tau) \leq A \times L(m, \tau)$$

As well, as assumed by most models, the transfer time cost grows linearly with the increase of the message size, named the *linearity principle*, and hence:

$$L(A \times m, \tau) = A \times L(m, \tau)$$

In a transfer, the model considers not only the cost that is attributable to the bare copy $c(m, \tau)$, but also the cost due to *synchronization* management $y(\tau)$, defined as the time that the processes involved in the τ transfers invest on the synchronizing resources. Each of the τ transfers involves normally two processes, except in the special case of collective

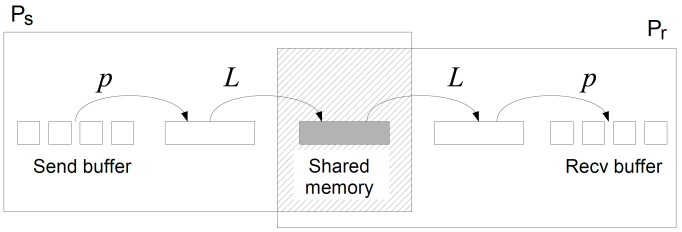


FIGURE 3.2: A *message transmission* in τ–Lop is composed by two *transfers* on the shared memory channel.

transfers. $y(\tau)$ can represent the time spent polling a flag or using synchronization objects, such as *mutexes* or lock-free queues. The transfer time is hence defined as the sum $L(m,\tau) = c(m,\tau)+y(\tau)$. Though $y(\tau)$ is difficult to measure empirically, it has been observed to dominate the global cost of tiny messages. In particular, as $c(0,\tau) = 0$, the cost to transfer a null message is due to synchronization alone. This definition allows τ–Lop to model the cost of operations without data transfer as *MPI_Barrier*, where synchronization is the exclusive cost.

p: *Packing* cost (p) is the cost due to *pack* and *unpack* a non-contiguous message. It depends on the message size $p(m)$, as well as on the layout of message fragments in memory. It is library specific, because each MPI library implements its own data type management procedure. It applies only to the first and last transfers of the operation, be it point-to-point or collective (see Figure 3.2). For simplicity, only contiguous messages ($p(m) = 0$) are considered in the following, unless otherwise specified.

o: *Overhead* cost (o), despite its name, has nothing to do with the parameter of the same name in $\log_n P$. It is defined as the time elapsed since the invoking of a message transmission operation until the beginning of data injection into the channel. Function of the message size, $o(m)$, is the sum of software stack O_S and the protocol O_P. It is a step function with threshold H, so that $o(m) = O_S$ if $m < H$, $o(m) = O_S + O_P$ if $m \geq H$. The implementation of shared memory collectives in the studied libraries MPICH and Open MPI does not use protocol activities, so $O_P = 0$, however, this value is meaningful in transmissions progressing through the network. As regards O_S, it is significant only on transmissions of small messages, and hence this value can be ignored in practice most of the times.

γ_{op}: Represented as $\gamma_{op}(m,\tau)$, it is the time invested in computing *two* vectors of size m by the arithmetic operation *op*. The number of elements in m bytes depends on the type of the data. Parameter γ_{op} depends on the number of simultaneous operations in course (τ), because of the need to access the channel to obtain the operands. This parameter allows to represent reduction operations defined in the MPI standard.

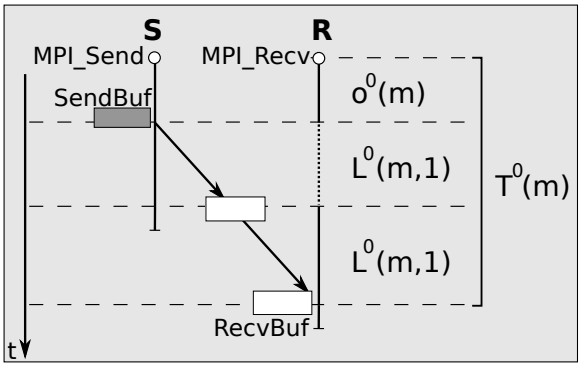


FIGURE 3.3: τ–Lop representation of a contiguous message transmission through the shared memory communication channel ($c = 0$) including the overhead cost.

Two additional parameters are usually referred in the modeling of algorithms, specially in collective operations and multi-core clusters:

P: Number of processes involved in the MPI operation.

N: Number of different communication channels in the system.

3.4 Modeling message transmissions

τ–Lop uses the concept of *concurrent transfers* and middleware-related parameters to model communications. It represents a point-to-point message transmission between two processes as a sequence of transfers (data movements), progressing through a communication channel. Its cost is denoted as $T(m)$ and it indicates the cost of transmitting a message of size m. It is always applied to a contiguous message. Under τ–Lop, the cost of packing and unpacking a non-contiguous message is attributed to the operation, with the cost denoted as $\Theta(m)$, so that, $\Theta_{p2p}(m) = 2\,p(m) + T(m)$.

Next, the transmission cost modeling in several scenarios is addressed, finally concluding in a generic definition representing a point-to-point message transmission which supposes the basis of the model.

3.4.1 Shared memory transmissions

The most simple contiguous message transmission in the shared memory communication channel ($c = 0$)[1] between processes in MPI implementations is done through an intermediate buffer. This buffer is mapped to a common memory zone accessible to sender

[1]The communication channels in a system are identified by a number (c) starting from 0, the number usually assigned to the channel with the best overall performance. The context makes clear when c refers to the communication channels or the copy function $c(m, \tau)$.

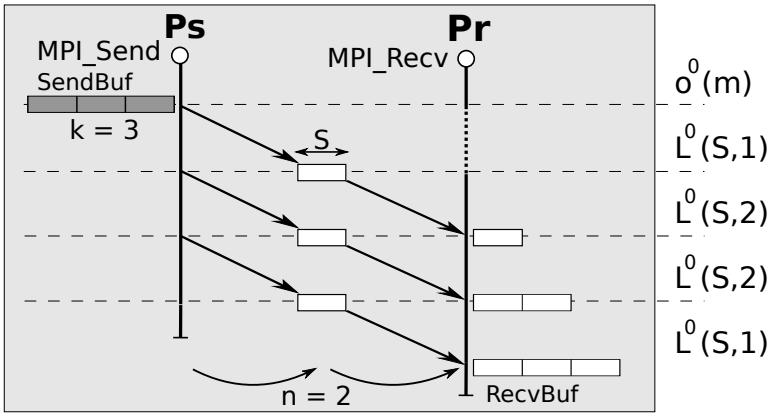


FIGURE 3.4: A shared memory transmission of a message divided up in $k = 3$ segments of size S. It needs a total of 4 transfers (L^0).

and receiver processes. Figure 3.3 shows the message transmission. The transmission starts after a time represented by the overhead parameter and requires two transfers ($n = 2$). The first transfer copies the data to the intermediate buffer, hence with the cost $L^0(m, 1)$. When data is ready in the intermediate buffer, the receiver starts the second transfer to its user buffer. The total cost of the operation is represented as:

$$T^0(m) = o^0(m) + L^0(m, 1) + L^0(m, 1) \tag{3.1}$$

There are not concurrent transfers in this simple scenario. This is the common mechanism adopted by widespread implementations MPICH and Open MPI for transferring small messages between separate-address processes, with slight modifications.

3.4.2 Representation of other transmission techniques in shared memory

Next, commonly used techniques for transmitting messages in shared memory are discussed and modeled using τ–Lop. These techniques reflect the importance of the middleware cost in the transmission of messages in current platforms.

Message segmentation. As introduced previously, segmentation of large messages is a common technique used in shared memory for improving the transmission time. Segmentation consists in dividing the message in the sender user buffer in segments of size S. The technique uses a set of intermediate buffers of the same size S, and it overlaps the transfers of the segments from the sender user buffer to the intermediate buffers, with the transfers of segments to the receiver user buffer.

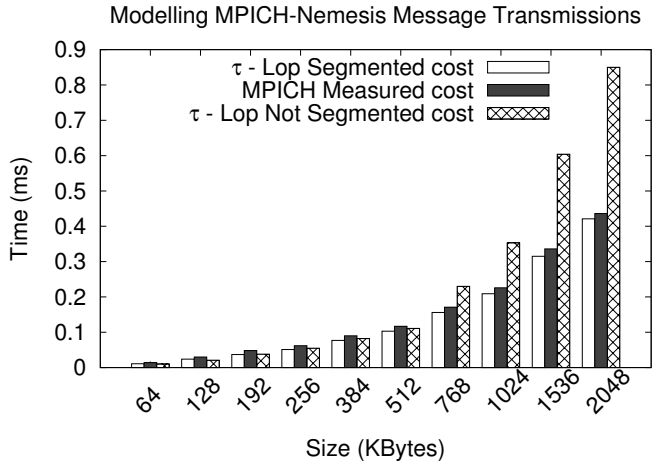


FIGURE 3.5: MPICH empirically measured point-to-point transmission time compared to its segmented and non-segmented cost modeling with τ–Lop. The target machine is *Metropolis* (see section 7.1.2), and processes are bound to cores in the same socket, hence sharing L3 cache memory.

Figure 3.4 illustrates how a segmented transmission flows in a number of stages (s), a figure that depends on n, the number of transfers each segment needs to reach the receive buffer, as well as on k, the number of segments the message is split into, so that $s = k + n - 1$. n depends on the location of processes, as well as on the software mechanisms used, which can vary with the message size or with the communication mode. $m = S \times k$, S being the constant size of the segment in the implementation. The total number of transfers in a message transmission is $k \times n$. Figure 3.4 shows the transmission of a message broken into $k = 3$ segments with $n = 2$, hence with a total of $k \times n = 6$ transfers and $s = 4$ stages. Note how the first and last transfers proceed alone, while the other four proceed in concurrent pairs, with a total cost of:

$$T^0(m) = o^0(m) + 2\,L^0(S,1) + (k-1)\,L^0(S,2) \tag{3.2}$$

The cost (3.2) becomes that of the single transmission already defined in (3.1) when the message size is smaller than segment size ($m \leq S$).

Note that (3.1) equals the cost given by $\log_n P$ in (2.5); so, $\log_2 P$ could be considered as a particular case of τ–Lop when $n = 2$ and $k = 1$, namely, when τ–Lop ignores the segmentation and the contention.

Segmentation is not represented in other models. For example, $\log_n P$ ignores the internal structure of the transmission and hence $T(m)$ becomes a black-box that has to be measured for every value of m. The only parameter in (2.5) is the overhead o, which is directly calculated as half of a MPICH ping, hence the estimation error of a point-to-point message in $\log_n P$ is simply zero. While $\log_n P$ *measures* $T(m)$ for every m, τ–Lop

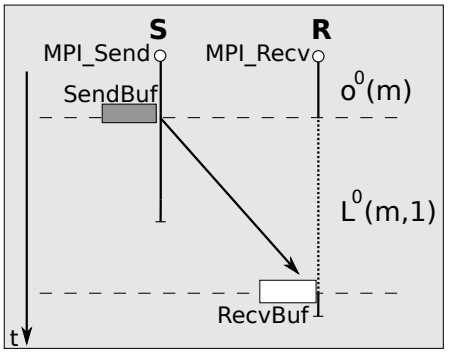


FIGURE 3.6: A zero-copy message transmissions ($n = 1$) in MPICH-KNEM using the operating system. There are a total of $s = 1$ stages.

estimates it as the suitable composition of the $L(S, \tau)$ terms, which makes a difference. Figure 3.5 compares costs from applying definitions (3.1) and (3.2) with respect to the real measured cost of MPICH in shared memory. The message size m ranges from 64 KB to 2 MB, and the segment size is $S = 32KB$, producing an average proportional error of $\mu = 1.16$ in (3.2), while achieving $\mu = 1.31$ using (3.1) (see section 7.6 for the definition of the *proportional error*). The overlapped interchange of segments of sender and receiver reduces the transmission cost, an effect that τ–Lop captures with reasonable accuracy.

Zero-copy transmissions. MPICH-KNEM sends a long message through shared memory in a single transfer ($n = 1$) using operating system services. This is the same mechanism used in thread-based MPI implementations as AzequiaMPI. It results in $s = 1$ stages, shown in Figure 3.6, and the cost becomes $T^0(m) = o^0(m) + L^0(m, 1)$.

The former KNEM cost halves the cost of a simple point-to-point transmission defined in (3.1). However, overhead in calling the operating system is high, and for small messages the overhead dominates the overall cost, so this technique could be used just for long message transmissions.

Buffered transmissions. The buffered transmissions (*MPI_Bsend*) could be considered as composed of three transfers ($n = 3$). Copying data to an internal buffer makes the sender process to return as fast as possible from the function invocation, leaving the transfer to the middleware, which sends the data in two transfers through a shared memory buffer. In this case the formulation is quite similar, with a cost $T^0(m) = o^0(m) + 3\,L^0(m, 1)$.

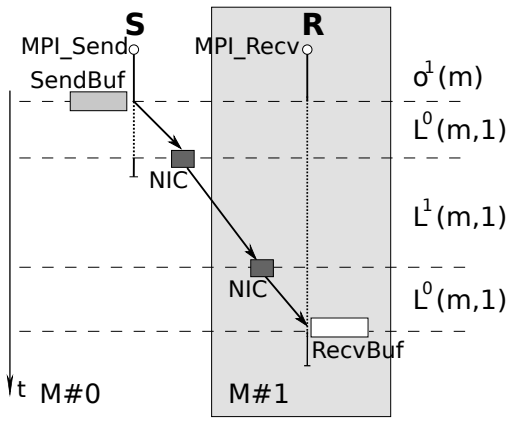


FIGURE 3.7: τ–Lop cost of a point-to-point network transmission.

3.4.3 Network transmissions

In the network channel (channel number $c = 1$), τ–Lop follows the same scheme by Cameron et al. in [18], that consider a simple message transmission between two processes as composed of 3 transfers: first, from the sender buffer to the internal buffer of the NIC in the sender node, second, to the NIC in the receiver node, and last, to the receiver buffer. τ–Lop considers both the first and last transfers as shared memory transfers (L^0), whereas the second transfer progresses through the network (L^1), as shown in Figure 3.7. The cost is represented as:

$$T^1_{p2p}(m) = o^1(m) + 2\,L^0(m,1) + L^1(m,1) \tag{3.3}$$

As before, the number of transfers a network transmission is composed of, depends on the network technology. For instance, some networks as Infiniband allows the direct transfer of data to the buffer of the receiver using a Remote Direct Memory Access mechanism.

3.4.4 Message transmission definition

With regards to the former results, the point-to-point transmission cost of a message of size m, through the communication channel c, is represented in τ–Lop as a two-terms sum, the overhead and the aggregated cost of the individual transfers:

$$T^c_{p2p}(m) = o^c(m) + \sum_{j=0}^{s-1} L^c_j(m/k, \tau_j) \tag{3.4}$$

That is, the overhead (o^c), including the stack and protocol costs as processor and network attributable costs, in addition to the sequence of transfer times costs (L^c) of

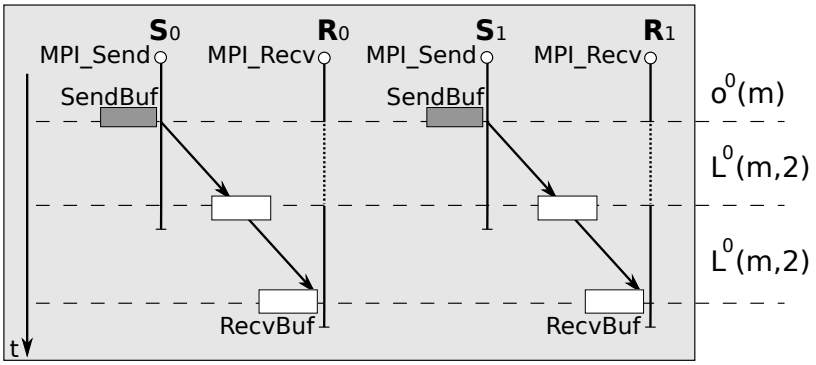


FIGURE 3.8: Cost of the concurrency of two transmissions in shared memory through an intermediate buffer expressed in terms of τ–Lop.

the message. Each one of the s transfers composing the transmission can progress in concurrency with others, a fact represented by τ. The transmission can be split in k segments, as it is usual for long messages in shared memory.

3.5 Concurrent transmissions

The $\parallel$ operator[2] is introduced to represent the contention of concurrent transfers that takes place mainly in MPI operations, both point-to-point and collective.

The expression $A \parallel L(m,1)$ represents the cost of A concurrent transfers. An expression that extends to $A \parallel L(m,\tau)$, representing the cost of A concurrent sets of in turn τ concurrent transfers. It is defined as $A \parallel L(m,\tau) = L(m, A \times \tau)$. Also, it is defined that $A \parallel (L_1(m,\tau) + L_2(m,\tau)) = A \parallel L_1(m,\tau) + A \parallel L_2(m,\tau)$, so that concurrency operator $\parallel$ possesses the distributive property with respect to the addition (sequence) of the costs of serial transfers.

As a message transmission is built as a sequence of transfers, the concurrency operator can be applied to characterize the cost of A concurrent message transmissions contending for the c communication channel, as $A \parallel T^c(m)$, defined as follows:

$$A \parallel T^c(m) = A \parallel \sum_{j=0}^{s-1} L_j^c(m,\tau_j) = \sum_{j=0}^{s-1} A \parallel L_j^c(m,\tau_j) = \sum_{j=0}^{s-1} L_j^c(m, A \times \tau_j) \tag{3.5}$$

Figure 3.8 illustrates the concept. It shows a pair of concurrent message transmissions ($A = 2$) through an intermediate buffer in shared memory. Each transmission, with the cost defined by (3.1), is composed of two transfers actually contending for the communication channel, leading to an increase in the cost.

[2]The operator is named *concurrency operator*.

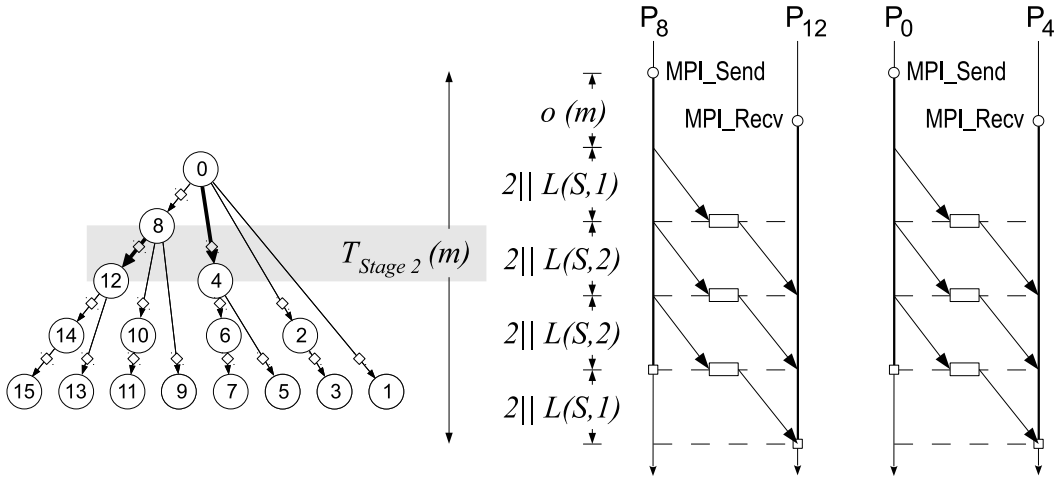


FIGURE 3.9: τ–Lop cost of stage 2 of a Binomial broadcast, expressed in terms of concurrent transfers.

Figure 3.8 scenery modeled with τ–Lop leads to a cost:

$$2 \parallel T^0(m) = 2 \parallel \left[o^0(m) + 2\, L^0(m,1) \right] = o^0(m) + 2\, L^0(m,2),$$

higher than the cost of a single point-to-point transmission, given by (3.1). Note that the overhead cost is attributable mainly to the processor, a non-shared resource, and hence it is considered not affected by $\parallel$.

The concurrency operator allows a simple and expressive representation of point-to-point transmissions in a collective operation. Figure 3.9 illustrates an example of a segmented message taking place at the second stage of a Binomial tree in shared memory. If the cost of the message transmission between processes 0 and 8 in Figure 3.9 is $T^0(m)$, the cost between processes 8 and 12 is higher because two concurrent transmissions share the memory bandwidth. In contrast, the costs are the same with $\log_n P$ and LogGP, which is unrealistic. The true cost between processes 8 and 12 ranges between $T^0(m)$, for a perfectly parallel channel, and $2 \times T^0(m)$ for a serial channel. The actual cost is represented in τ–Lop as:

$$\begin{aligned} 2 \parallel T^0(m) &= 2 \parallel \left[o^0(m) + 2\, L^0(S,1) + (k-1)\, L^0(S,2) \right] \\ &= o^0(m) + 2\, L^0(S,2) + (k-1)\, L^0(S,4) \end{aligned}$$

In the network channel the behavior is similar. Concurrent message transmissions share the communication channel bandwidth. The contention increases the cost of each individual transmission. Figure 3.10 shows the growth of the overall cost with the increase of the number of concurrent transmissions τ progressing through an Ethernet network

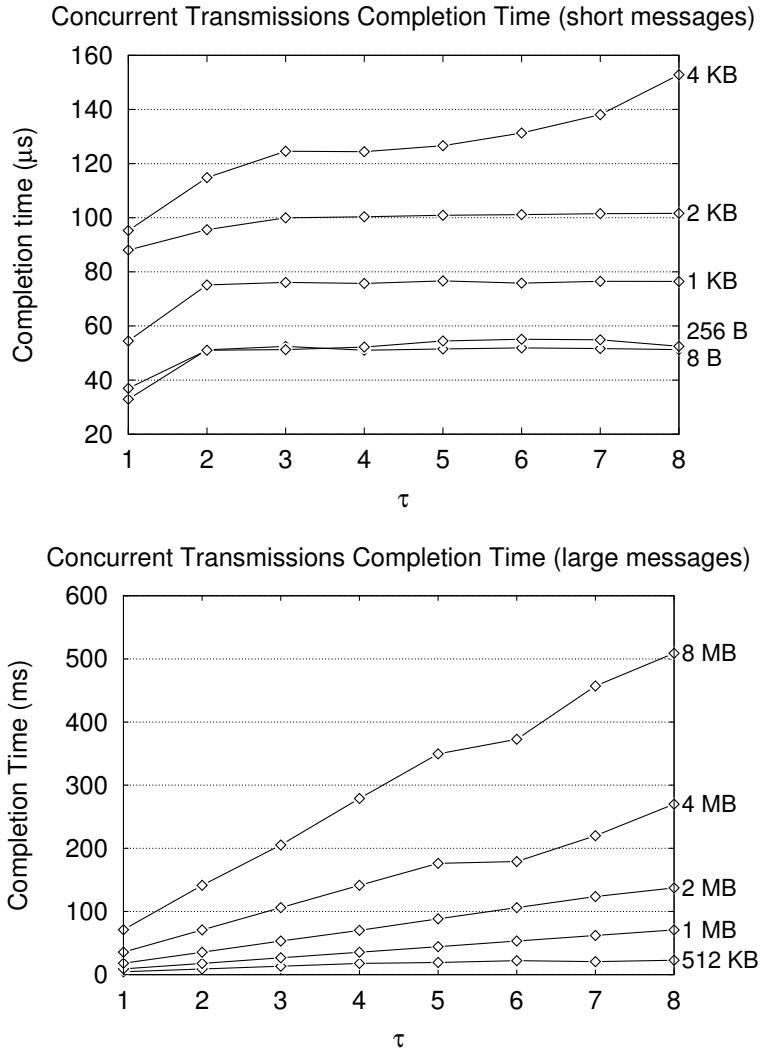


FIGURE 3.10: Cost of a transmission as a function of message size (m) and number of processes concurrently copying (τ) in an Ethernet network communication channel.

channel. The impact of concurrency on the cost is significant when the size of message is greater than $2KB$ in the *Metropolis* cluster, described in section 7.1.2.

Figure 3.11 shows the τ–Lop modeling of two concurrent point-to-point transmissions through the network communication channel. An analysis of contention follows. The two transfers in the node 0 between the source buffer and the NIC progress concurrently, hence with the cost $L^0(m, 2)$. Data sent through the network has the same destination node 1, therefore, contention is generated in the destination NIC, with the cost $L^1(m, 2)$. Finally, the data is transferred from the NIC to the destination buffers ($L^0(m, 2)$). The total cost is:

$$2 \parallel T^1(m) = o^1(m) + 2\,L^0(m, 2) + L^1(m, 2)$$

For the sake of simplicity, the analysis in this thesis makes two assumptions regarding

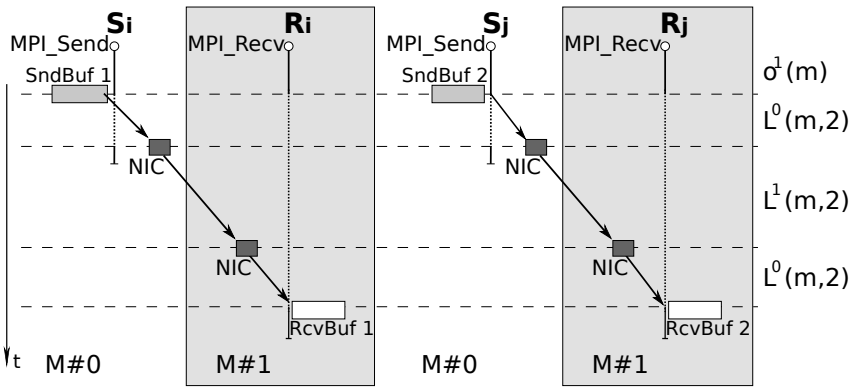


FIGURE 3.11: An example of two concurrent transmissions from node $M\#0$ to the node $M\#1$ expressed in terms of τ–Lop transfers.

network transfers. First, it does not consider the overlap of the transfer from a buffer to the local NIC with the transfer from the local NIC to the remote NIC, assuming, in common with the rest of the models, the error in the prediction. Such concurrency has a random behavior difficult to measure and highly dependent on the network technology. Second, the only network transfers considered as concurrent are those reaching the same destination node at the same time, hence contending in the destination NIC.

The impact of the concurrency on the cost of transfers from NIC to NIC is hardware dependent. For instance, it varies from Ethernet to Infiniband networks. While in the *Metropolis* Ethernet network the cost of two concurrent transmission of messages approaches to their sequential cost ($L(m,2) \approx 2\,L(m,1)$), specially for large messages (as can be seen in Figure 3.10), the effect of the contention on an Infiniband network could be lower, approaching to the pure parallel cost ($L(m,2) \approx L(m,1)$). Nevertheless, each individual platform characteristics have to be taken into account, and hence, the parameters of the τ–Lop model have to be estimated in each of such individual platforms. In any case, the flexibility of the model allows itself to adapt to any other system characteristics found in other high performance platforms. For an exhaustive taxonomy of contention effects in high performance networks, see Jerome et al. work [45].

3.6 Other point-to-point transmissions

Communication between processes others than a simple point-to-point message exists. For instance, the commonly used *Send-receive* operation named as *MPI_Sendrecv* in the MPI Standard. On it, a process P_i sends a message of size m to a process P_j and next P_i receives another message of size m from a process P_k. Usually, the Send-receive operation has a higher cost than a point-to-point simple transmission (T_{p2p}) defined in (3.1), because processes access the channel simultaneously, stressing its bandwidth.

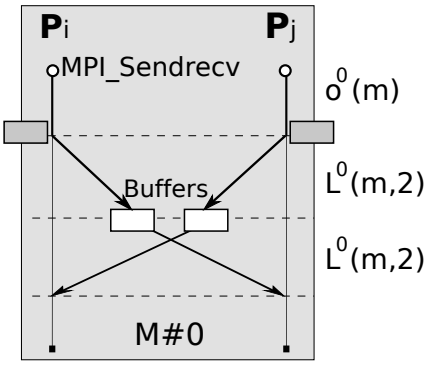


FIGURE 3.12: An *Exchange* point-to-point operation of a short message between two processes represented by τ–Lop.

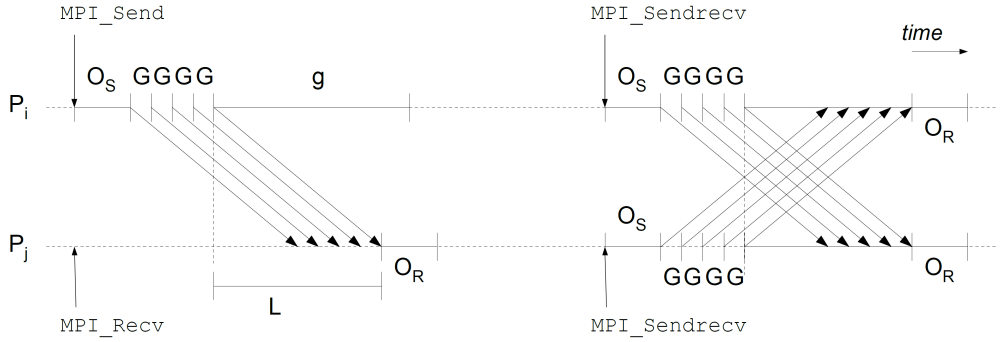


FIGURE 3.13: A point-to-point (left) and an Exchange (right) between two processes represented by LogGP. Interestingly enough the parameters of LogGP are not affected by processes operating concurrently. As a result LogGP equals the cost of *MPI_Send* and *MPI_Sendrecv*.

For P participants, the cost of the operation under τ–Lop in shared memory is:

$$T^0_{sr}(m) = P \parallel T^0_{p2p}(m) = P \parallel \left[o^0(m) + 2\,L^0(m,1)\right] = o^0(m) + 2\,L^0(m,P) \tag{3.6}$$

For $m > S$, the segmented formulation will be applied. Each of the P processes involved in a Send-receive operation makes two segmented transmissions: the send part requires a k segments transfer to the intermediate buffer (with cost $k\,L(S,1)$), and the receive part requires an additional k segments transfer from the intermediate buffer (again with cost $k\,L(S,1)$). The total cost is applied for P concurrent transmissions as:

$$T^0_{sr}(m) = P \parallel \left[o^0(m) + 2\,k\,L^0(S,1)\right] = o^0(m) + 2\,k\,L^0(S,P) \tag{3.7}$$

The Send-receive operation is usually called *Exchange* when only $P = 2$ processes are involved, with a cost of:

$$T^0_{exch}(m) = 2 \parallel \left[o^0(m) + 2\,L^0(m,1)\right] = o^0(m) + 2\,L^0(m,2) \tag{3.8}$$

and, in segmented transmissions:

$$T^0_{exch}(m) = 2 \parallel \left[o^0(m) + 2\,k\,L^0(S,1)\right] = o^0(m) + 2\,k\,L^0(S,2) \tag{3.9}$$

Figure 3.12 shows the decomposition in transfers of a message Exchange between two processes in shared memory. The figure shows the increasing in the cost compared to the Figure 3.3 representing a simple point-to-point operation.

The left side of the Figure 3.13 shows the cost of the Exchange operation represented under LogGP. Note that the total cost of the operation is not different from the point-to-point cost (represented at the right of the figure), because contention is not considered. The $\log_n P$ model makes the same error.

Regarding network transmissions, due to the processes run in different nodes, two point-to-point transmissions composing the Exchange arrive to different nodes. Therefore, $T^1_{exch}(m) = T^1_{p2p}(m)$ is assumed, although modeling could be adapted to the particular behavior of the channel in each particular platform. Section 5 discusses the Send-receive operation in different network scenarios.

Chapter 4

Modeling MPI collective algorithms

Formal analysis and modeling of algorithms used in the implementation of MPI collective operations is important because it allows their comparison and optimization, as well as an architecture independent cost prediction. Collective operations have a great influence in the overall performance of an MPI application. As new clusters are built upon a growing number of cores per node, efficient implementation of intra-node collective operations is becoming more relevant.

This chapter puts forward the modeling of several algorithms. For the sake of clarity we restrict ourselves to a scenery of shared memory. The algorithms are used to implement MPI collective operations in mainstream libraries as MPICH and Open MPI. A homogeneous platform is considered, with contention between all processes in the access to the communication channel. τ–Lop estimations are analytically compared to those developed for LogGP and $\log_n P$ models. The concept of concurrent transmissions, introduced in the previous chapter (section 3.5), is essential to model and represent the MPI collective algorithms with τ–Lop. Currently used models represent a collective as a sequence of mere point-to-point transmissions, which leads to an insufficient level of representation and hence to a higher error. τ–Lop uses the Send-receive and Exchange, in addition to the point-to-point transmissions, as the basis of several collective operations.

Indeed, this chapter evaluates the accuracy of the models cost estimation, comparing their predictions with the real values in the test platforms. The study is not intended to be an exhaustive analysis of the most known algorithms, but an incremental introduction to how modeling algorithms with increasing complexity using the τ–Lop model. Nevertheless, the algorithms modeled, used in MPICH and Open MPI COLL Tuned component, are representative, and they are also used as building blocks by other MPI

collective operations. For instance, the Recursive Doubling algorithm is used in the *MPI_Allgather* collective, but it is also used in the implementation of the *MPI_Bcast* for medium size messages and power of two number of processes in MPICH, preceded by a Binomial Scatter. For other message sizes, the Ring algorithm replaces Recursive Doubling, and it is used as well in the *MPI_Allgather* collective.

From another point of view, the algorithms studied are of further interest because they have contrasting features with respect to the size of the messages and the number of processes communicating in each stage. A common feature of these algorithms is that they are built as a point-to-point communication graph between processes and executed in a sequence of stages. Regarding the differences, for instance, in the Binomial tree Broadcast algorithm, the number of processes involved in transmissions grows in successive stages, while the message size remains constant. In the Binomial Scatter algorithm, the number of involved processes grows, while the message size halves. In the Recursive Doubling algorithm, the number of involved processes remains constant while the message size doubles. Finally, in the Ring algorithm both the number of communicating processes and the size of the interchanged messages remain constant.

The Open MPICOLL SM component implements a different kind of algorithms, targeted to shared memory and not based on point-to-point messages, but on shared buffers between processes involved in the operation. As an example, the analysis of the Broadcast and the reduction algorithms under τ–Lop is performed.

In the MPI collective operations, except the Broadcast and Barrier, each process provides two buffers, called the *input* and *output* buffers[1]. The input buffer contains the data contributed by the process, and the output buffer contains the product of the operation for the process.

4.1 The Binomial tree Broadcast algorithm

In the Broadcast collective operation (*MPI_Bcast*) a process called *root* sends a message of size m to the rest of the processes in the group. The Binomial tree is a representative algorithm used to implement the *MPI_Bcast*. In the first stage of this algorithm, the root sends the message to the process with rank root + P/2. The algorithm recursively continues with both processes acting as roots of their respective sub-trees with P/2 processes. The height of a Binomial tree of P processes is $h(P) = \lceil \log_2(P) \rceil$, and hence it requires $\lceil \log_2(P) \rceil$ stages.

[1] MPI standard allows to use the same buffer as the input and output, however, in this document they are considered logically separated, without consequences in the modeling.

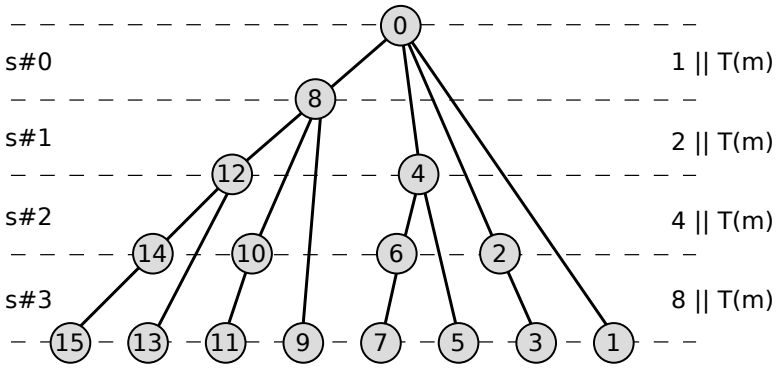


FIGURE 4.1: Binomial tree algorithm with $P = 16$ processes sharing a communication channel. The number of stages is the same as the heigh of the tree, and the number of concurrent transmissions doubles in each stage.

As shown in Figure 4.1, the cost of the first stage ($s\#0$) equals the cost of a point-to-point message transmission $T(m)$ between two processes. The cost of the subsequent stages is represented using the concurrency operator developed in section 3.5 to express the competing concurrent transmissions in each stage as $2 \parallel T(m)$, $4 \parallel T(m)$, $8 \parallel T(m)$ and so on. Hence, the cost of the whole Broadcast on a full tree is[2]:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel T(m) \right] \tag{4.1}$$

If $T(m)$ is given by (3.1), i.e. short messages, the Broadcast cost becomes:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel (o(m) + 2\,L(m,1)) \right] = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[o(m) + 2\,L(m,2^i) \right] \tag{4.2}$$

On the contrary, if $T(m)$ is given by (3.2), i.e. large segmented messages, then (4.1) becomes as follows:

$$\begin{aligned} \Theta_{Bin}(m) &= \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel \left(o(m) + 2\,L(S,1) + (k-1)\,L(S,2) \right) \right] \\ &= \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[o(m) + 2\,L(S,2^i) + (k-1)\,L(S,2^{i+1}) \right], \end{aligned} \tag{4.3}$$

that, for great values of k could be simplified to:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[o(m) + k\,L(S,2^{i+1}) \right] \tag{4.4}$$

[2]While T is commonly used in the literature to represent the cost of a point-to-point transmission involving two processes, we use Θ to represent the cost of a collective algorithm as a whole.

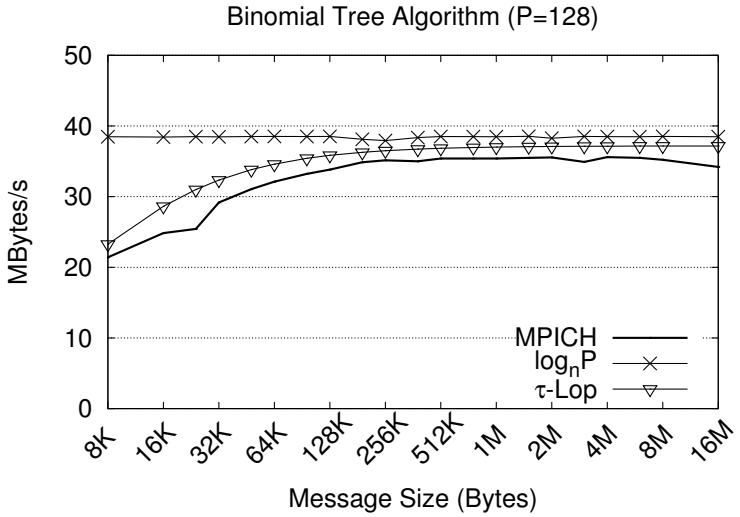


FIGURE 4.2: Binomial tree algorithm cost in terms of bandwidth by $\log_n$P and τ–Lop cost models. They compare to the real cost of the MPICH implementation for $P = 128$ processes and increasing message sizes in *Lusitania*. The LogGP estimated bandwidth is above 130MB/s, and it is not showed by clarity.

LogGP models the cost of a Binomial tree as the largest path from the root to a leaf, hence, as a sequence of $\lceil \log_2(P) \rceil$ point-to-point operations. For instance, for non segmented transmissions, the cost is:

$$T_{bin}^{LogGP} = \lceil \log_2(P) \rceil \times \left(2\,o + L + (m-1)\,G\right) \tag{4.5}$$

The cost of the Binomial tree Broadcast algorithm in $\log_n P$ leads to the same type of expression than that of LogGP:

$$T_{bcast}^{\log_n P} = \lceil \log_2(P) \rceil \times (2\,o) \tag{4.6}$$

As a result, in both (4.5) and (4.6), for instance, the cost for $P = 65$ and that for $P = 128$ in a shared memory system will be the same ($7 \times \left(2\,o + L + (m-1)\,G\right)$ and $7 \times (2\,o)$ respectively), which is far from being a correct prediction. τ–Lop, in contrast, models them as $2 \times \left[L(m,1) + L(m,2) + L(m,4) + \ldots + L(m,32) + L(m,1)\right]$ for $P = 65$, but $2 \times \left[L(m,1) + L(m,2) + L(m,4) + \ldots + L(m,32) + L(m,64)\right]$ for $P = 128$, which includes the concurrency in all the stages of the algorithm.

Figure 4.2 plots the costs estimated by the models compared to the real value measured using MPICH in the shared memory 128-core *Lusitania* machine, described in the section 7.1.1. The bandwidth is used, instead of the execution time, for reasons of clarity in the plot, and it is calculated as m/t, being m is the size of the message and being t the cost

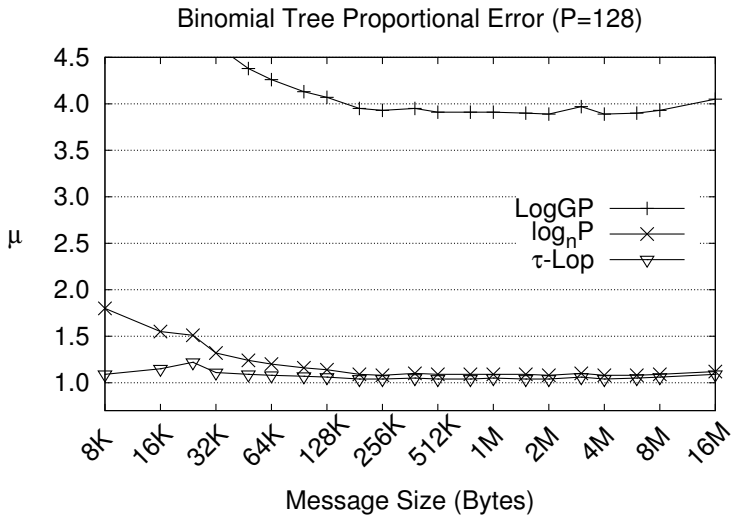


FIGURE 4.3: Binomial tree algorithm proportional error μ made by LogGP, $\log_n P$ and τ–Lop cost models with respect to the real cost of the MPICH implementation for $P = 128$ processes and growing message sizes in *Lusitania*.

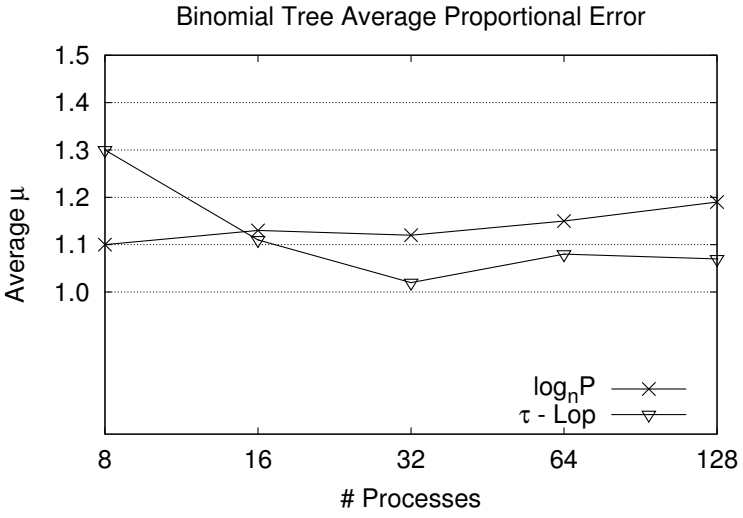


FIGURE 4.4: Binomial tree algorithm average proportional error compared to MPICH implementation for increasing number of processes P, in *Lusitania* machine. The message size used in each P ranges between 8 KB and 16 MB. LogGP average error is higher than $4x$, and it is omitted by clarity.

time returned by the models. The parameters of the models are measured as explained in chapter 7.

In its parameter assessment, $\log_n P$ assigns half of a MPICH point-to-point cost to the overhead o parameter (see (4.6)). The cost of the point-to-point message transmission of size m is the unity of measurement in $\log_n P$. This fact, together with the low contention of a Binomial tree, leads to the good prediction exhibited by $\log_n P$.

Regarding LogGP, measuring its parameters in shared memory is difficult. A minimal

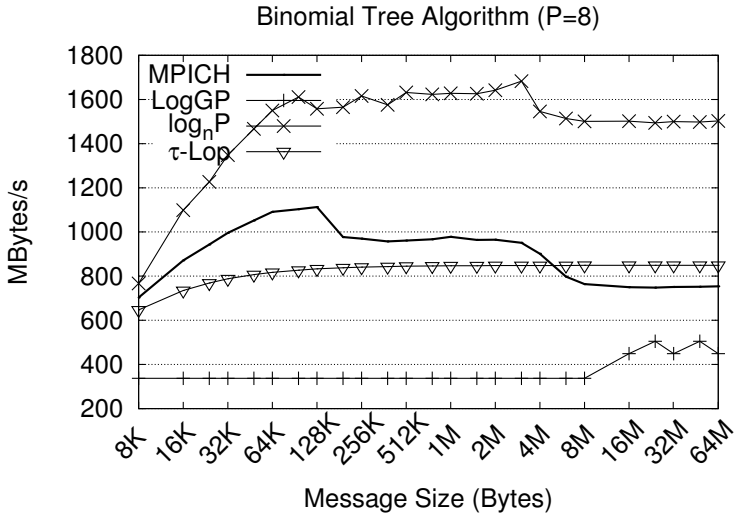


FIGURE 4.5: Binomial tree algorithm estimation made by cost models with respect to the real cost of the MPICH implementation for eight processes and growing message size in *Metropolis*.

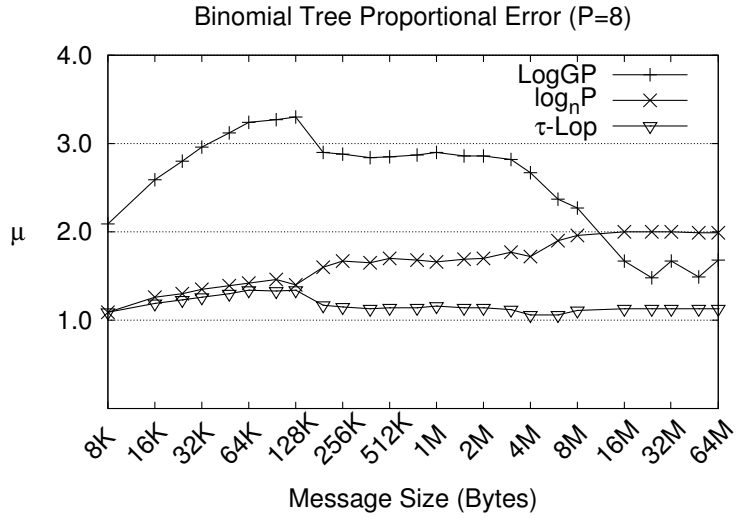


FIGURE 4.6: Binomial tree algorithm proportional error μ made by LogGP, $\log_n P$ and τ–Lop cost models together with the real cost of the MPICH implementation for eight processes and growing message size in a node of *Metropolis*.

variation, mainly in G, leads to a great prediction error, because the parameter value is quite low (see section 7.2) and it is multiplied by the message size in bytes, as shown in definition (4.5). The tool used for the measurement ([32]) leads to more than $4x$ proportional error for any P, as shown in Figure 4.3. Although it might be understood to mean a parameter estimation error that leads to a high bandwidth, the obtained parameter values are used in the cost prediction of other collective operations in this thesis, with the opposite results. $\log_n P$ and τ–Lop achieve a similar accuracy in the cost estimations, due to fact that the concurrency effect in the Binomial tree algorithm on the *Lusitania* shared memory ccNUMA architecture is not high.

Figure 4.5 shows the cost estimations of the models with respect to the real measurements in a node of the *Metropolis* platform with $P = 8$ processes. Each core in the node has a L2 cache of 256 KB and a L3 shared cache of 12 MB, figures that leave their footprint in the *MPI_Bcast* execution time despite the intent of the *-off_cache* IMB option to avoid the cache effect. Such cache effect increases the τ–Lop error in the range of messages which fits in the L2 and L3 caches, as can be seen in Figure 4.6. The figure plots the proportional error made by LogGP, $\log_n P$ and τ–Lop with respect to the measured cost of the Binomial tree broadcast algorithm (4.3). The $\log_n P$ error ends up growing with m, because concurrency effect increases for large messages, while the cache influence decreases, leading to more accurate predictions by τ–Lop. LogGP shows an expected behavior with a very low predicted execution time (hence, the showed bandwidth is high) because it does not consider the contention cost. Indeed, as argued before, a little error in the G parameter raises the error in the final prediction, although the G estimation by the measurement tool ([32]) in *Metropolis* slightly improves the average proportional error with respect to that in *Lusitania*. Average proportional error for the range of message sizes for LogGP, $\log_n P$ and τ–Lop are 2.36, 1.62 and 1.20 respectively.

4.2 Recursive Doubling

The *Recursive Doubling* algorithm is used in several collective operations, as Broadcast and Allgather. It is based on the Send-receive operation instead of on the simpler point-to-point transmission. In the Recursive Doubling algorithm for the Allgather collective operation (*RDA*), each involved process contributes with a message of size m and receives from the rest of processes $P - 1$ messages, ordered in the reception buffer by rank, for a total of $P \times m$ bytes. The RDA algorithm is executed in $\log_2(P)$ stages when P is a power of two. In each stage i, rank p exchanges $2^i\, m$ bytes with rank $p \oplus 2^i$. An initial local copy of the m bytes message takes place in each process, from the input to the output buffer, with offset $p \times m$ bytes. The number of processes communicating in each stage remains constant in this algorithm, whereas the message size doubles.

Figure 4.7 shows the Recursive Doubling algorithm communication pattern for $P = 16$ processes. For clarity, Figure 4.8 details per stage the communication cost of RDA with $P = 4$ processes. τ–Lop models the algorithm cost using the Send-receive operation between each two processes, and hence, P processes are executing the Send-receive at a time[3]. The cost becomes:

$$\Theta_{RDA}(m) = c(m,P) + \sum_{i=0}^{\log(P)-1} \left[P \parallel T_{sr}\left(2^i\, m\right) \right] \tag{4.7}$$

[3]The RDA cost can also be modeled using the Exchange operation, as $\frac{P}{2} \parallel T_{exch}$.

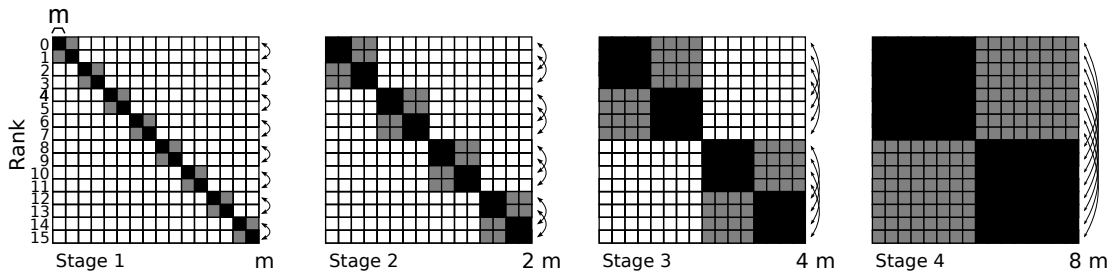


FIGURE 4.7: Recursive Doubling Allgather algorithm for $P = 16$ processes. A previous stage locally copies initial data into the *diagonal* of the buffer. An arc is a Send-receive message transmission between two processes. The size of messages doubles in each stage and all P processes communicate in each stage.

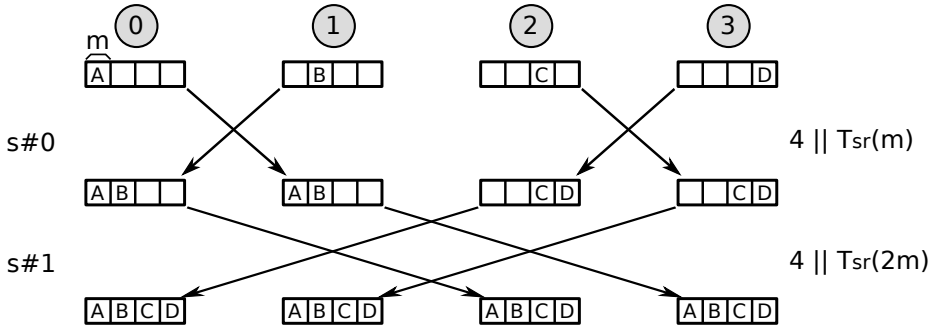


FIGURE 4.8: Per stage communication cost of the Recursive Doubling Allgather algorithm for $P = 4$ processes. Arrows indicate a Send-receive message transmission between the output buffers of each two processes.

After the initial local copy represented by $c(m, P)$, there are P stages of concurrent Send-receive transmissions (T_{sr}) of increasing message size ($2^i\, m$). If short messages are considered as in (3.6), the cost expands to:

$$\begin{aligned}
\Theta_{RDA}(m) &= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[P \parallel \Big(o(m) + 2\, L(2^i\, m, 1) \Big) \right] \\
&= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[o(m) + 2^{i+1}\, L(m,P) \right] \\
&= c(m,P) + \log P\, o(m) + (P-1)\, 2\, L(m,P)
\end{aligned} \tag{4.8}$$

Note that definition (4.8) has been simplified using the *linearity principle*, defined in section 3.3.

The operation expands to segmented messages (see (3.7)) as:

$$\begin{aligned}\Theta_{RDA}(m) &= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[P \parallel \left(o(m) + 2^{i+1}\, k\, L(S,1) \right) \right] \\ &= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[o(m) + 2^{i+1}\, k\, L(S,P) \right] \\ &= c(m,P) + \log P\, o(m) + (P-1)\, 2\, k\, L(S,P)\end{aligned} \tag{4.9}$$

As the LogGP cost of RDA is the mere addition of the costs of all the stages ([13]), we have that:

$$\begin{aligned}T_{RDA}^{LogGP} &= \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} T_{p2p}\left(2^i\, m\right) \\ &= \log_2 P \cdot (L + 2\,o - G) + (P-1)\; m\, G\end{aligned} \tag{4.10}$$

Estimation for segmented messages (defined in 2.2) is:

$$T_{RDA}^{LogGP} = \delta\, m + \log P\, \left(L + 2\,o + (S-1)\, G\right) + (P-1)\,(k-1)\left(g + (S-1)\, G\right) \tag{4.11}$$

In LogGP, the δ parameter is usually used for representing the cost of a local copy of a byte. The former cost can be divided in two parts, named the fixed and variable part respectively. Fixed part represents the initial latency cost and depends on the logarithm in base 2 of the number of processes. Variable part depends on the message size (number of segments k and the size of a segment S), and represents the bandwidth. Variable cost approximates to $P\, k\, (g + S\, G)$, the cost of transmitting k messages of size $P\, S$, which means that the cost represents just the cost of a single process, ignoring the contention.

For $\log_n P$, original notation in [18] is used, taking into account that the overhead cost o refers to a message of size m. In addition, we introduce a new term o_m to represent the cost of the initial local copy in RDA, usually with a value $o_m \leq o$. The cost for the algorithms is:

$$T_{RDA}^{\log_n P} = o_m + \sum_{i=0}^{\log(P)-1} \left(2^i\, 2\, o\right) = o_m + (P-1)\, 2\, o, \tag{4.12}$$

similar to that of LogGP cost, which just considers the cost of the longest path.

The contention in the RDA algorithm, in contrast to that in the Binomial tree, is high in shared memory. Note that in each stage all P processes access to the communication channel concurrently. Figure 4.9 shows the estimated bandwidth of LogGP, $\log_n P$ and τ–Lop models in *Lusitania* for $P = 128$ processes. The lack of contention modeling makes $\log_n P$ prediction to distance from the real MPICH measured value. It leads to

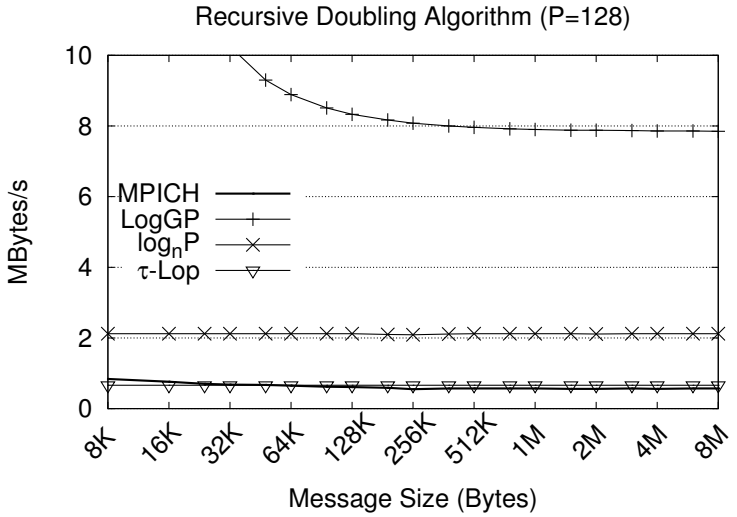


FIGURE 4.9: Recursive Doubling algorithm cost estimation by LogGP, $\log_n$P and τ–Lop cost models, as well as the real cost of the MPICH implementation for $P = 128$ processes and increasing message sizes in *Lusitania*.

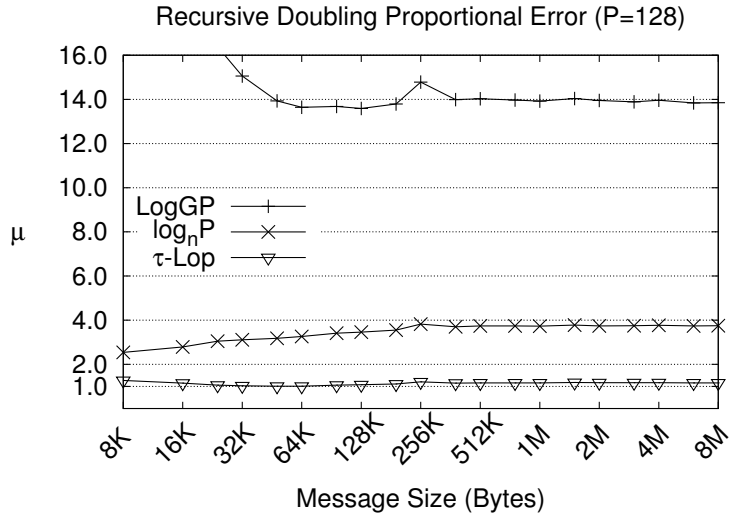


FIGURE 4.10: Recursive Doubling proportional error made by LogGP, $\log_n P$ and τ–Lop models with respect to the real cost of the MPICH implementation for $P = 128$ processes and growing message sizes in *Lusitania*.

a mean proportional error of 3.48, showed for a range of messages in the Figure 4.10, where the contention is more appreciable for increasing message sizes.

Figure 4.11 shows the proportional error made by τ–Lop (4.7), LogGP (4.11) and $\log_n P$ (4.12) on the RDA algorithm for increasing P. Above discussion of the Binomial tree revealed that LogGP is incapable of predicting shared memory performance. Its error on RDA, however, is not as high as expected, although much higher than that of τ–Lop. Again τ–Lop presents error figures that remain constant as P increases, showing hence a good scalability in shared memory. As LogGP and $\log_n P$ do not model the contention

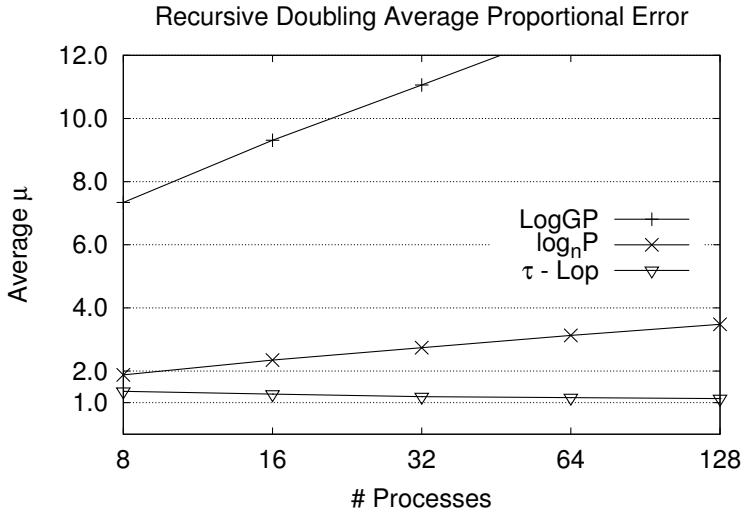


FIGURE 4.11: Average proportional error of the three performance models of Recursive Doubling Allgather algorithm, with respect to the real cost of its MPICH implementation in *Lusitania* machine. The message size m ranges between 8 KB and 8 MB on an increasing number of processes P involved in the operation.

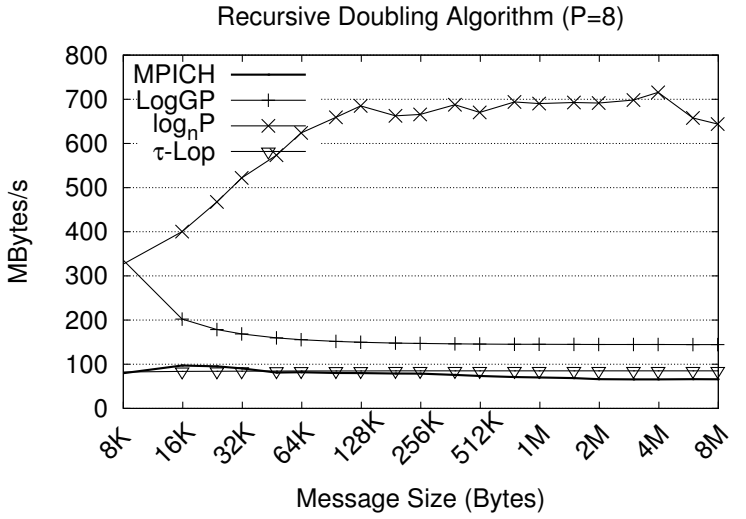


FIGURE 4.12: Recursive Doubling algorithm cost estimation by LogGP, $\log_n$P and τ–Lop models together with the real cost of the MPICH implementation for $P = 8$ processes and increasing message sizes in the *Metropolis* multi-core platform.

derived cost, their cost estimation is very optimistic for all the range of process number. Furthermore, the most important point is their growth with P of the proportional error, which confirms the importance of the contention modeling in current HPC platforms. Figure 4.12 and Figure 4.13 confirm this fact in each node of the multi-core *Metropolis* platform, with only $P = 8$ processes.

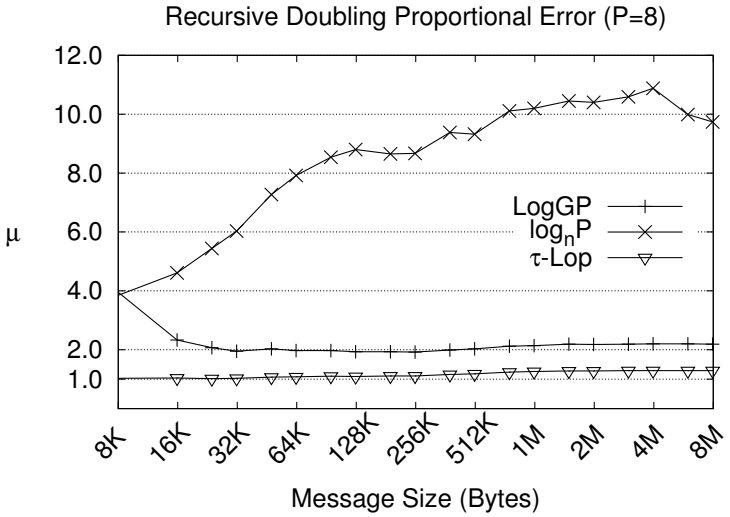


FIGURE 4.13: Recursive Doubling proportional error made by LogGP, $\log_n P$ and τ–Lop models with respect to the real cost of the MPICH implementation for $P = 8$ processes and growing message sizes in *Metropolis*.

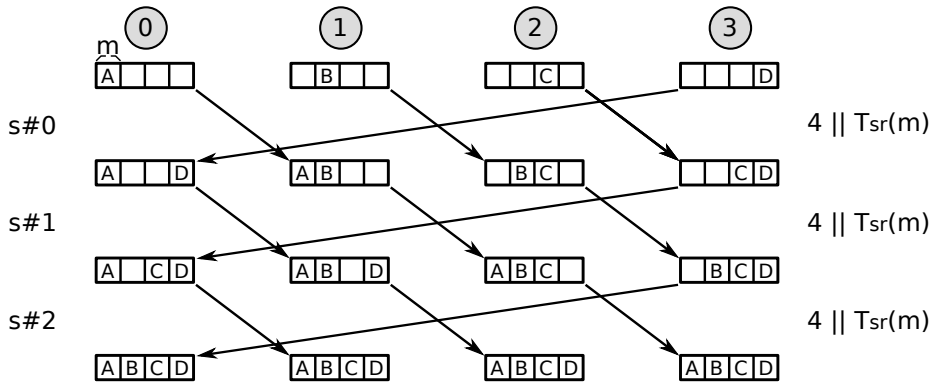


FIGURE 4.14: An Allgather *Ring* operation for $P = 4$ processes. Arrows indicate Send-receive operations for each process.

4.3 Ring

The *Ring* algorithm is used in the *MPI_Allgather* collective operation. It is also used in others, for instance, preceded by an *Scatter* in the *MPI_Bcast* of MPICH, for medium and large size of messages and non-power-of-two number of processes.

The Ring algorithm for the Allgather collective operation is executed in $P-1$ stages. Each stage executes P simultaneous Send-receive operations (defined in 3.6) of a message of size m in a ring of P processes. Its behavior is represented in Figure 4.14, in which, after an initial local copy to the receive buffer with offset $p \times m$, a process p concurrently sends a message to the process $p+1$ and receives from the $p-1$, with wraparound. The

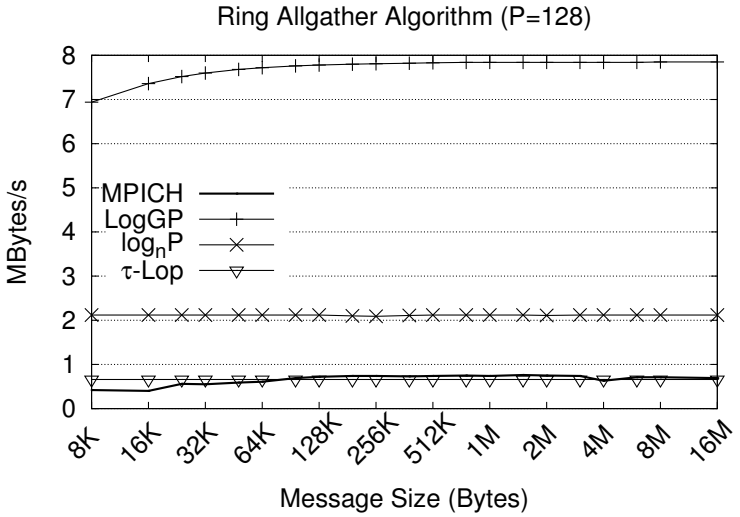


FIGURE 4.15: Ring algorithm cost estimation by LogGP, $\log_n$P and τ–Lop models together with the real cost of the MPICH implementation for $P = 128$ processes and increasing message sizes in the *Lusitania* machine.

cost is modeled in τ–Lop as:

$$\Theta_{Ring}(m) = c(m,P) + (P-1) \times \left[P \parallel T_{sr}(m)\right], \tag{4.13}$$

which is immediately expanded to short messages as:

$$\Theta_{Ring}(m) = c(m,P) + (P-1)\, o(m) + (P-1)\, 2\, L(m,P), \tag{4.14}$$

and to segmented messages (see definition (3.7)) as:

$$\Theta_{Ring}(m) = c(m,P) + (P-1)\, o(m) + 2\, k\, (P-1)\, L(S,P) \tag{4.15}$$

LogGP models the Ring algorithm as the occurrence of $P-1$ transmissions of a message of size m, as:

$$T_{Ring}^{LogGP}(m) = \delta\, m + (P-1) \times \big(L + 2\, o + (S-1)\, G\big) + (P-1)(k-1)\, \big(g + (S-1)\, G\big), \tag{4.16}$$

while $\log_n P$ uses the same type of expression:

$$T_{Ring}^{\log_n P}(m) = o_m + (P-1)\, 2\, o \tag{4.17}$$

Note that o_m represents the cost of a local copy as a local memory transfer.

Although accuracy is important in a cost prediction model, its expressive power is even more important. It is to be noted how a complex algorithm, such as the Ring, is modeled

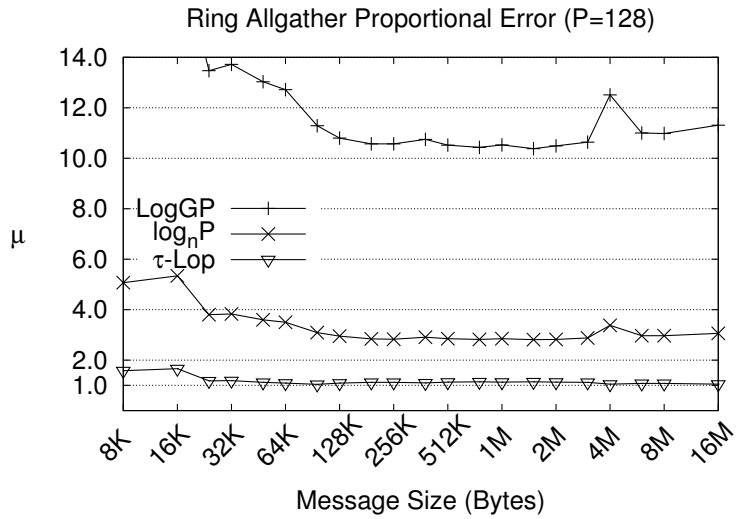


FIGURE 4.16: Ring proportional error made by LogGP, $\log_n P$ and τ–Lop models with respect to the real cost of the MPICH implementation for $P = 128$ processes and growing message sizes in *Lusitania*.

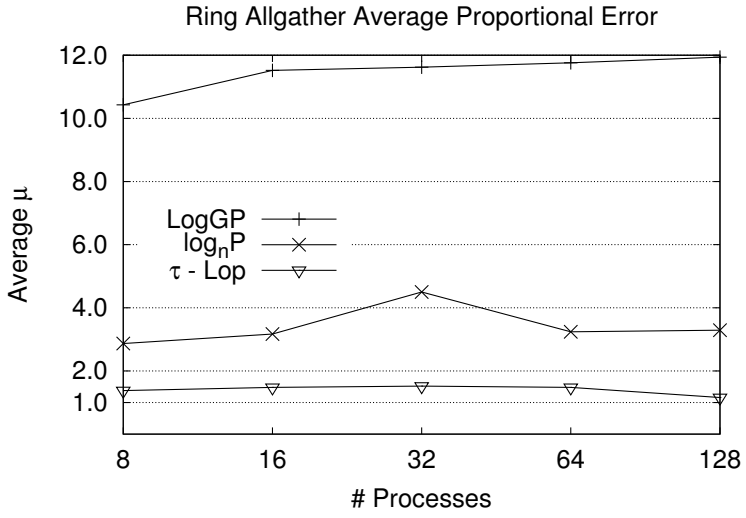


FIGURE 4.17: Average proportional error of the three performance models of the Ring Allgather algorithm, with respect to the real cost of its MPICH implementation in the *Lusitania* machine. The message size m ranges between 8 KB and 16 MB on an increasing number of processes P involved in the operation.

by τ–Lop in (4.13) in a compact and expressive way. LogGP and $\log_n P$ produce less meaningful formulations, because they simply multiply definitions (2.2) and (2.5) by $(P - 1)$ stages.

Figure 4.15 shows the estimated bandwidth in *Lusitania* with respect to the real MPICH value for a range of messages and $P = 128$ processes. As can be seen, LogGP fails again to explain the experimental results of the Ring Allgather algorithm. In spite of the simplicity of expression (4.17), the output of $\log_n P$ in Ring is too optimistic, leading to a bandwidth higher than real, again mainly because of the lack of the contention

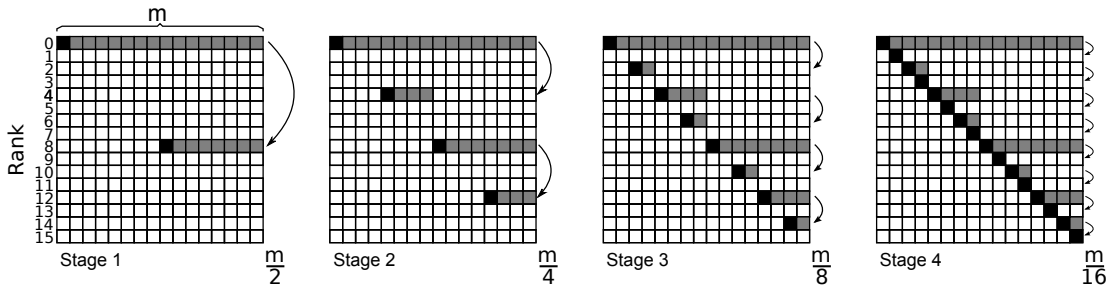


FIGURE 4.18: Scatter algorithm as a Binomial tree for $P = 16$ processes and $\lceil \log_2(P) \rceil = 4$ stages. An arc is a single point-to-point message transmission T. The root process has rank 0. The size of messages halves in each stage, while the number of communicating processes doubles.

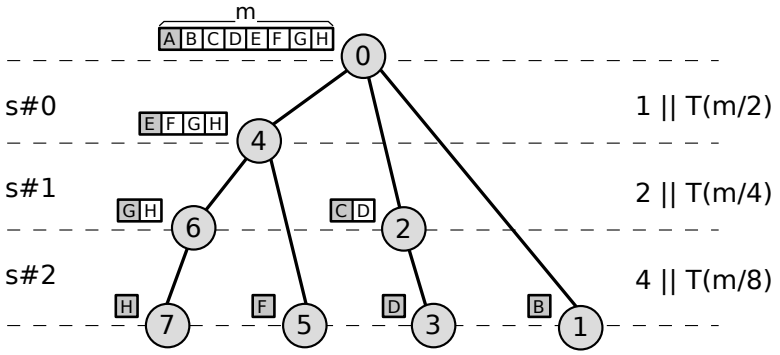


FIGURE 4.19: Detail of the per stage communication cost of the *Binomial Scatter* algorithm with $P = 8$ processes and process 0 as root. The root scatters data $[A - H]$ among the rest of the processes. The first item A ends up in the process of rank 0. The second item B ends up in the process of rank 1, and so on.

modeling. This fact is confirmed in the Figure 4.16. Mean proportional error for $P = 128$ for LogGP, $\log_n P$ and τ–Lop are $11.94x$, $3.29x$ and $1.16x$ respectively. These figures are the rule for the whole range of number of processes P, as Figure 4.17 shows.

4.4 Binomial Scatter

The algorithm implementing the *MPI_Scatter* collective scatters a message of size m across P processes. Figure 4.18 shows the communication pattern of the algorithm for $P = 16$ processes, and Figure 4.19 details the per stage communication cost of the algorithm for $P = 8$ processes and process 0 acting as root. The buffer of the root is divided in P fragments of size m/P, and the process with rank p expects the fragment in the position p. The algorithm uses a Binomial tree in such a way that the size of the sent messages halves at each stage, and each receiver process receives all the data its

children expect. τ–Lop allocates to the operation the following cost:

$$\Theta_{BinSct}(m) = \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[2^i \parallel T_{p2p}\left(\frac{m}{2^{i+1}}\right) \right] \tag{4.18}$$

Considering non-segmented messages, expression (4.18) expands to:

$$\begin{aligned} \Theta_{BinSct}(m) &= \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[2^i \parallel \left(o(m) + 2\,L\left(\frac{m}{2^{i+1}}, 1\right)\right)\right] \\ &= \log P\, o(m) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[2\,L\left(\frac{m}{2^{i+1}}, 2^i\right)\right] \end{aligned} \tag{4.19}$$

While segmented messages lead to:

$$\begin{aligned} \Theta_{BinSct}(m) &= \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[2^i \parallel \left(o(m) + k\,L\left(\frac{S}{2^{i+1}}, 2\right)\right)\right] \\ &= \log P\, o(m) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[\frac{k}{2^{i+1}}\,L\left(S, 2^{i+1}\right)\right] \end{aligned} \tag{4.20}$$

The difference in the cost between non-segmented and segmented messages comes in two points. First, non-segmented messages require to fully send a message of size m before the reception. Second, the segmented technique splits the interchanged messages but increases the concurrency. For instance, expressions (4.19) and (4.20) for the particular value of $P = 8$ processes, excluding the overhead, become:

$$\Theta_{BinSct}(m) = 2\,L\left(\frac{m}{2}, 1\right) + 2\,L\left(\frac{m}{4}, 2\right) + 2\,L\left(\frac{m}{8}, 4\right) = L(m,1) + L\left(\frac{m}{2}, 2\right) + L\left(\frac{m}{4}, 4\right) \tag{4.21}$$

$$\Theta_{BinSct}(m) = \frac{k}{2}\,L(S,2) + \frac{k}{4}\,L(S,4) + \frac{k}{8}\,L(S,8) \tag{4.22}$$

Note that comparing term to term of (4.21) and (4.22), though the cost of the segmented technique doubles the contention, it halves the message size. These subtle but performance critical issues have driven the design of shared memory communication in current MPI implementations. τ–Lop is able to retain and describe them in a simple enough way.

LogGP prediction is again the addition of the cost of each one of the stages of the Binomial tree in the longest path from the root to a leaf. For instance, considering

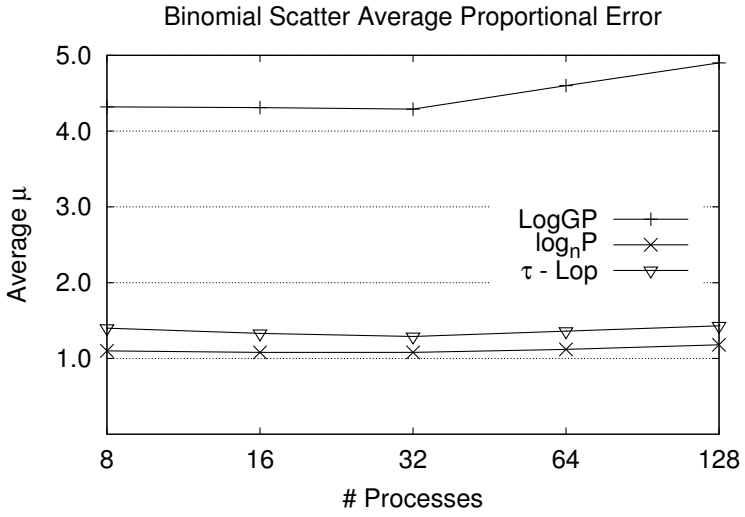


FIGURE 4.20: Average proportional error of three performance models of Binomial Scatter algorithm, with respect to the real cost of its MPICH implementation in *Lusitania* machine. The message size m ranges between 8 KB and 16 MB on an increasing number of processes P involved in the operation.

segmented messages, the cost is:

$$T_{BinSct}^{LogGP} = \log P \left(L + 2\,o + (S-1)\;G\right) + \frac{P-1}{P}\;(k-1)\;\left(g + (S-1)\;G\right) \tag{4.23}$$

The fixed part (latency) of (4.23) depends on the logarithm on base 2 of the number of processes. The variable part (bandwidth) depends on the size of the message, with k the number of segments.

$\log_n P$ cost is similar to that of LogGP, in the sense that it considers the longest path. Note once more that the referred messages size is hidden in the overhead parameter, as done in the original paper:

$$T_{BinSct}^{\log_n P} = \sum_{i=0}^{\lceil \log(P) \rceil - 1} 2\,o\left(\frac{m}{2^{i+1}}\right) = \frac{P-1}{P}\,2\,o \tag{4.24}$$

When the number of processes is high, expression (4.24) derives in the cost of a point-to-point message of size m. $T_{BinSct}^{\log_n P} = 2\,o$ is far from being a correct representation of the cost of a collective operation, despite of the fact that the total amount of bytes sent by the root process is close to m. LogGP has the same behavior but including an additional cost per stage related to the latency of the point-to-point operations. In contrast, on τ–Lop each stage explicitly shows its own contention and overhead, which raises accuracy and expressiveness.

Figure 4.20 shows the average proportional error of the Binomial Scatter algorithm for increasing number of processes in the *Lusitania* machine. LogGP error is high, but remains constant for the range of processes. $\log_n P$ gives a very low error, owing to two factors: the first is the low contention of the algorithm, even lower than Binomial broadcast, and the second, as explained in section 4.1, is the fact that $\log_n P$ measures the overhead parameter value directly from the point-to-point MPICH cost, and later uses it to predict the Scatter cost. While, τ–Lop offers a slightly higher cost.

4.5 Other collective algorithms

We provide next a survey of the cost modeling of several well-known algorithms which are commonly used in the implementation of MPI collective operations. Some of them have been proposed to improve the performance of collective operations in specific architectures, or under specific conditions such as the number of involved processes, and might not be found in mainstream MPICH and Open MPI libraries. Notwithstanding these algorithms are visited to provide further insights into the modeling capabilities of τ–Lop.

4.5.1 Neighbor Exchange

The *Neighbor Exchange* algorithm proposed in [65] for the *MPI_Allgather* collective operation was designed to exploit the piggy-backing feature of the TCP/IP protocols. The algorithm operates in $P/2$ stages with an even number of processes, and data is interchanged between nearest neighbor rank numbers. Initially, each process p copies m bytes from the input to the output buffer with offset $p \times m$. In the first stage, a process with rank p interchanges a message of size m with the rank $p+1$. In each of the rest of the stages, an even p rank interchanges $2\,m$ bytes with the $p-1$ and $p+1$ ranks alternately. The algorithm is similar to the Ring Allgather, except that it executes in half of the stages and doubles the message size per stage. The cost is:

$$\Theta_{Neigh}(m) = c(m,P) + \left[P \parallel T_{sr}(m)\right] + \left(\frac{P}{2} - 1\right)\left[P \parallel T_{sr}(2\,m)\right] \tag{4.25}$$

Considering the definition of T_{sr} operation in (3.6), the expression (4.25) expands to:

$$\begin{aligned}\Theta_{Neigh}(m) &= c(m,P) + P \parallel \big(o(m) + 2\,L(m,1)\big) + \left(\frac{P}{2} - 1\right)\Big(P \parallel \big(o(2\,m) + 2\,L(2\,m,1)\big)\Big) \\ &= c(m,P) + o(m) + 2\,L(m,P) + \left(\frac{P}{2} - 1\right)\big(o(2\,m) + 2\,L(2\,m,P)\big) \\ &= c(m,P) + \frac{P}{2}\,o(2\,m) + (P-1)\;2\,L(m,P)\end{aligned} \tag{4.26}$$

Note that, for clarity, we simplify $o(m) + \left(\frac{P}{2} - 1\right)\,o(2\,m) = \frac{P}{2}\,o(2\,m)$, with a minimum impact in the cost. If segmented messages are considered for T_{sr} in (3.7), (4.25) becomes:

$$\begin{aligned}\Theta_{Neigh}(m) &= c(m,P) + P \parallel \big(o(m) + 2\,k\,L(S,1)\big) + \left(\frac{P}{2} - 1\right) P \parallel \big(o(2\,m) + 2\,k\,L(2\,S,1)\big) \\ &= c(m,P) + o(m) + 2\,k\,L(S,P) + \left(\frac{P}{2} - 1\right)\big(o(2\,m) + 2\,k\,L(2\,m,P)\big) \\ &= c(m,P) + \frac{P}{2}\,o(2\,m) + (P-1)\;2\,k\,L(S,P)\end{aligned} \tag{4.27}$$

LogGP formulation for segmented messages is:

$$\begin{aligned}T_{Neigh}^{LogGP}(m) &= \delta\,m + T_{p2p}^{LogGP}(m) + \left(\frac{P}{2} - 1\right)\,T_{p2p}^{LogGP}(2\,m) \\ &= \delta\,m + \frac{P}{2}\,\big(L + 2\,o + (S-1)\,G\big) + (P-1)\,(k-1)\,\big(g + (S-1)\,G\big)\end{aligned} \tag{4.28}$$

Finally, $\log_n P$ cost is:

$$\begin{aligned}T_{Neigh}^{\log_n P}(m) &= o_m + T_{exch}^{\log_n P}(m) + \left(\frac{P}{2} - 1\right)\,T_{exch}^{\log_n P}(2\,m) \\ &= o_m + 2\,o + \left(\frac{P}{2} - 1\right)\,2\,2\,o = o_m + (P-1)\,2\,o\end{aligned} \tag{4.29}$$

LogGP cost includes the local copy and divides up the equation (4.28) in a latency related part, dependent on the number of stages ($P/2$), and the bandwidth related part dependent on the size of the messages sent. $\log_n P$ fails in providing expressivity in (4.29), reducing it to $2\,P$ transfers. τ–Lop takes into account, again, the initial copy, and the fixed and variable parts of the cost, including the fact that all P processes stress the communication channel at a time in each stage.

4.5.2 Dissemination

Dissemination algorithm for the *MPI_Allgather* is proposed in [8]. It executes in $\log_2 P$ stages, similar to the Recursive Doubling algorithm discussed in section (4.2), but with less stages if the number of processes is not a power of two. In the stage i, a process p sends $2^i\,m$ bytes to the process $(p+2^i)\%P$, and receives the same amount of data from $(p-2^i)\%P$. As the data sent and received could be non-contiguous, usually a local temporal buffer is used, and local copies are needed between the local and the output buffer. In an stage i, a total of 2^i-1 processes need to make a local copy of $2^i\,m$ bytes, leading τ–Lop to estimate a cost of:

$$\Theta_{Dissm}(m) = c(m,P) + \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[P \parallel T_{sr}(2^i\,m) + c\left(2^i\,m, 2^i-1\right) \right] \tag{4.30}$$

Again, expression (4.30) is extended for short and segmented messages. Considering short messages in (3.1), the cost is:

$$\begin{aligned}
\Theta_{Dissm}(m) &= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[o(2\,m) + 2\,L(2^i\,m,P) + c\left(2^i\,m, 2^i-1\right) \right] \\
&= c(m,P) + \log P\, o(m) + \sum_{i=0}^{\log(P)-1} \left[2^{i+1}\,L(m,P) \right] + \sum_{i=0}^{\log(P)-1} \left[2^i\,c\left(m, 2^i-1\right) \right] \\
&= c(m,P) + \log P\, o(m) + (P-1)\,2\,L(m,P) + \sum_{i=0}^{\log(P)-1} \left[2^i\,c\left(m, 2^i-1\right) \right],
\end{aligned} \tag{4.31}$$

while segmented messages in (3.2) lead to:

$$\begin{aligned}
\Theta_{Dissm}(m) &= c(m,P) + \sum_{i=0}^{\log(P)-1} \left[o(2^1\,m) + 2\,k, L(2^i\,S,P) + c\left(2^i\,m, 2^i-1\right) \right] \\
&= c(m,P) + \log P\, o(m) + \sum_{i=0}^{\log(P)-1} \left[2^{i+1}\,k\,L(S,P) \right] + \sum_{i=0}^{\log(P)-1} \left[2^i\,c\left(m, 2^i-1\right) \right] \\
&= c(m,P) + \log P\, o(m) + (P-1)\,2\,k\,L(S,P) + \sum_{i=0}^{\log(P)-1} \left[2^i\,c\left(m, 2^i-1\right) \right]
\end{aligned} \tag{4.32}$$

LogGP cost is modeled as:

$$\begin{aligned} T_{Dissm}^{LogGP}(m) &= \delta\, m + \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i\, \delta\, m + T_{p2p}^{LogGP} \right] \\ &= \delta\, m + \sum_{i=0}^{\log_2(P)-1} \left[2^i\, \delta\, m \right] + \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[T_{p2p}^{LogGP} \right] \\ &= P\, \delta\, m + \log P\, \left(L + 2\, o + (S-1)\, G \right) + (P-1)\, (k-1)\, \left(g + (S-1)\, G \right) \end{aligned} \tag{4.33}$$

Finally, $\log_n P$ cost is modeled with segmented messages as:

$$\begin{aligned} T_{Dissm}^{\log_n P}(m) &= \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[2\, o(2^i\, m) + o_m(2^i\, m) \right] \\ &= (P-1)\, 2\, o + (P-1)\, o_m \end{aligned} \tag{4.34}$$

The three models represent the local copies and the variable part of the expressions as dependent on the number of processes. LogGP and τ–Lop include the fixed part as dependent on the logarithm in base 2 of P. However, τ–Lop provides with more information about the algorithm behavior by including the concurrency in the expressions, affecting to the local copies and transfers.

4.5.3 Bruck

Bruck algorithm [66] is used in MPICH to implement efficiently the *MPI_Allgather* when the size of messages is small. The algorithm could be used as well for implementing the *MPI_Alltoall* and *MPI_Barrier* collective operations. It requires a per-process temporal buffer and three phases. The first phase is a local copy of the message of m bytes in the input buffer to the temporal buffer. The second is composed of $\log_2 P$ stages; in the stage i a process p sends a message of size $2^i\, m$ bytes to the process $p + 2^i$, and receives from the process $p - 2^i$ a message of the same size. The messages are received in the temporal buffer unordered. In the last phase, a process copies from the temporal buffer to the output buffer the $P\, m$ bytes received in the correct order. The three phases are reflected in the following definition:

$$\begin{aligned} \Theta_{Bruck}(m) &= c(m, P) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[P \parallel T_{sr} \left(2^i\, m \right) \right] + c\,(P\, m, P) \\ &= (P+1)\, c(m, P) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[P \parallel T_{sr} \left(2^i\, m \right) \right] \end{aligned} \tag{4.35}$$

The short messages cost is represented as:

$$\begin{aligned}\Theta_{Bruck}(m) &= (P+1)\,c(m,P) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[P \parallel \Big(o(2^i\,m) + 2\,L(2^i\,m, 1) \Big) \right] \\ &= (P+1)\,c(m,P) + \log P\,o(m) + (P-1)\,2\,L(m,P),\end{aligned} \tag{4.36}$$

while the segmented messages cost is:

$$\begin{aligned}\Theta_{Bruck}(m) &= (P+1)\,c(m,P) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \left[P \parallel \Big(o(2^i\,m) + 2\,k\,L(2^i\,S, 1) \Big) \right] \\ &= (P+1)\,c(m,P) + \log P\,o(m) + (P-1)\,2\,k\,L(S,P)\end{aligned} \tag{4.37}$$

In LogGP, the cost of the algorithm with segmented messages is:

$$\begin{aligned}T_{Bruck}^{LogGP}(m) &= \delta\,m + \sum_{i=0}^{\lceil \log(P) \rceil - 1} \Big[L + 2\,o + (S-1)\,G + (k-1)\,\big(g + (S-1)\,G\big) \Big] + P\,\delta\,m \\ &= (P+1)\,\delta\,m + \log P\,\big(L + 2\,o + (S-1)\,G\big) + (P-1)\,(k-1)\,\big(g + (S-1)G\big)\end{aligned} \tag{4.38}$$

$\log_n P$ models the cost as:

$$\begin{aligned}T_{Bruck}^{\log_n P}(m) &= o_m(m) + \sum_{i=0}^{\lceil \log(P) \rceil - 1} 2\,o(2^i\,m) + o_m(2^i\,m) \\ &= P\,o_m + (P-1)\,2\,o\end{aligned} \tag{4.39}$$

As discussed in previous sections, the overhead is referred to the message size for clarifying, and the transfer cost o_m for local copies is introduced in the modeling to represent local memory copies or transfers. Again, the concurrency represented by τ–Lop makes a difference comparing this algorithm and, for instance, the Dissemination algorithm in the former section. While, the costs of LogGP and $\log_n P$ are almost identical in both algorithms.

4.5.4 Pairwise Exchange

In the Total Exchange or Complete Exchange collective operation, referred as *MPI_Alltoall* in MPI, each of the P processes contributes with a buffer of m bytes. The buffer is split in m/P blocks. The process with rank p sends the block in the position j to the process with rank j. A process p receives a block from a process k in the position k of its output buffer. The *Pairwise Exchange* is an algorithm implementing this operation. It requires

$P-1$ stages, after a previous local copy of the local block to the output buffer. In a stage i, rank p sends the block of m/P bytes to $p+i$, and it receives a block from $p-i$, with wraparound. All processes communicate at the same time, leading to a cost of:

$$\Theta_{pexch}(m) = c\left(\frac{m}{P}, P\right) + (P-1) \times \left[P \parallel T_{sr}\left(\frac{m}{P}\right)\right] \tag{4.40}$$

Note that for a complete exchange of a buffer of size m, the blocks have size m/P. Regarding τ–Lop, the costs for short and segmented messages are:

$$\begin{aligned}\Theta_{pexch}(m) &= c\left(\frac{m}{P}, P\right) + (P-1) \times \left[P \parallel \left(o\left(\frac{m}{P}\right) + 2\,L\left(\frac{m}{P}, 1\right)\right)\right] \\ &= c\left(\frac{m}{P}, P\right) + (P-1)\,o\left(\frac{m}{P}\right) + (P-1)\,2\,L\left(\frac{m}{P}, P\right)\end{aligned} \tag{4.41}$$

$$\begin{aligned}\Theta_{pexch}(m) &= c\left(\frac{m}{P}, P\right) + (P-1) \times \left[P \parallel \left(o\left(\frac{m}{P}\right) + 2\,k\,L\left(\frac{S}{P}, 1\right)\right)\right] \\ &= c\left(\frac{m}{P}, P\right) + (P-1)\,o\left(\frac{m}{P}\right) + \frac{(P-1)}{P}\,2\,k\,L\,(S, P)\end{aligned} \tag{4.42}$$

LogGP models the algorithm as a local copy in addition to the $P-1$ stages performing segmented point-to-point message transmissions, with the cost:

$$\begin{aligned}T_{pexch}^{LogGP} &= \delta\,\frac{m}{P} + (P-1) \times T_{p2p}^{LogGP} \\ &= \delta\,\frac{m}{P} + (P-1)\,\Big(L + 2o + (S-1)\,G + (k-1)\,\big(g + (S-1)\,G\big)\Big) \\ &= \delta\,\frac{m}{P} + (P-1)\,\big(L + 2o + (S-1)\,G\big) + (P-1)\,(k-1)\,\big(g + (S-1)\,G\big)\end{aligned} \tag{4.43}$$

Similarly, $\log_n P$ models the cost as:

$$T_{pexch}^{\log_n P} = o_m + (P-1) \times T_{p2p}^{\log_n P} = o_m + (P-1)\,2\,o \tag{4.44}$$

Note that o_m and o refers to a message of size m/P in a Total Exchange operation of m bytes. The three models estimate the cost similarly with respect to the size of the messages sent and the local copies. However, the difference in the modeling comes again in the representation of the concurrency by τ–Lop, which gives a more realistic picture of the behavior of the algorithm.

4.6 The Open MPI COLL SM Broadcast algorithm

This section develops a cost study of a collective algorithm not implemented based on a graph of point-to-point transmissions between pairs of processes, but on shared memory transfers performed by a group of processes concurrently. The goal of the section is twofold:

- Showing the capabilities of τ–Lop to model complex scenarios at a transfer level, rather that at the higher, already discussed, point-to-point transmission level.
- Showing how τ–Lop overcomes the limitations of other models.

The Open MPI Broadcast algorithm implemented in the collective SM (shared memory) component of the library is built based on what we term *collective transfer* construct. The algorithm operates as a tree of degree g (known as *binary* tree for $g = 2$), instead of as a Binomial tree, and it is applied to all the message sizes. The height of a tree of degree g of P processes is $h(P) = \lceil \log_g \left((g-1)\,P + 1 \right) \rceil$. Level i has g^i processes. Data communication is not based on message transmissions between pairs of processes, but on using a memory zone which is shared by all the processes involved in the operation. Specifically, a parent process broadcasts data to its g children through an intermediate buffer, composed by segments of $S = 8$ KB. Look at Figure 4.21, a non-full tree of degree $g = 2$, with $P = 8$ processes, user buffers of $k = 4$ segments (marked as u), and shared intermediate two-segment buffers (marked as b). Given a parent process P_p and a child P_c, child P_c copies every segment from his father's intermediate buffer b_p, first to its own intermediate buffer b_c, and then to its own user buffer u_c. The idea of this approach is that parent and child transfers progress in parallel in pipeline. Thus, child transfers from P_1 to P_3 and from P_1 to P_4 progress, with some delay, in parallel with the parent transfers from P_0 to P_1.

As the concept of transmission disappears from the algorithm, cost terms T vanish from its τ–Lop prediction, which has to be formulated directly in terms of the lower level transfer costs L, a change that does not contribute to expressiveness. Nevertheless, the algorithm has a pipeline behavior, fully detailed in Figure 4.22. Three sections can be identified in the pipeline, the head, the body and the tail, with respective costs Θ_H, Θ_B and Θ_T, so that $\Theta_{SM}(m) = \Theta_H + \Theta_B + \Theta_T$.

An analytical transfer-based expression of the cost can be obtained from this pattern with τ–Lop, though not necessarily as directly and elegantly as from the former transmission-based expressions. It can be shown, for instance, that if $g = 2$, the body is a repetition of $k - 2$ two-stage blocks of identical cost when $\log_2(P)$ is even. Hence

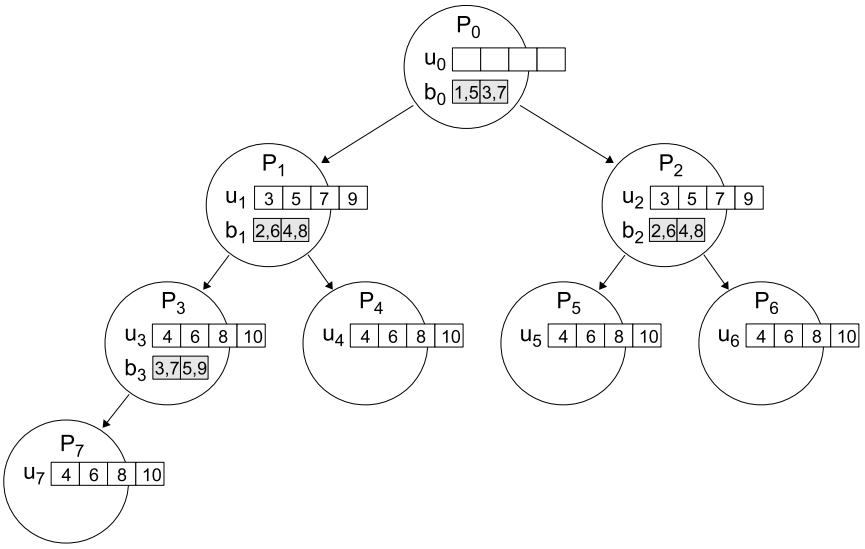


FIGURE 4.21: Buffering layout of the Open MPI COLL SM Broadcast algorithm in a non-full binary tree of $P = 8$ processes. Each involved process P_i has its user buffer u_i divided into $k = 4$ segments u_{ij}, $0 \leq j < k$. In addition, P_i owns an intermediate buffer b_i (in gray) of two segments b_{ij}, $0 \leq j < 2$, in shared memory, used to communicate to his children. Numbers inside the segments are temporal labels. Label n denotes that the segment is written in the pipeline step n. Segment b_{00}, for instance, is written in stage 1 and again in stage 5. Segments b_{01}, u_{10}, u_{20} and b_{30} are written concurrently at stage 3.

		Pipeline stage																			
		1		2		3		4		5		6		7		8		9		10		11	
		src	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*	*src*	*dst*
P_i	i=0	u_{00}	b_{00}	x		u_{01}	b_{01}	x		u_{02}	b_{00}	x		u_{03}	b_{01}	x							
		$L(S,2^0)$				$L(S,2^0)$				$L(S,2^0)$				$L(S,2^0)$									
	i=1,2			b_{00}	b_{i0}	b_{00}	u_{i0}	b_{01}	b_{i1}	b_{01}	u_{i1}	b_{00}	b_{20}	b_{00}	u_{i2}	b_{01}	b_{21}	b_{01}	u_{i3}				
				$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$		$L(S,2^1)$					
	i=3-6					b_{10}	b_{30}	b_{10}	u_{i0}	b_{11}	b_{31}	b_{11}	u_{i1}	b_{10}	b_{30}	b_{10}	u_{i2}	b_{11}	b_{31}	b_{11}	u_{i3}		
						$L(S,2^0)$		$L(S,2^2)$		$L(S,2^0)$		$L(S,2^2)$		$L(S,2^0)$		$L(S,2^2)$		$L(S,2^0)$		$L(S,2^2)$			
	i=7							b_{30}	u_{i0}	x		b_{31}	u_{i1}	x		b_{30}	u_{i2}	x		b_{31}	u_{i3}	x	
								$L(S,2^0)$				$L(S,2^0)$				$L(S,2^0)$				$L(S,2^0)$			

FIGURE 4.22: Evolution of the eleven stages pipeline produced by Figure 4.21. Time runs to the right. Each cell shows the source and destination buffers the stage involves in the top half, and its cost in the bottom half. The total cost of the stage 4, for instance, is $L(S, 2^1) \parallel L(S, 2^2) \parallel L(S, 2^0) = L(S, 2^1 + 2^2 + 2^0) = L(S, 7)$. The cost of the operation is the sum of the costs of all the stages.

$\Theta_B = (k-2)(\Theta_{B_0} + \Theta_{B_1})$, where Θ_{B_0} and Θ_{B_1} are the costs of the first and second stages of the block. A stage Θ_{B_0} has to be added when $\log_2(P)$ is odd (as is the case of Figure 4.22); so, $\Theta_B = (k-2)(\Theta_{B_0} + \Theta_{B_1}) + (log_2 P \bmod 2)\Theta_{B_0}$. Figure 4.22 illustrates that, on a full tree, $\Theta_{B_0}(m) = \mathop{\Vert}_{i=0}^{h(P)-1} L\left(S, g^i\right)$ and $\Theta_{B_1}(m) = \mathop{\Vert}_{i=1}^{h(P)} L\left(S, g^i\right)$.

LogGP and $\log_n P$, transmission-based models, just cannot represent the cost of this algorithm. As an approximation, the LogGP cost would be $T_{SM}^{LogGP} = h(P) \times \Big(2\,o + L + (S-1)\,G + (k-1)\,\big(g + (S-1)\,G\big)\Big)$. $\log_n P$ could predict the cost as

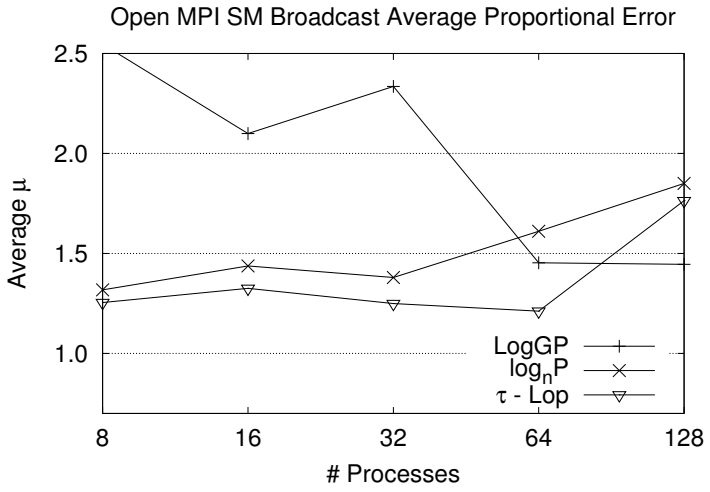


FIGURE 4.23: Average proportional error of the three performance models estimations on the Open MPI COLL SM broadcast algorithm in the *Lusitania* machine. The message size ranges between 8 KB and 16 MB on an increasing number of processes P involved in the operation.

$T_{SM}^{\log_n P} = h(P) \times (2\,o)$, simply by multiplying the height of the binary tree with the cost of a point-to-point message. Of course, neither of the two formulas can draw a picture of the concurrent pipeline behavior of the algorithm. Figure 4.23 shows the average proportional error of the models in *Lusitania*. For some reason, the performance of the Open MPI implementation of the algorithm degrades for $P > 64$, which leads to a noticeable error on τ–Lop. It is also to be noted that with LogGP, the error varies erratically with P. The LogGP results are, therefore, considered unreliable.

Figure 4.24 shows in detail the performance predictions in the *Metropolis* multi-core platform, while Figure 4.25 shows the proportional error. As it can be seen, τ–Lop provides with an accurate estimation for all the range of the messages, except for the larger ones, where the measured performance of the algorithm degrades unexpectedly, raising the error in $\log_n P$ and τ–Lop models. However, LogGP partially fits this change, although its estimation error for the range of messages is globally too high.

4.7 Reduction

A reduction operation involves not only communication, but also computation on the communicated data. τ–Lop models reduction introducing the γ parameter. It represents the computation time in the execution of a collective operation, and its value is highly dependent on several factors, some of them enumerated below:

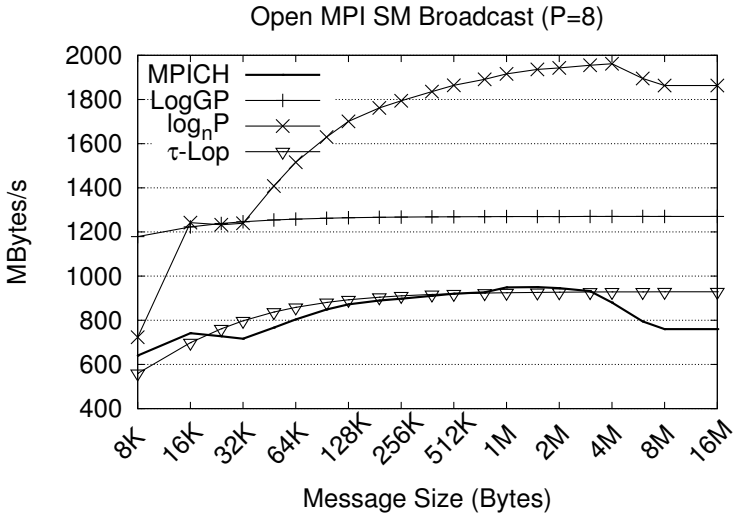


FIGURE 4.24: Bandwidth estimation of the three performance models on the Open MPI COLL SM Broadcast algorithm in the *Metropolis* machine for $P = 8$ processes.

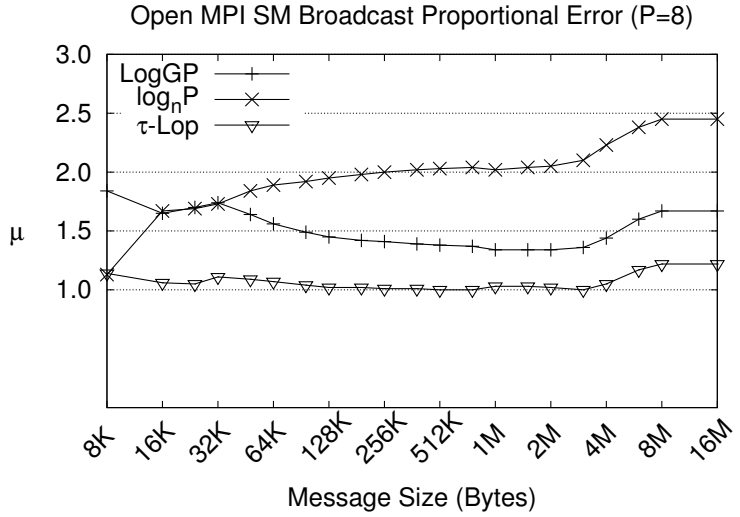


FIGURE 4.25: Proportional error of the three models on the Open MPI COLL SM Broadcast algorithm in the *Metropolis* multi-core machine for $P = 8$.

- As the rest of parameters, it is highly related to the platform, the speed of the processor and the capabilities of the architecture. For instance, the possibility of leaving the communication to the network hardware while computation is performed makes γ difficult to measure.

- Data types of the operands in the operation, their sizes and the hardware to operate them (e.g. floating point units).

- The cache layout of the data to operate and the software mechanisms to extract performance from computation (e.g. data tiling techniques).

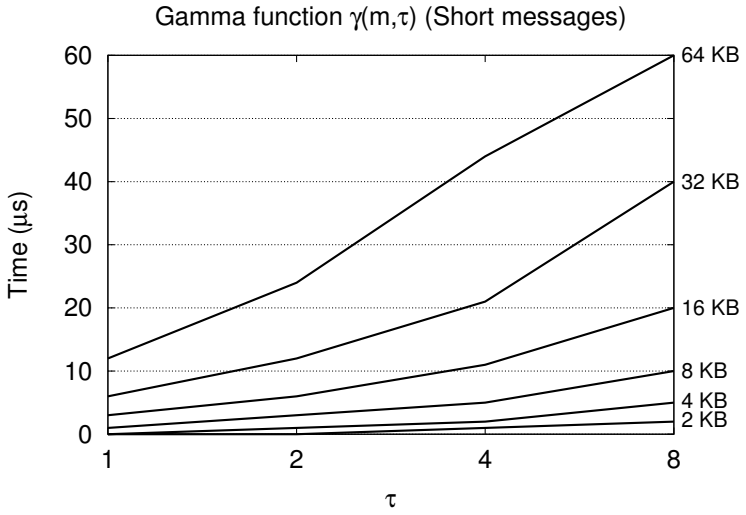


FIGURE 4.26: Estimations of the $\gamma_{MPI\_SUM}(m, \tau)$ parameter for some short message sizes in the *Metropolis* platform. The message, whose size m is shown at the right, is composed of *float* elements. These measurements have been obtained from a micro-benchmark, executed with data out of L3 cache.

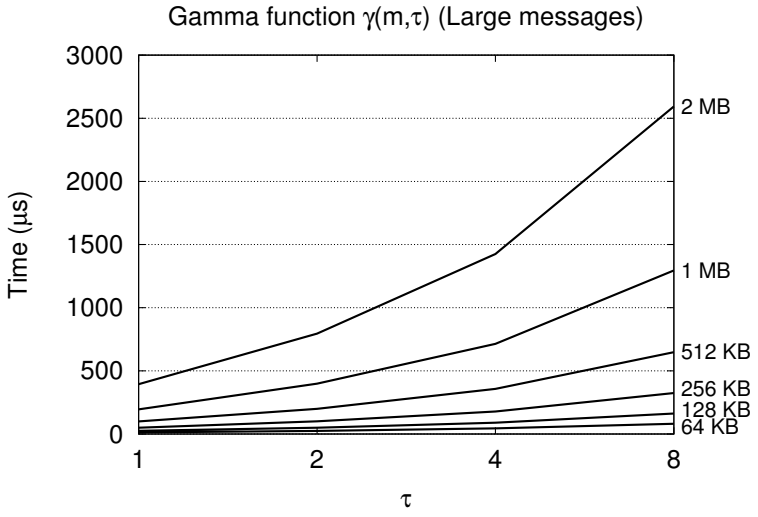


FIGURE 4.27: Estimations of the $\gamma_{MPI\_SUM}(m, \tau)$ parameter for some large message sizes in the *Metropolis* platform. The message, whose size m is shown at the right, is composed of *float* elements. These measurements have been obtained from a micro-benchmark, executed with data out of L3 cache.

Despite of these drawbacks in estimating γ, in the author's view, the parameter is needed for representing the computation time, which usually accounts for an important fraction of the total cost of reduction operations. Furthermore, the consideration of the contention in accessing the operands for computing could be a key factor in the correct estimation of the cost of a reduction operation. Figure 4.26 and Figure 4.27 show the impact of the contention in the addition of two vectors of *float* type elements in the *Metropolis* platform, for increasing number of processes. We implemented a

micro-benchmark to obtain the times shown, which operates two vectors of size m bytes composed of elements of a given datatype. For instance, in a vector of $m = 512$ KBytes of *float* elements there will be $512\,KB/sizeof(float) = 128$ *K-float* elements. The vectors are initially out of the L3 cache, in main memory. As a consequence, the computation time is an upper bound for $\gamma_{op}(m)$.

The MPI standard defines the *MPI_Reduce* collective operation, which operates the elements provided in the input buffer of each process and returns the combined result in the output buffer of the *root*. This section models four algorithms of the *MPI_Reduce* operation used in different MPI libraries. The first algorithm is a Binomial tree used for small messages in MPICH. The second is a combined Reduce-Scatter and Gather operations for long messages. The third is an algorithm that exploits shared memory. It is used in the COLL SM component of Open MPI. The last is an algorithm designed and implemented in AzequiaMPI, taking advance of its shared memory environment. The costs of the algorithms are evaluated in the *Metropolis* platform.

4.7.1 MPICH Binomial Tree reduction algorithm

MPICH implements a Binomial tree algorithm for small messages ($m \leq 2$ KBytes), similar to that used in the Broadcast collective operation (see section 4.1). The algorithm builds the binomial tree from down to top. Every message transmission entails a reduce operation on reception. The intermediate results accumulate in every stage until reaching the root process. The cost of the algorithm is modeled applying short point-to-point messages definition in (3.1), because the operation is done for sizes smaller than a segment (2 KBytes $< S$). Note that computation cost γ is defined and represented as dependent on the number of involved processes:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel T(m) + \gamma_{op}(m, 2^i) \right] \tag{4.45}$$

$\log_n P$ does not model reduction operations. Even in the case of being defined, the term $\gamma_{op}(m)$ would only be dependent on the message size, as it does not consider the effect of the concurrent access to memory. Hence, the reducing cost in $\log_n P$ is introduced by adding computation to the cost of a Binomial tree in (4.6), leading to the cost of:

$$T_{BinRed}^{\log_n P} = \log(P) \left(2\,o + \gamma_{op} \right) \tag{4.46}$$

In LogGP, γ symbol has been used to represent the cost of computing the elements of a vector of size m bytes (see [29] and [13]). Based on the Binomial tree algorithm, the

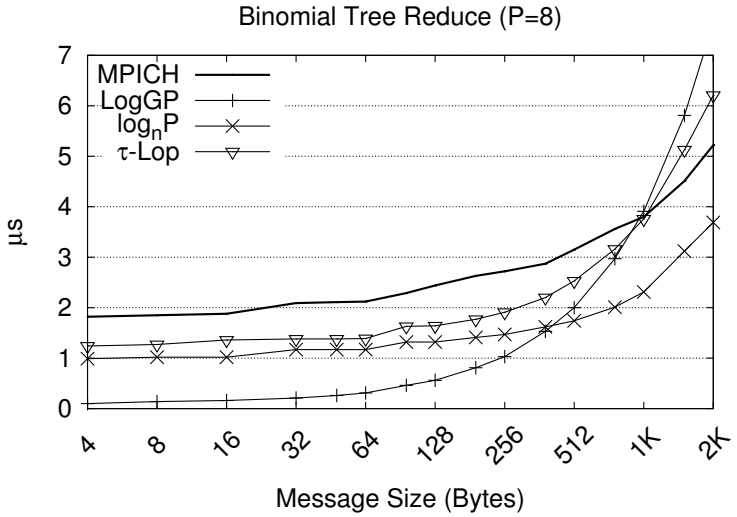


FIGURE 4.28: Cost estimations of the MPICH Binomial tree algorithm for the *MPI_Reduce* collective with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements in the *Metropolis* platform.

cost is:

$$T_{BinRed}^{LogGP} = \log(P)\left(L + 2\,o + (m-1)\,G + \gamma\,m\right) \tag{4.47}$$

Figure 4.28 shows the execution time of the Binomial tree reduction algorithm on the message sizes ranging in 4 Bytes $< m \leq 2$ KBytes, with $P = 8$ and the predefined computation operation MPI_SUM applied to MPI_FLOAT type elements. Figure 4.29 shows the proportional error of the estimations. This algorithm works for short messages, so the concurrency is low. However, added to the computation, contention leaves its footprint in the figures. $\log_n P$ predicts lower execution time than τ–Lop, raising the error in the estimation. LogGP shows an erratic behavior for short messages in shared memory. The mean proportional error thrown by τ–Lop is 1.35, while it is 1.75 in $\log_n P$ and 4.88 in LogGP.

4.7.2 MPICH Reduce-Scatter plus Gather algorithm

For message sizes of $m > 2$ KBytes, MPICH uses a Reduce algorithm which acts in two phases. The first one performs a *Reduce-Scatter* operation (Θ_{RS}) that concentrates the computation. The second phase collects the generated partial results of each process executing a *Gather* operation Θ_{Gth}. The total cost is represented as $\Theta_{RSG}(m)$:

$$\Theta_{RSG}(m) = \Theta_{RS}(m) + \Theta_{Gth}\left(\frac{m}{P}\right) \tag{4.48}$$

Following, the algorithms used in both phases are discussed.

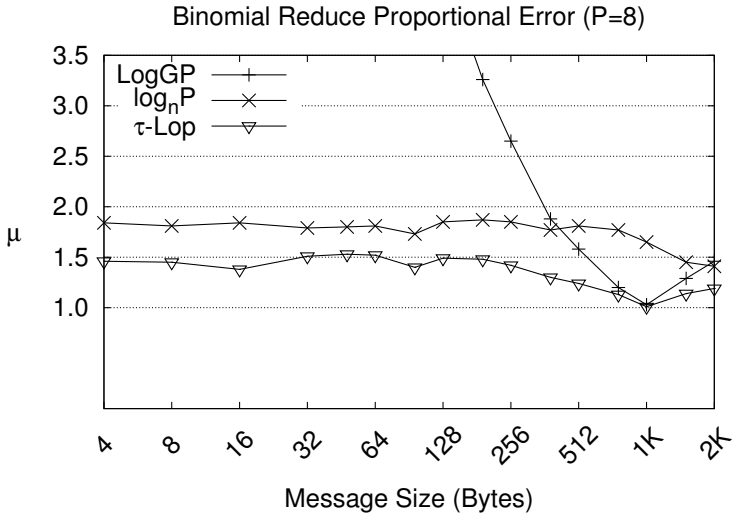


FIGURE 4.29: Proportional error in the modeling of the Binomial tree algorithm for the *MPI_Reduce* collective with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements in the *Metropolis* platform.

The Reduce-scatter operation is implemented using several algorithms depending on the size of the message. For up to $m = 512\,\text{Kbytes}$, a *Recursive Halving* algorithm is used, performing the computation in each stage. This algorithm is the inverse of the Recursive Doubling (see Figure 4.7). The initial message size is m, and it halves in each stage. P processes communicates in each stage using the Send-receive operation defined in (3.7). The Θ_{RS} cost is similar to that in (4.9) adding computation on the received data using the γ parameter, that is:

$$\begin{aligned}\Theta_{RS}(m) &= \sum_{i=0}^{\lceil \log_2(P)\rceil - 1} \left[P \parallel \left(T_{sr}\left(\frac{m}{2^{i+1}}\right) + \gamma_{op}\left(\frac{m}{2^{i+1}}, 1\right)\right)\right] = \\ &= \sum_{i=0}^{\lceil \log_2(P)\rceil - 1} \left[P \parallel \left(o(m) + \frac{2\,k}{2^{i+1}} L(S,1)\right) + \gamma_{op}\left(\frac{m}{2^{i+1}}, 1\right)\right] \\ &= \log P\, o(m) + \sum_{i=0}^{\lceil \log_2(P)\rceil - 1} \left(\frac{k}{2^i} L(S,P) + \gamma_{op}\left(\frac{m}{2^{i+1}}, P\right)\right)\end{aligned} \tag{4.49}$$

In the second phase, the *Gather* operation is implemented by an inverse Binomial Scatter tree, i.e., from leaves to the root (see Figure 4.19). It departs from a message of size m/P, which doubles in each stage, however, the concurrency halves, leading to a cost

of:

$$\begin{aligned}\Theta_{Gth}\left(\frac{m}{P}\right) &= \sum_{i=0}^{\lceil \log_2(P)\rceil -1}\left[\frac{P}{2^{i+1}} \parallel T\left(\frac{2^i\, m}{P}\right)\right] \\ &= \sum_{i=0}^{\lceil \log_2(P)\rceil -1}\left[\frac{P}{2^{i+1}} \parallel \left(o\left(\frac{m}{P}\right) + \frac{2^i\, k}{P}\, L(S,2)\right)\right] \\ &= \log_2(P)\, o\left(\frac{m}{P}\right) + \sum_{i=0}^{\lceil \log_2(P)\rceil -1}\left(\frac{2^i\, k}{P}\, L\left(S, \frac{P}{2^i}\right)\right)\end{aligned} \tag{4.50}$$

Considering the difference between $o(m)$ and $o(m/P)$ as negligible, the cost of the *Reduce-Scatter* algorithm for segmented messages results in:

$$\begin{aligned}\Theta_{RSG}(m) = 2\, \log P\, o(m) &+ \sum_{i=0}^{\lceil \log_2(P)\rceil -1}\left(\frac{k}{2^i}\, L(S,P) + \gamma_{op}\left(\frac{m}{2^{i+1}}, P\right)\right) \\ &+ \sum_{i=0}^{\lceil \log_2(P)\rceil -1}\left(\frac{2^i\, k}{P}\, L\left(S, \frac{P}{2^i}\right)\right)\end{aligned} \tag{4.51}$$

Finally, although it is not evaluated here, for larger messages ($m > 512\,$KBytes), the algorithm used for the Reduce-scatter phase in MPICH is a Pairwise Exchange, modeled in section 4.5.4. The computation cost is added in each phase to the definition (4.42) for segmented messages. The result is:

$$\begin{aligned}\Theta_{RedPE}(m) &= c\left(\frac{m}{P}, P\right) + (P-1) \times \left[P \parallel \left(o\left(\frac{m}{P}\right) + 2\, k\, L\left(\frac{S}{P}, 1\right)\right) + \left(\gamma_{op}\left(\frac{S}{P}, 1\right)\right)\right] \\ &= c\left(\frac{m}{P}, P\right) + (P-1)\, o\left(\frac{m}{P}\right) + \frac{(P-1)}{P}\left(2\, k\, L\,(S,P) + \gamma_{op}\,(S,P)\right)\end{aligned} \tag{4.52}$$

LogGP models the algorithm based on the Recursive Doubling algorithm modeled in (4.11), adding the cost of the computation and the Binomial tree Gather cost (the same as Scatter), as:

$$T_{RSG}^{LogGP} = 2\, \log P\left(L + 2\, o + (S-1)\, G\right) + 2\, \frac{P-1}{P}\, (k-1)\, \left(g + (S-1)\, G\right) + \frac{P-1}{P}\, \gamma\, m \tag{4.53}$$

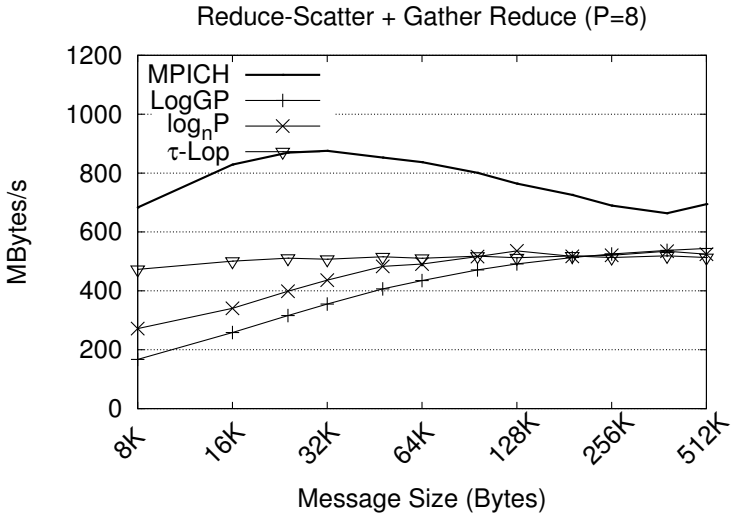


FIGURE 4.30: Modeling of the Reduce-Scatter plus Gather algorithm for the MPICH implementation of the *MPI_Reduce* with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements in the *Metropolis* platform.

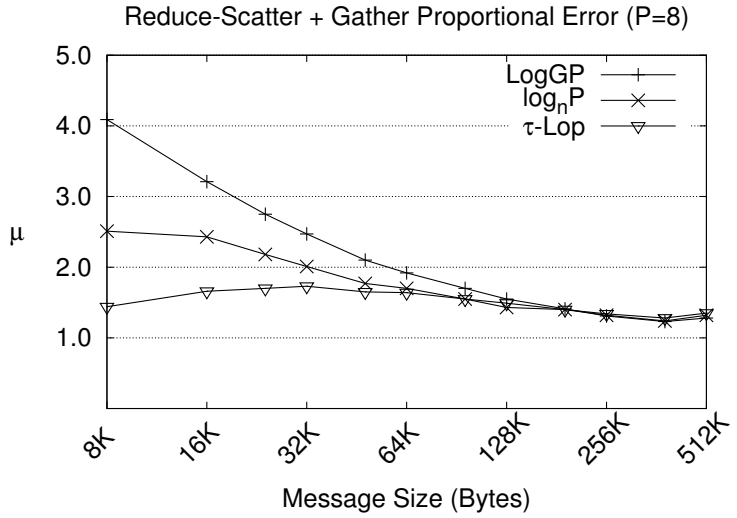


FIGURE 4.31: Proportional error in the modeling of the Reduce-Scatter plus Gather algorithm for the *MPI_Reduce* collective operation with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements in the *Metropolis* platform.

As well, based on the Recursive Doubling cost in (4.12), and the Binomial Scatter cost in (4.24), $\log_n P$ models the algorithm cost as:

$$T_{RSG}^{\log_n P} = o_m + (P-1)(2o + \gamma) + \frac{P-1}{P} 2o \tag{4.54}$$

Figure 4.30 shows the bandwidth estimation for the models with respect to the algorithm execution in MPICH. Due to the complexity of the modeling of this algorithm, the models do not show a good prediction of the cost. They fail specially in short messages, as Figure 4.31 clarifies, however, τ–Lop error is quite lower than that of LogGP and

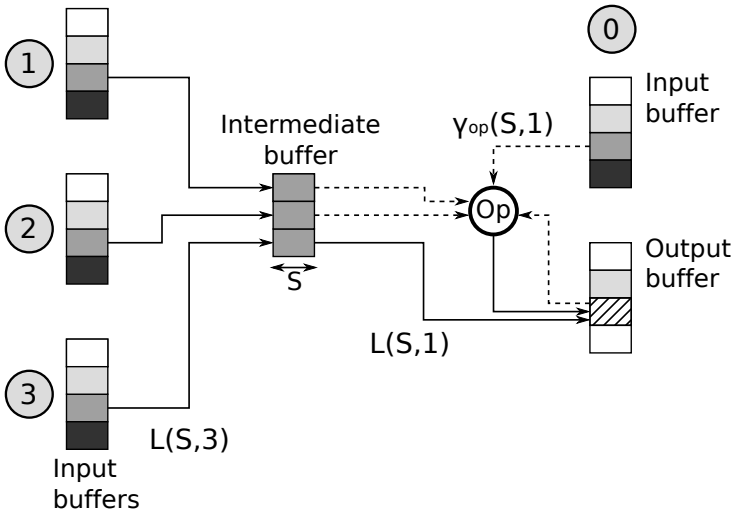


FIGURE 4.32: Open MPI COLL SM Reduce with $P = 4$ processes and root 0 executing the iteration $k = 2$. First, the processes 1 to 3 copy their segments to the intermediate buffer (with cost $L(S,3)$). After that, the root process, first copies the segment of the process with highest rank (3) to its output buffer (with cost $L(S,1)$), and then it operates in inverse rank order with the rest of the segments ($(P-1) \times \gamma_{op}(S,\tau)$).

$\log_n P$. The contention cost in this algorithm, with only $P = 8$ processes, is not enough to the contention modeling makes a significant difference in long messages. LogGP estimation is unexpectedly good, in spite of it does not behave good in other similar algorithms. The mean proportional error of applying τ–Lop in the size range of messages is 1.48, while it raises to 2.50 with LogGP and to 1.68 with $\log_n P$.

4.7.3 Open MPI SM Reduce algorithm

Open MPI COLL SM algorithm implementing the Reduce operates in three stages. The three stages are repeated k times, where $m = k \times S$. Figure 4.32 shows the $k = 2$ iteration of the algorithm with $P = 4$ processes. In the first stage, each process moves a segment of size $S = 8\,\text{KBytes}$, the size of the segment, to a common memory area. The cost of this stage under τ–Lop is $L(S, P-1)$ because all the processes except the root perform the copy concurrently. In the second stage, the root copies the segment from the process $P-1$ to its output buffer. The cost of this stage is of $L(S,1)$. Finally, the root takes the segments one by one in inverse rank order. The root operates each segment to that in the output buffer in terms of the operation specified in *MPI_Reduce*. This last stage has a cost of $(P-1)\,\gamma_{op}(S,1)$. The total cost is:

$$\Theta_{RedSM}(m) = k \times \left[L(S,P-1) + L(S,1) + (P-1)\gamma_{op}(S,1)\right] \tag{4.55}$$

In $\log_n P$, the cost is modeled as:

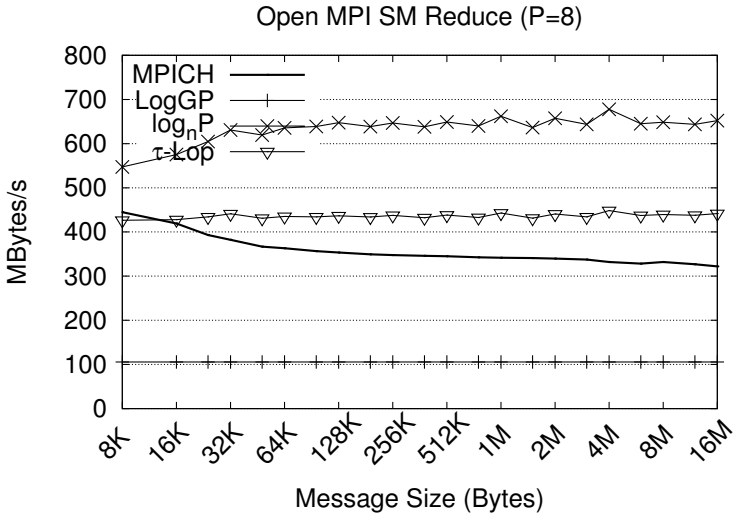


FIGURE 4.33: Open MPI COLL SM Reduce cost estimations together with the real cost with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements.

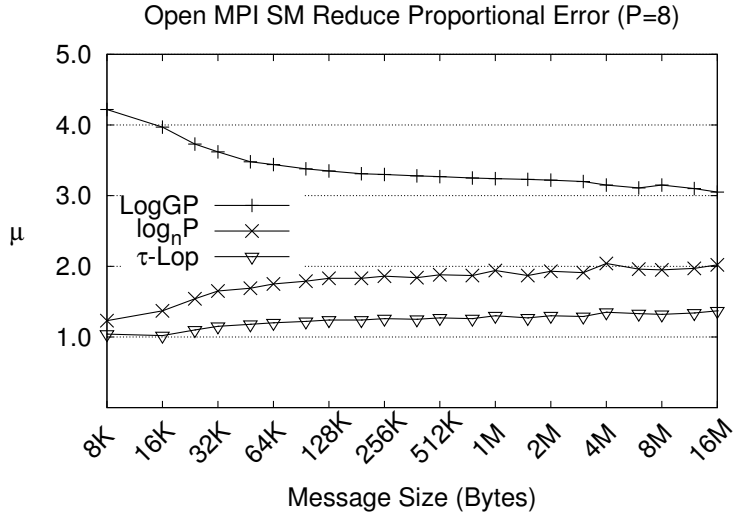


FIGURE 4.34: Mean proportional error in the modeling of the Open MPI COLL SM *MPI_Reduce* with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$ on *float* elements.

$$T_{RedSM}^{\log_n P} = 2\,o + (P-1)\,\gamma_{op}, \tag{4.56}$$

meaning the total amount of bytes in the transfers $(2\,o)$, and the $P-1$ operations performed by the root on the messages of size m from each process.

The cost under LogGP is modeled as the addition of two point-to-point transmissions defined in (2.2) and the cost of the computation:

$$T_{RedSM}^{LogGP} = 2 \times T_{p2p_s}^{LogGP} + (P-1)\,\gamma\,m \tag{4.57}$$

Figure 4.33 compares the application of the models to the experimental execution time

in terms of bandwidth obtained in Open MPI COLL SM Reduce. As it can be seen, the LogGP prediction is far from real values, mainly owing to the fact that the cost expression is an effort for applying the unsuitable LogGP model to an algorithm based on transfers rather than in message transmissions. τ–Lop fits better the real cost in the Open MPI Reduce because it considers the concurrency cost, as Figure 4.34 shows. The mean proportional error of τ–Lop on the sizes of Figure 4.34 is 1.24, while it rises to 1.81 in $\log_n P$ and to 3.37 in LogGP.

4.7.4 AzequiaMPI Reduce algorithm

AzequiaMPI library implements each MPI rank as a thread, therefore, in shared memory all ranks have direct access to the user buffers of the other ranks. AzequiaMPI uses a four-stage reduction algorithm that does not require data movement because the P threads operate directly on the receive buffer provided by the root in *MPI_Reduce*.

At the first stage every non-root thread puts the address of its send buffer at the disposal of the rest by first sending it to root, which receives them in sequence. Next, the root broadcasts a vector with the P addresses using a Binomial tree, with size $P \times m_{addr}$ bytes. This is an Allgather operation whose cost is:

$$\Theta_{Allgth}(m_{addr}) = (P-1)\ T_{p2p}(m_{addr}) + \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel T_{p2p}\left(P \times m_{addr}\right) \right]$$

Note that all transmissions are non-segmented in AzequiaMPI, except tiny messages that come through intermediate buffers for decoupling sender and receiver in the communication. Indeed, the cost of a point-to-point transmission is $T_{p2p} = o(m) + L(m,1)$ in shared memory, because of the fact that data are directly copied from the sender buffer to the receiver buffer.

At the second stage root broadcasts using a Binomial tree the address of its receive buffer, which holds the output vector of *MPI_Reduce*, with a cost of:

$$\Theta_{Bcast}(m_{addr}) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \parallel T_{p2p}\left(m_{addr}\right) \right]$$

During the third stage, every thread operates the data on its original send buffers. To this end every send buffer is chopped into P segments. Thread P_i applies the operation on the P segments i. Figure 4.35 illustrates the set-up of the approach.

$$T_{comp}(m) = P \times \gamma_{op}\left(\frac{m}{P}, P\right)$$

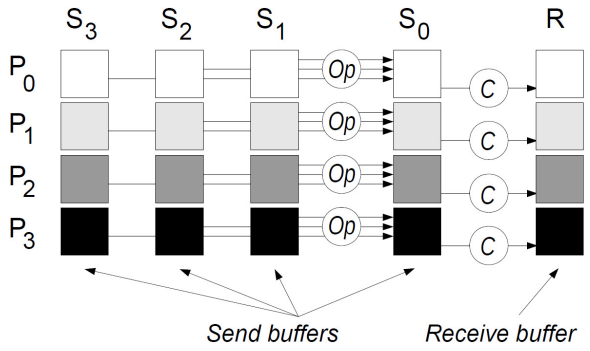


FIGURE 4.35: Representation of the computation phase (T_{comp}) of the AzequiaMPI Reduce algorithm for P processes. S_i is the send buffer of process P_i. R is the receive buffer of root. S_i and R are chopped into P segments S_{ij} and R_j for $0 \leq j < P$. Process $P_i, \forall i$, for $(1 \leq j < P)$ in increasing order of j, applies the operation Op between the elements of S_{ji} and S_{0i}, storing the result in S_{0i}. Finally P_i copies S_{0i} to R_i.

Non-root threads notify root at the last stage that the receive buffer is finished with a Binomial tree synchronization operation. In this operation, the message size is $m = 0$, leading to a cost of:

$$\Theta_{sync}(0) = \sum_{i=0}^{\lfloor \log_g(P) \rfloor - 1} \left[2^i \parallel T\,(0) \right]$$

The total cost of *MPI_Reduce* is then:

$$\Theta_{AzqMpi}(m) = \Theta_{Allgth}(m) + \Theta_{Bcast}(m) + T_{comp}(m) + \Theta_{sync}(0) \tag{4.58}$$

Figure 4.36 shows the application of the models to AzequiaMPI Reduce algorithm. τ–Lop gives in (4.58) a mean proportional error of 1.22. For the $\log_n P$ model the formula $(P-1)\,o(m_{addr}) + \log(P)\,o(P\,m_{addr}) + \log(P)\,o(m_{addr}) + (P-1)\,\gamma$ is applied, taking into account the different message sizes in the transfers and computation. As shown in the figure, the resulting error rate is higher, and the mean proportional error raises to 1.69. LogGP has not enough capabilities to represent correctly this type of algorithms based exclusively on transfers. It was showed in the former section 4.7.3 for the Open MPI Reduce algorithm, and hence, it is omitted here from the discussion.

Below some discussion points are raised. The AzequiaMPI algorithm minimizes data communication because it can afford an in situ calculation approach, working directly on the user space buffers provided by each thread. Figure 4.37 shows how *MPI_Reduce* produces differences in the cost estimation done by τ–Lop and $\log_n P$, attributable to the fact that this algorithm entails concurrency in each stage. The figure also shows that τ–Lop looses precision modeling the reduction operation in AzequiaMPI with respect to Open MPI, carrying a quite pessimistic prediction. This is explained by the fact that parameter γ was built as a "worst case" cost estimation, i.e., with the elements on the

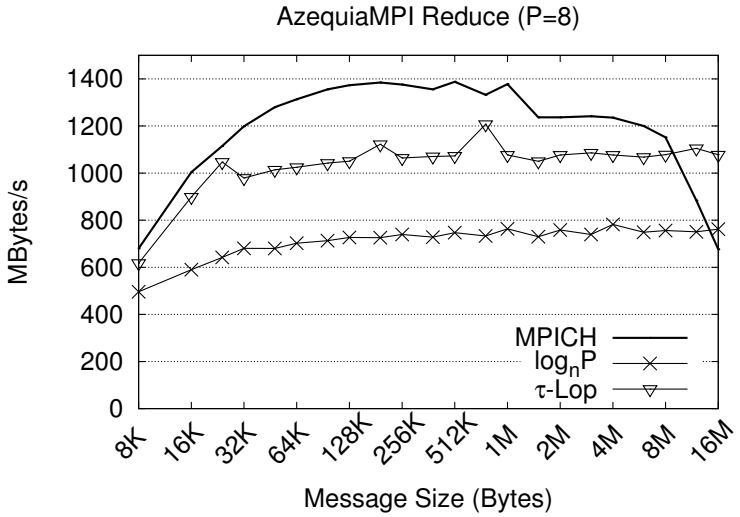


FIGURE 4.36: Modeling of AzequiaMPI Reduce algorithm with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$, on *float* type elements.

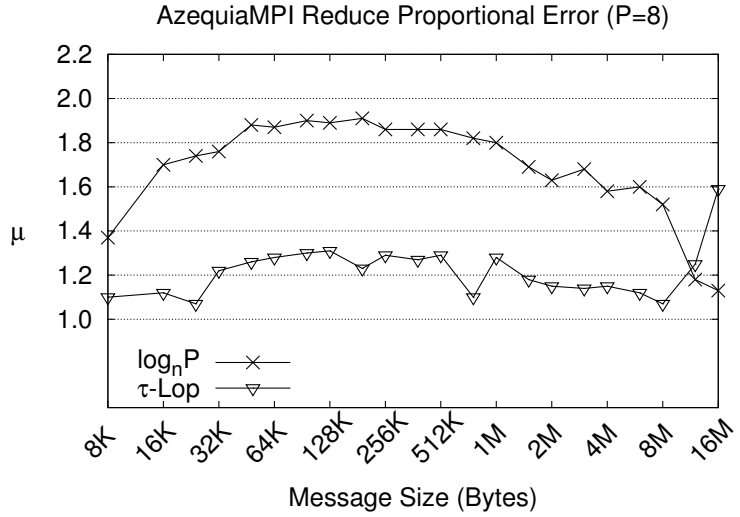


FIGURE 4.37: Mean proportional error of the modeling of the AzequiaMPI Reduce algorithm with $P = 8$ and $\gamma_{MPI\_SUM}(m, \tau)$, on *float* type elements.

vectors to operate out of the cache. This fact speeds up the real Reduce operation with respect to the modeled one.

4.8 Conclusions

The main contribution of τ–Lop discussed in this chapter is its ability to model concurrent transmissions in channels that are shared by all the processes implied in the communication. That results in an improvement of the accuracy of the cost estimations and the representation of a diversity of MPI collective algorithms implemented

in both process-based MPICH and Open MPI libraries, and thread-based AzequiaMPI library, including algorithms with computational costs. The reasons for the poor fit in this domain of the more representative models, LogGP and $\log_n P$, are described. The most important is that the contention cost influence in the overall communication cost is significant, specially in more elaborated algorithms than simple Binomial trees.

In addition, the concurrent transfer concept allows the representation of sophisticated communication techniques used by current implementations of MPI in shared memory, such as message segmentation and collective operations not based on point-to-point operations. It is shown that the expressivity of the cost expressions of the collective operations is key, for instance, in the comparison of a Total Exchange (All-to-all) and the Binomial Scatter operations. LogGP, for instance, erroneously assigns both the same cost. More examples will be shown in the next chapters.

Chapter 5

Modeling MPI collective algorithms in hierarchical multi-core clusters

Modern high performance computing systems are composed of nodes with a significant number of cores per node connected by high performance networks. Obtaining as much performance as possible from such machines requires to consider the hierarchy of communication channels imposed by the difference in the channels bandwidth, in addition to the contention effects due to the sharing of the resources.

Furthermore, related to the contention is the assignation of the ranks to the processes deployed over the system processors (virtual topology or mapping). Assumed a multi-core cluster with a hierarchical organization of communication channels based on their performance, the mapping of processes could improve or aggravate the contention effect.

The contribution of this chapter is the application of the τ–Lop model, which was initially developed to model the behavior of collective algorithms in shared-memory nodes, to hierarchical multi-core clusters. The model extension proposes a new approach to the representation and analysis of parallel algorithms in that architectures, with a focus on MPI collectives. In addition, the extension captures the way the virtual topology defined by the algorithm maps onto the physical topology of the machine. This ability provides a more realistic representation of the algorithm, leading to a better cost prediction.

5.1 Modeling hierarchical platforms with current models

LogGP and $\log_n P$ partially address the hierarchy modeling by adding new parameters to support the representation of the hierarchical communications, as in LogGPH [40] and $m\log_n P$ [42], extensions of the LogGP and $\log_n P$ respectively. This section revisits LogGPH and $m\log_n P$. They are later used in cost evaluations compared to τ–Lop. Multi-core clusters are addressed through the discussion of key algorithms implementing MPI collective operations.

LogGPH supports the representation of hierarchical architectures by a set of parameter values for each communication channel. Considering two communication channels, as shared memory and network, the *latency* parameter, for instance, is represented as L^0 for shared memory latency, and L^1 for network latency. The definition of the parameters, although estimated per-channel, is the same as in LogGP.

$m\log_n P$ extends $\log_n P$ by considering the hierarchy of communication channels in a system. In $m\log_n P$, the cost of a message transmission is the sum of the costs of its individual transfers, each through its specific communication channel. $m\log_n P$ formally characterizes this cost with six parameters:

o : *overhead*, the amount of time the processor is engaged during the transmission or reception of a contiguous message through a communication channel.

l : *latency*, the additional processor time when the message is not contiguous in memory.

g : *gap*, minimum interval of time between consecutive receptions of two messages.

P : the number of processes.

m : the number of different channels in the system.

n : is a vector with one component per communication channel, indicating the number of transfers between two processes along the message path[1]. It is the fundamental contribution of $m\log_n P$ over $\log_n P$.

The cost of a transmission through the channel c is the sum of the costs of its individual transfers[2]:

$$T^c_{p2p} = \sum_{j=0}^{n^c-1} \left(o^c_j + l^c_j\right), c \in \{0, 1, \cdots, m-1\} \tag{5.1}$$

[1]Note that the message *path* of a transmission can involve transfers in different communication channels. However, the term *channel* is used for clarity to refer to the whole transmission.

[2]For convenience, a superscript c to indicate the component of the vector n is used, instead of the most common index notation $n[c]$. The vector n has m components, as it can be deduced from the parameter definition, i.e., $c \in \{0, \ldots, m-1\}$.

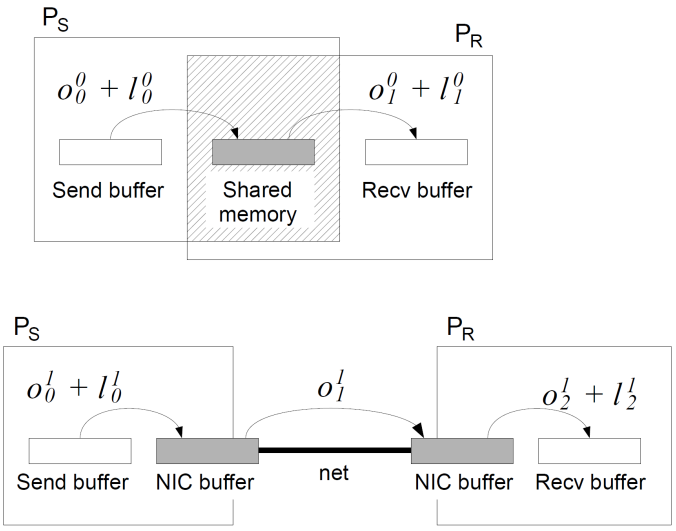


FIGURE 5.1: Concept of *message transmission* in $2\log_{\{2,3\}} P$. The message transmission through the shared memory channel is done in two *transfers* (already shown in Figure 5.1). The network channel needs three transfers. P_S and P_R are the sender and receiver processes respectively.

Thus, the overhead parameter takes a different value depending on the communication channel. Figure 5.1 shows an example of an architecture with two channels, shared memory and network, used to communicate processes intra and inter-node, that is, $m = 2$. The vector indicating the number of transfers per communication channel is $n = \{2, 3\}$. In the shared memory channel ($c = 0$), a transmission needs two transfers $n^0 = 2$, going through an intermediate buffer. In the network channel ($c = 1$) a transmission needs three transfers $n^1 = 3$, because the message is copied from the source process to the network interface card (NIC), transmitted through the network to the destination NIC and being finally copied to the target process. Definition (5.1) is specified using tailored $2\log_{\{2,3\}}P$ model to represent the cost of a transmission in shared memory as:

$$T^0_{p2p} = \sum_{j=0}^{n^0-1} o^0_j = o^0_0 + o^0_1 = o_{mw} \tag{5.2}$$

Note that, in shared memory, the two transfers have the same cost, and their addition is represented originally as o_{mw} (*middleware* overhead). In the network channel, the first and last transfers have the same cost, whose join contribution is represented as o'_{mw}. The intermediate transfer, between NICs, progresses through the network and its cost is o'_{net}:

$$T^1_{p2p} = \sum_{j=0}^{n^1-1} o^1_j = o^1_0 + o^1_1 + o^1_2 = o'_{mw} + o'_{net} \tag{5.3}$$

5.2 Modeling algorithms in hierarchical platforms with τ–Lop

This section evaluates the τ–Lop capability to model three algorithms widely used in MPI collective operations in a multi-core cluster: Binomial tree, Ring and Recursive Doubling. The accuracy, scalability and expressiveness of τ–Lop in modeling these algorithms are compared to that of LogGPH and $m\log_n P$ on a multi-core cluster.

Each algorithm involves P processes, ranked in the range $0 \dots P-1$, deployed over the multi-core cluster. Two communication channels are considered, shared memory and network. The algorithms statically establish a communication graph between the ranks, and execute in a sequence of stages, where the number of involved processes and the message size may change along them. Considering that the mapping of processes to the system processors determines the used communication channels, the algorithm performance highly depends on the chosen mapping. Experimental work is conducted in *Metropolis*, a multi-core cluster using Ethernet (see section 7.1).

Unless otherwise stated, this thesis considers processes deployed in *Sequential mapping*, that is, the process with rank i runs in the processor $i \,\%\, Q$ of the node $i \div Q$, with Q the number of cores per node or, in simple words, processes fill each node before going to the next one. Nevertheless, modeling under other mappings is straightforward. M represents the number of nodes in the cluster.

5.2.1 Binomial Tree

In the Binomial tree algorithm a process named *root* broadcasts a message of size m to the rest of the processes. As introduced in section 4.1, in the first stage, the root sends the message to the process with rank $root + P/2$. The algorithm recursively continues with both processes acting as roots of their respective sub-trees with $P/2$ processes. The number of stages is the height of the tree ($\lceil \log_2(P) \rceil$). The number of sending processes doubles in each stage, whereas the message size remains constant.

LogGPH and $m\log_n P$ estimate the algorithm cost as the height of the tree multiplied by the cost of a point-to-point transmission. Figure 5.2 shows the generated message transmissions in a theoretical multi-core cluster composed of $P = 16$ processes disposed in Sequential mapping on $M = 4$ nodes. As can be distinguished, the number of stages transmitting data through the network channel is $\log_2(M)$, while the number of stages with data transmissions through shared memory is $\log_2(Q)$, with $Q = P/M$ the number

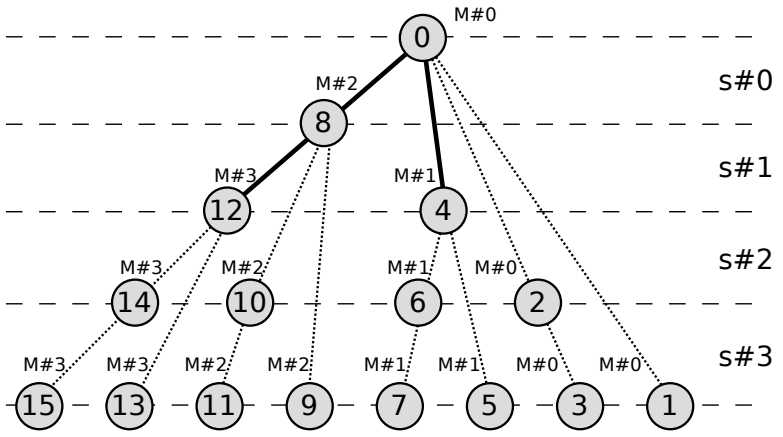


FIGURE 5.2: A Binomial tree with $P = 16$ processes ($root = 0$) deployed in $M = 4$ nodes with Sequential mapping. Transmissions through used communication channels are shown, shared memory (dotted line) and network (bold line). $M\#j$ indicates the destination node of each transmission.

of processes in each node. Adding both costs, the LogGPH estimation is:

$$T_{Bin,LogGPH} = \log_2 M \cdot T^1_{LogGPH} + \log_2 Q \cdot T^0_{LogGPH} \tag{5.4}$$

Note that $T^1_{LogGPH} \neq T^0_{LogGPH}$, because the values of the L, o and G parameters are different in each communication channel (as proposed in the LogGPH model), making the mapping of processes a key factor in the performance of algorithms in a multi-core cluster.

The $m\log_n P$ modeling of the cost is:

$$T_{Bin,m\log_n P} = \log_2 M \cdot (o\prime_{mw} + o\prime_{net}) + \log_2 Q \cdot o_{mw} \tag{5.5}$$

The contention generated by the algorithm highly depends on the deployment of the processes in the system. Take as an example the last stage ($s\#3$) of Figure 5.2, where two transmissions arrive concurrently to each node. Unlike LogGPH and $m\log_n P$, τ–Lop captures this fact, and hence it better models the impact of mapping and its effects on the contention of the algorithm. The cost of the Binomial tree will be represented as:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[C_i^{map} \parallel T^c_{p2p}(m) \right] \tag{5.6}$$

$T^c_{p2p}(m)$ is the cost of a point-to-point transmission through the communication channel c, and is defined in (3.1,3.2) for shared memory and in (3.3) for the network. C_i^{map} is the maximum number of concurrent transmissions through the communication channel used in the stage i, determined by the process mapping (map).

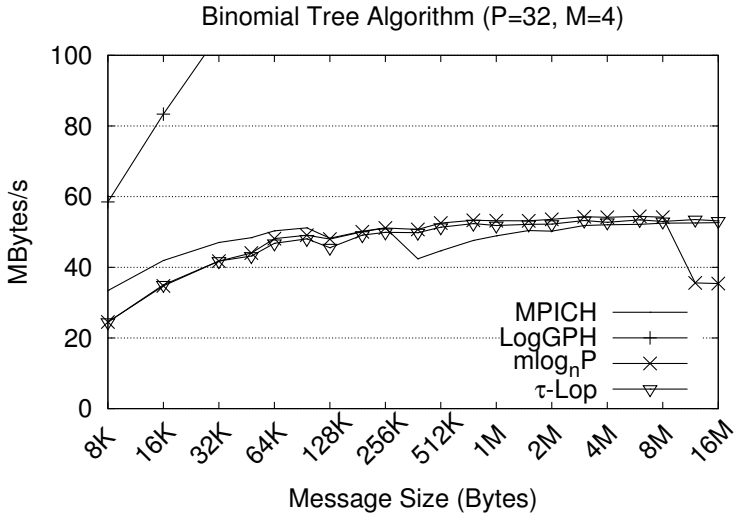


FIGURE 5.3: Estimation of the cost of a Binomial tree compared to the real cost of MPICH in terms of bandwidth. The number of processes is $P = 32$, deployed sequentially on $M = 4$ nodes.

Based on the analysis of Figure 5.2, definition (5.6) can be developed as follows. In the first stage ($s\#0$), there is a point-to-point transmission through the network, with cost $1 \parallel T^1_{p2p}(m)$. In the second stage ($s\#1$) there are two transmissions again progressing through the network. As the Sequential mapping causes that their respective destination nodes to be different ($M\#3$ and $M\#1$), no contention takes place, and cost is $1 \parallel T^1_{p2p}(m)$ (as assumed in section 3.5). The third stage ($s\#2$) has four transmissions in shared memory, scattered in different nodes, with the cost $1 \parallel T^0_{p2p}(m)$. In the last stage ($s\#3$), there are two concurrent shared memory transmissions in each node, therefore, $C_3^{SEQ} = 2$ and the stage cost is $2 \parallel T^0_{p2p}(m)$. The total cost will be:

$$\Theta_{Bin}(m) = 2\,T^1_{p2p}(m) + T^0_{p2p}(m) + 2 \parallel T^0_{p2p}(m) \tag{5.7}$$

Figure 5.3 shows the execution time in terms of bandwidth of the Binomial tree used by the *MPI_Bcast* collective operation of MPICH, for the increasing message sizes and $P = 32$ processes distributed sequentially in $M = 4$ nodes. The MPICH measured bandwidth is compared to the estimations derived from (5.4), (5.5) and (5.7).

LogGPH does not show a good accuracy in this configuration, while the difference between $m\log_n P$ and τ–Lop is small, because the Sequential mapping causes a minimal contention in this algorithm. In the range of long messages, however, τ–Lop shows a slightly better fit.

Figure 5.4 reinterprets the Figure 5.3 to show the *proportional error* (see section 7.6) in the estimation of the cost by models with respect to the real MPICH measured value.

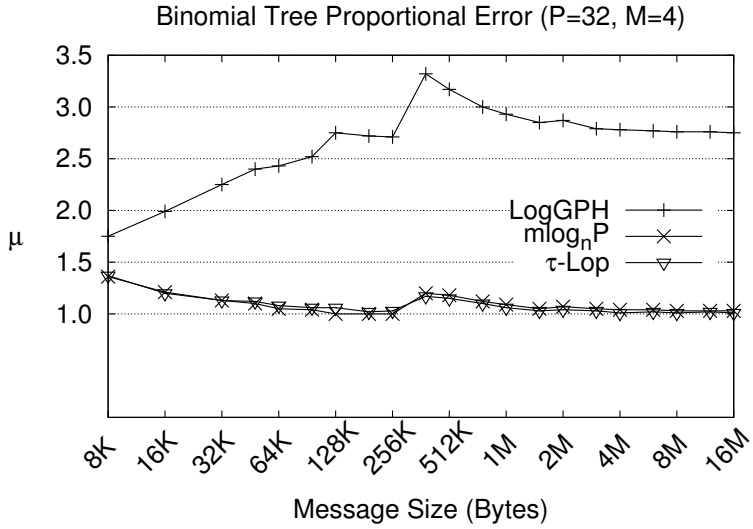


FIGURE 5.4: Proportional error in the cost estimation for a range of medium and large message lengths. Process number is $P = 32$, deployed sequentially on $M = 4$ nodes.

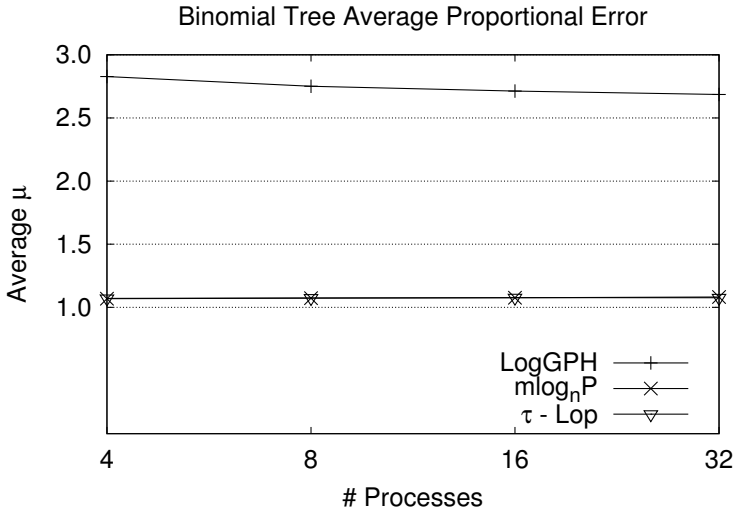


FIGURE 5.5: Average proportional error in the cost estimation of a Binomial tree with increasing number of processes per node in the test platform.

$m\log_n P$ and τ–Lop have a similar error, near to the optimum value $\mu = 1$.

Figure 5.5 represents the mean proportional error in the range of message sizes of Figure 5.4 for increasing number of processes. The cost estimation of the Binomial shows a scalable behaviour, that is, error does not grow with the number of processes, because of the minimal contention of the algorithm in the test platform.

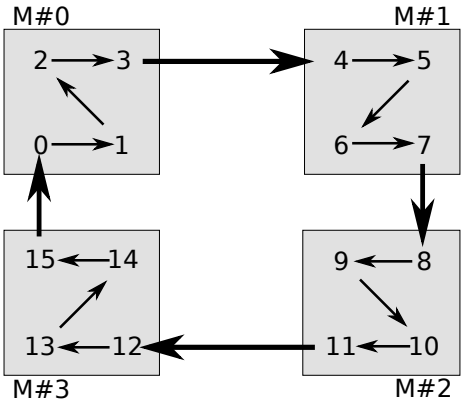


FIGURE 5.6: Representation of the transmissions of the Ring algorithm for $P = 16$ processes in $M = 4$ nodes arranged in Sequential mapping.

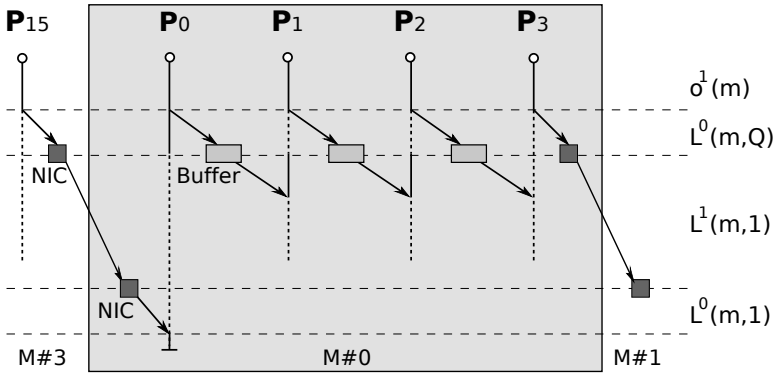


FIGURE 5.7: Ring algorithm short message (no segmented) transmission of sequentially mapped processes with transfers in node $M\#0$ deployed.

5.2.2 Ring

This section evaluates the Ring algorithm, introduced in section 4.3. It is composed of an initial local copy and $P-1$ stages. Each process makes the initial copy of m bytes from its input to its output buffer, with offset $p \times m$. In each stage, the process with rank p sends to the process with rank $(p+1)$ mod P the m bytes received in the output buffer in the previous stage. The number of processes and the message size remain constant along the stages.

Figure 5.6 represents the intra- and inter-node transmissions generated by the algorithm under Sequential mapping. The LogGPH and $m\log_n P$ models estimate the cost of a stage as the most expensive addition of a send and a receive carried out by a process, which in the figure corresponds to the addition of a send through the network and a receive in shared memory (for instance, see the process 3 transmissions). Ignoring the

negligible cost of the initial local copy, the algorithm cost in LogGPH will be:

$$T_{Ring,LogGPH}(m) = (P-1) \times \left[\left(L^0 + 2\,o^0 + (m-1)\,G^0 \right) + \left(L^1 + 2\,o^1 + (m-1)\,G^1 \right) \right] \tag{5.8}$$

Similarly, in $m\log_n P$, according to (5.2) and (5.2), the cost will be:

$$T_{Ring,m\log_n P}(m) = (P-1) \times \left(o_{mw} + o'_{mw} + o'_{net} \right) \tag{5.9}$$

As the Sequential mapping divides evenly the network transmissions among the nodes (see Figure 5.6), the algorithm communication pattern avoids contention in the network channel. Nevertheless, contention in shared memory affects the final cost of the algorithm, and the impact grows with the increase of the number of processes per node Q. In each stage, a rank sends a message to the next rank and receives a message from the previous rank by invoking the *MPI_Sendrecv* operation, with cost represented as $T_{sr}(m)$. Furthermore, the destination and the source rank of both transmissions can be in the same or different node, therefore, transmissions could progress through shared memory or network. τ–Lop models the Ring algorithm as follows:

$$\Theta_{Ring}(m) = (P-1) \times \left[C^{map} \parallel T_{sr}(m) \right], \tag{5.10}$$

where C^{map} is the maximum number of concurrent transmissions in each stage. In the Sequential mapping of Figure 5.6, detailed in Figure 5.7, C^{map} will have the same value along all stages, $C^{SEQ} = Q$. How will $T_{sr}(m)$ be affected by C^{SEQ}? Next, departing from the generic cost expression (5.10), two scenarios are discussed that lead to different cost depending on the message size.

For short messages segmentation is not a concern. As represented in Figure 5.7, the first transfer of a process, either to the NIC or to the intermediate buffer, progresses through shared memory, hence, the transfer cost will be $L^0(m, Q)$. Next, a network transfer of one process per node starts while the rest of processes $(Q-1)$ finish their communication through shared memory. The cost of these transfers is represented as $max\{L^0(m, Q-1), L^1(m, 1)\}$. $Q-1$ transfers contend in shared memory, while one transfer progresses through the network. Note that $L^1(m,1) \gg L^0(m, Q-1)$ in the platform evaluated, with $Q = 8$ (see section 7.1.2). This analysis may change according to the network technology and the number of processes per node, given Sequential mapping. Finally, the last transfer is done by only a receiver process per node from the network (represented as rank P_0 in Figure 5.7). Thus, the total cost will be:

$$\Theta_{Ring}(m) = (P-1) \times \left[L^0(m, Q) + L^1(m, 1) + L^0(m, 1) \right] \tag{5.11}$$

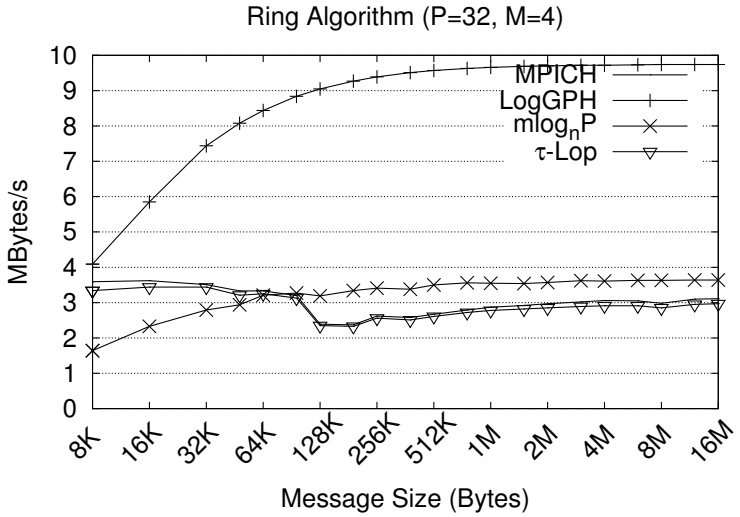


FIGURE 5.8: Estimation of the cost of Ring Allgather with several models, compared to the real MPICH measurements in terms of bandwidth. The number of processes is $P = 32$, deployed on $M = 4$ nodes.

For long messages, each stage is composed of two separate transmissions, which do not intersperse their transfers. The first is in shared memory, as a segmented message transmission by all processes inside each node, hence with cost $Q \parallel 2\,k\,L^0(S,1)$ (see expression (3.7)). The second is a point-to-point transmission performed through the network by a single process in each node, hence without contention, with cost of $1 \parallel T^1_{p2p}(m)$. The total cost ignoring the overhead will be as follows:

$$\begin{aligned}\Theta_{Ring}(m) &= (P-1) \times \left[Q \parallel 2\,k\,L^0(S,1) + 1 \parallel T^1_{p2p}(m)\right] \\ &= (P-1) \times \left[2\,k\,L^0(S,Q) + 2\,L^1(m,1)\right]\end{aligned} \tag{5.12}$$

Figure 6.4 shows the executing time in terms of the bandwidth of the Ring algorithm implemented in the *MPI_Allgather* collective operation of MPICH in the *Metropolis* platform. The measured value is compared to (5.8), (5.9) and (5.11, 5.12). Although the impact of contention on the cost is small in the Ring algorithm with Sequential mapping, because it occurs in shared memory, τ–Lop better fits the real measurements than $m\log_n P$, especially for long messages. The difference slightly increases with respect to the Broadcast in Figure 5.3, due to the stronger shared memory contention. Figure 5.9 shows the proportional error made by the models as compared with the real MPICH measurements.

Figure 5.10 shows the proportional error for the range of message sizes in previous figure, for the increasing number of processes. The modeling of the cost of the Ring algorithm with transmissions, which are progressing concurrently through the two channels, is a complex task. All models underestimate the shared memory contention influence, due

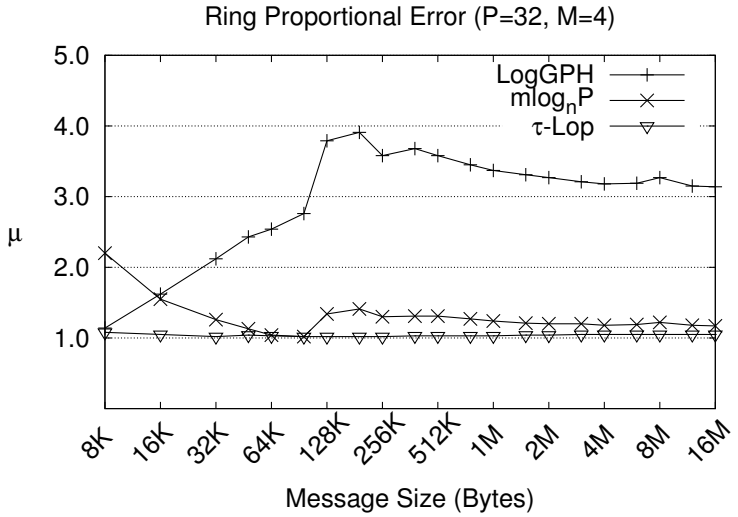


FIGURE 5.9: Proportional error of the Ring cost estimation for a range of medium and large message lengths. $P = 32$ processes deployed in $M = 4$ nodes.

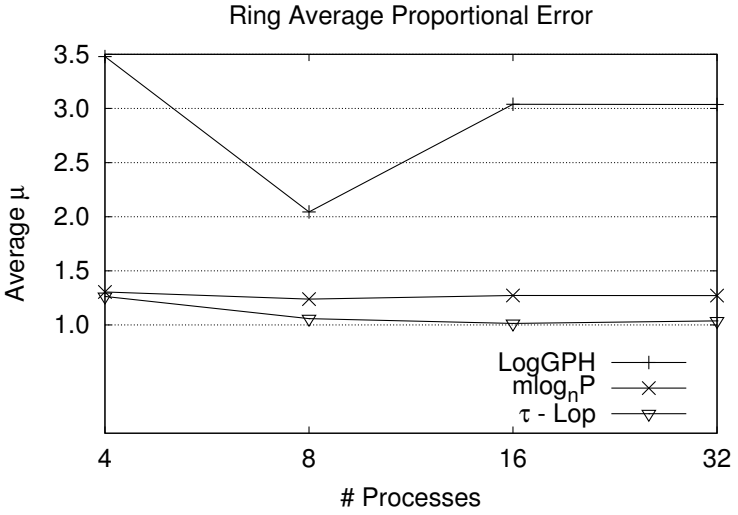


FIGURE 5.10: Average proportional error of the Ring for increasing number of processes deployed on $M = 4$ nodes.

to the fact that, compared to network transmissions, the shared memory cost is low. The cost will become significant when the number of processes per node Q increases.

5.2.3 Recursive Doubling

The Recursive Doubling Allgather algorithm (RDA), discussed previously in section 4.2, is used in collectives such as *MPI_Allgather* and *MPI_Bcast*. The involved processes contribute with a message of size m and receive from the rest of processes $P-1$ messages, ordered by rank in the reception buffer, for a total of $P \times m$ bytes. The algorithm is executed in $log_2(P)$ stages when the number of processes is a power of two. In stage i,

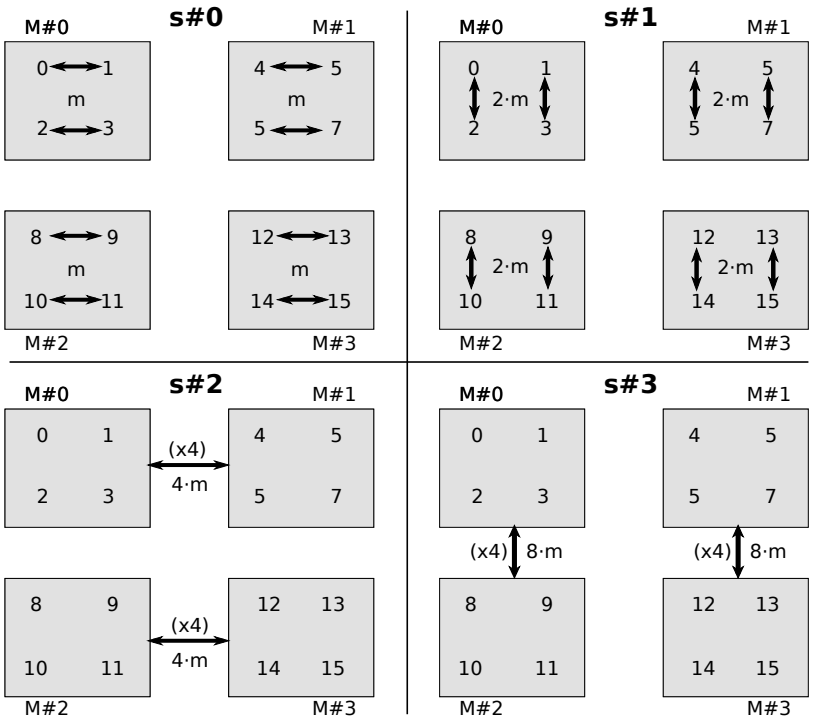


FIGURE 5.11: Stages of the Recursive Doubling algorithm with $P = 16$ processes deployed sequentially in $M = 4$ nodes. A double-headed arrow represents a message interchange of the size specified (m, $2\,m$, $4\,m$ and $8\,m$), whose cost is modeled in τ–Lop as T_{exch}. The number of concurrent interchanges between nodes in the inter-node $s\#2$ and $s\#3$ stages is shown in parenthesis.

the process with rank p exchanges $2^i\,m$ bytes with the process with rank $p \oplus 2^i$. The initial local copy of the m-byte message takes place in each process, from the input to the output buffer. The number of processes communicating in each stage remains constant in this algorithm, whereas the message size doubles.

In a multi-core cluster with Sequential mapping, it holds that the $\log_2(Q)$ initial stages communicate processes in the same node while the rest of stages ($\log_2(M)$) communicate processes in different nodes, giving a total cost of $T_{RDA,LogGPH} = T^0_{RDA,LogGPH} + T^1_{RDA,LogGPH}$. Based on definition (4.10), the cost is:

$$T^0_{RDA,LogGPH} = \log_2 Q \cdot \left(L^0 + 2\,o^0 - G^0\right) + (Q-1)\;m\,G^0 \tag{5.13}$$

$$T^1_{RDA,LogGPH} = \log_2 M \cdot \left(L^1 + 2\,o^1 - G^1\right) + (M-1)\;Q\,m\,G^1 \tag{5.14}$$

Like LogGPH, the $m\log_n P$ model estimates the cost as the addition of the costs of the intra-node and inter-node stages:

$$\begin{aligned} T_{m\log_n P} &= \sum_{j=0}^{\log Q-1} o_{mw} + \sum_{j=\log Q}^{\log P-1} (o\prime_{mw} + o\prime_{net}) \\ &= (Q-1)\, o_{mw} + (M-1)\, Q\ (o\prime_{mw} + o\prime_{net}) \end{aligned} \tag{5.15}$$

Figure 5.11 shows, per stage, the pattern of communication of RDA in a system with $P = 16$ processes sequentially mapped in $M = 4$ nodes. The figure suggest that in the inter-node stages this pattern saturates links between pairs of nodes due to the occurrence of $Q = 4$ concurrent transmissions. As LogGPH and $m\log_n P$ do not model this contention, their cost estimation will be very optimistic. τ–Lop models the algorithm cost as follows[3]:

$$\Theta_{RDA}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[C_i^{map} \parallel T_{exch}^{c}(2^i\, m) \right] \tag{5.16}$$

C_i^{map} represents the maximum number of concurrent transmissions in the stage i given the process mapping *map*. In each stage, two processes interchange a message using an Exchange operation (T_{exch}).

Applied to the case of Figure 5.11, expression (5.16) takes the following form:

$$\begin{aligned} \Theta_{RDA}(m) = {} & C_0^{SEQ} \parallel T_{exch}^{0}(m) + C_1^{SEQ} \parallel T_{exch}^{0}(2\, m) \\ & + C_2^{SEQ} \parallel T_{exch}^{1}(4\, m) + C_3^{SEQ} \parallel T_{exch}^{1}(8\, m) \end{aligned}$$

The first two stages are executed independently in each node, with two concurrent intra-node interchanges, which makes $C_0^{SEQ} = C_1^{SEQ} = 2$. As discussed in section 3.6, the cost of the Exchange operation is $T_{exch}^{0}(m) = o^0(m) + 2\, L^0(m, 2)$. The inter-node stages include four concurrent transmissions involving two nodes, so that $C_2^{SEQ} = C_3^{SEQ} = 4$, because four transmissions arrive to the node NIC concurrently. The cost of the Exchange operation in the network is $T_{exch}^{1}(m) = o^1(m) + 2\, L^0(m, 1) + L^1(m, 1)$, also discussed in section 3.6. As a result, the cost will be as follows:

$$\begin{aligned} \Theta_{RDA}(m) = {} & 2\, o^0(m) + 2\, o^1(m) + \\ & + 2 \parallel 2\, L^0(m, 2) + 2 \parallel 2\, L^0(2\, m, 2) \\ & + 4 \parallel \left[2\, L^0(4\, m, 1) + L^1(4\, m, 1) \right] \\ & + 4 \parallel \left[2\, L^0(8\, m, 1) + L^1(8\, m, 1) \right] \end{aligned} \tag{5.17}$$

[3]Note that he T_{sr} operation could be used to model the RDA algorithm as well, as $C_i^{SEQ} \parallel T_{sr}$, and T_{sr} between each $P = 2$ processes.

Applying the *linearity principle* defined in section 3.3, the expression (5.17) is simplified to:

$$\begin{aligned}\Theta_{RDA}(m) = {} & 2\,o^0(m) + 2\,o^1(m) + \\ & + 2\,L^0(m,4) + 4\,L^0(m,4) \\ & + 8\,L^0(m,4) + 4\,L^1(m,4) \\ & + 16\,L^0(m,4) + 8\,L^1(m,4),\end{aligned}$$

finally resulting in the cost:

$$\Theta_{RDA}(m) = 2\,o^0(m) + 2\,o^1(m) + 30\,L^0(m,4) + 12\,L^1(m,4)$$

Figure 5.12 shows the cost, in terms of bandwidth, of the RDA algorithm implemented in the *MPI_Allgather* collective of MPICH, for a range of increasing message size and $P = 32$ processes deployed in $M = 4$ nodes, and hence $Q = 8$. The measured cost is compared to the estimations (5.13,5.14), (5.15) and (5.16). One interesting conclusion can be made for long messages, when we can ignore the latency cost. In this case, the LogGPH cost becomes $7\,m\,G^0 + 24\,m\,G^1$, $m\log_n P\ 7\,o_{mw} + 24\,(o\prime_{mw} + o\prime_{net})$ and the τ–Lop cost $62\,L^0(m,8) + 24\,L^1(m,8)$. Let us have a look to the network costs in these expressions, higher than the shared memory costs, and with a 24 factor in all models. The main difference between them is the contention, only captured by τ–Lop. The contention in the network depends on the $Q = 8$ argument, which impacts the performance and limits the algorithm scalability. The contention effect in the RDA algorithm with Sequential mapping in a multi-core cluster is high in both channels, which makes the LogGPH and $m\log_n P$ cost predictions substantially lower than the real-life measurements.

Figure 5.13 shows the proportional error in the estimation of the models against the real values observed in MPICH. The error made by τ–Lop is markedly lower in the whole range of message sizes, near to the optimum value $\mu = 1$. Finally, Figure 5.14 shows the proportional error through the range of message sizes in the previous figure for a growing number of involved processes P. The error of the τ–Lop estimation remains constant with P, whereas the error of the other models increases remarkably. In summary, the ability of τ–Lop to capture the effect of the contention in the channels (specially in network) for concurrent transfers provides the model with an accuracy that scales well with the number of the involved processes.

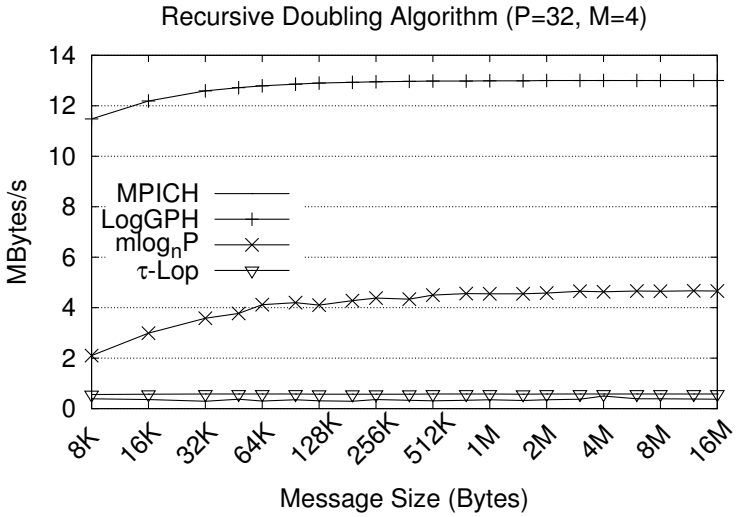


FIGURE 5.12: Estimation of the cost of Recursive Doubling Allgather with several models, compared to the MPICH measured cost in terms of bandwidth. The number of processes is $P = 32$, deployed on $M = 4$ nodes.

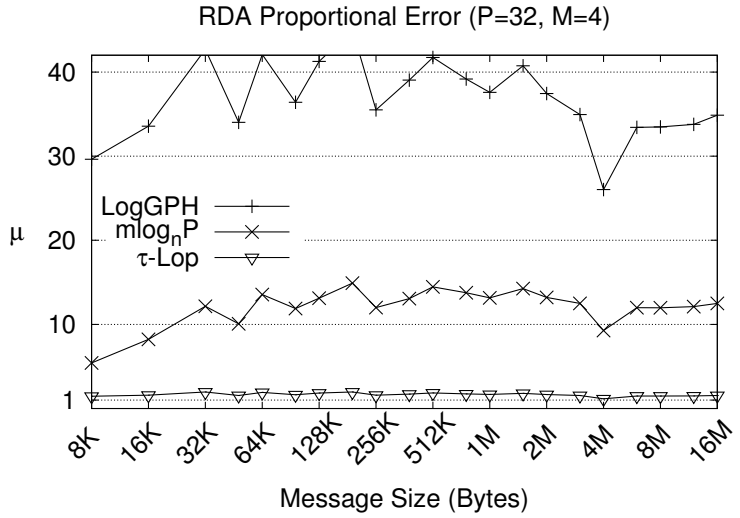


FIGURE 5.13: Proportional error of the Recursive Doubling cost estimation for a range of medium and large message lengths. The number of processes is $P = 32$, deployed on $M = 4$ nodes.

5.3 Conclusions

The algorithms discussed in this chapter, being fairly common, have been chosen because their complexity significantly exceeds that of the algorithms evaluated in previous related works. We analyze them under three models in hierarchical platforms. LogGPH extension of the LogGP model is a representative of the broadly used linear models. They base on network-related latency and bandwidth parameters, and estimate the cost of an algorithm as the cost of the longest path of the involved point-to-point transmissions. $m\log_n P$, an extension of the $\log_n P$ model, changes previous view of the point-to-point

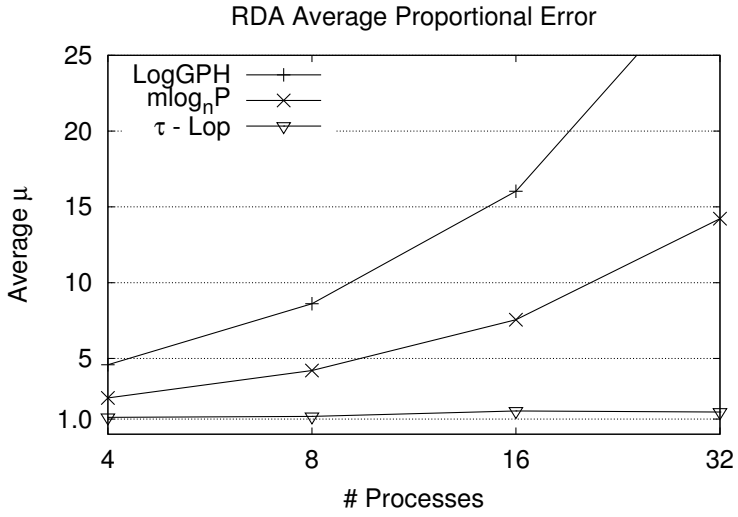


FIGURE 5.14: Average proportional error of the Recursive Doubling for increasing number of processes deployed on $M = 4$ nodes.

transmission as the cost unit modeled using network parameters and breaks it into data transfers progressing through the hierarchy of communication channels.

τ–Lop, initially conceived to model communication in shared channels, also models the impact of the uneven performance communication channels on multi-core clusters. Notwithstanding, its representation of the cost is, in the author's view, both simple and generic, and extensible to cover hardware specific technologies details.

This chapter reveals that overlooking the representation of the contention leads to unacceptable estimation errors. It also shows how the mapping of processes has a decisive impact on the cost of the algorithm, making clear that it needs to be modeled to provide an accurate prediction.

Although the algorithms have been evaluated under Sequential mapping, the *Round Robin* mapping[4] is commonly used as well, and leads to different cost estimations, as it will be extensively discussed in the next chapter. For instance, Round Robin mapping is detrimental to the Binomial and Ring algorithms, while it improves the performance of the Recursive Doubling. Rico et al. [23] supply a study of the influence of the mapping on the cost of collective algorithms. As an example, the average proportional error of the Binomial tree algorithm under Round Robin mapping is shown in Figure 5.15. Under round robin mapping, the network contention in the algorithm grows with the increase of the number of processes in last stages. It can be seen that the error in the estimation of the $m\log_n P$ model increases with P, while τ–Lop error remains nearly constant. LogGPH cost estimation behaves erratic, and hence its error. The scalability has been demonstrated to be influenced by the number of cores per node, in addition

[4]Round Robin mapping places a rank p in the node $p\%M$.

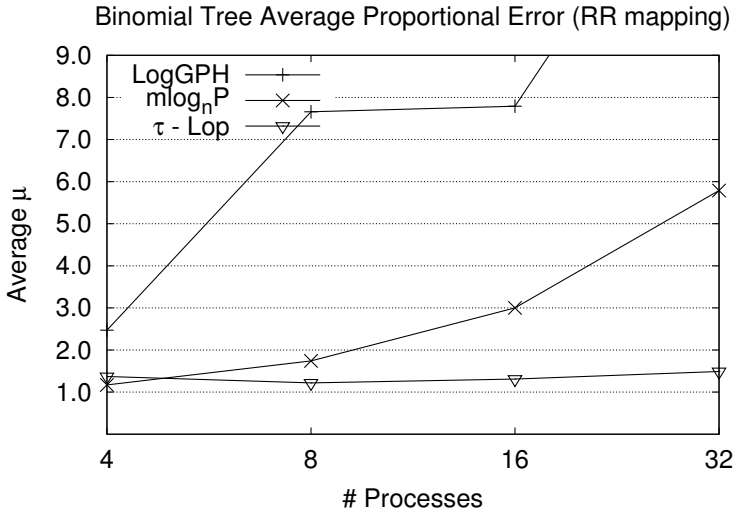


FIGURE 5.15: Mean proportional error of the Binomial tree under the Round Robin mapping of processes, for increasing number of processes deployed on $M = 4$ nodes.

to the number of nodes in a cluster. τ–Lop estimation error does not depend on the number of processes, hence showing higher scalability.

In summary, in the author's view a transfer-based model with the capacity of model contention and communication heterogeneity as τ–Lop, offers a higher analysis capacity to drive algorithm optimization, as well as accurate and scalable cost predictions.

Chapter 6

Analysis capabilities of τ–Lop

This chapter shows the analysis capabilities of τ–Lop through two cases of study.

The first case of study compares two common algorithms modeled in shared memory: Binomial tree and Recursive Doubling, when used in the implementation of the *MPI_Scatter* and the *MPI_Allgather* collective operations respectively. Such comparison give deeper insights in the effect of the contention in the shared memory communication channel, already discussed in chapter 4. Although both algorithms execute in the same number of stages, the former has a significantly lower cost because in each stage the number of transmissions progressing concurrently is smaller, resulting in lower contention.

The second case of study discusses the Ring algorithm when it is used in the *MPI_Allgather* collective operation in a multi-core cluster. Its cost is estimated for two common mappings under LogGPH, $m\log_n P$ and τ–Lop models. The ability of a model to capture the mapping influence on the cost (introduced in the conclusions of chapter 5) is an important issue. A sensitive model must provide with mechanisms to decide the optimal mapping of an algorithm in a given platform.

6.1 Case study 1: Comparing the costs of the Binomial Scatter and Recursive Doubling Allgather algorithms

This section goes into detail about the influence of the contention in the performance of algorithms, and clarifies some shortcomings in the cost expressions made by current models. As a conduit example, two common collective operations are evaluated: a Scatter and an Allgather. The algorithm for Scatter is a Binomial tree, and the Recursive Doubling for Allgather. The shared memory communication channel is considered, and

segmented point-to-point messages are used in the modeling. First, the algorithms are modeled under LogGP, and then, τ–Lop is used to compare their costs.

The Scatter Binomial tree executes in $\lceil \log_2(P) \rceil$ stages (see section 4.4), the height of the tree, as shown in Figure 4.18 and Figure 4.19. P is considered as power of two for simplicity. Note that the number of concurrent transmissions doubles in each stage, while the size of the message halves, so that the amount of data moved keeps constant along the stages.

Under LogGP, the total cost for the Binomial Scatter is calculated as the cost of the larger path from the root process to a leaf. The cost of a point-to-point segmented message transmission is defined in (2.2), and simplified for clarity in the following form[1]:

$$\begin{aligned} T_{p2p_S} &= L + 2\,o + (S-1)\,G + (k-1)\,(g + (S-1)\,G) \\ &\approx L + 2\,o + S\,G + (k-1)\,(g + S\,G) \end{aligned}$$

The total cost of the Binomial Scatter algorithm is the addition of the point-to-point transmissions costs of each stage, as shown in the Table 6.1.

TABLE 6.1: Binomial Scatter in LogGP

Stage	Message Size	Cost
1	$m/2$	$L + 2o + S\,G + ((k-1)/2)\,(g + S\,G)$
2	$m/4$	$L + 2o + S\,G + ((k-1)/4)\,(g + S\,G)$
3	$m/8$	$L + 2o + S\,G + ((k-1)/8)\,(g + S\,G)$
. . .		
$\log_2 P$	$m/2^{\log_2 P}$	$L + 2o + S\,G + ((k-1)/P)\,(g + S\,G)$
Scatter total cost		$\log_2 P\,(L + 2o + S\,G) +$ $+((P-1)/P)\,(k-1)\,(g + S\,G)$

As Figure 4.18 illustrates, the number of stages is $\log_2(P)$, and the message size halves in each stage. In the table 6.1, the addition of the *fixed* parts of the point-to-point costs is $\log_2(P) \times (L + 2\,o + S\,G)$, while the *variable* parts addition is:

$$\begin{aligned} &\left(\frac{k-1}{2} + \frac{k-1}{4} + \frac{k-1}{8} + \ldots + \frac{k-1}{P}\right) \times (g + S\,G) = \\ &(k-1) \times \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{P}\right) \times (g + S\,G) = \\ &(k-1) \times \left(\frac{P-1}{P}\right) \times (g + S\,G) \end{aligned}$$

For its part, as introduced in section 4.2, the Recursive Doubling also completes in $\lceil \log_2(P) \rceil$ stages, as shown in Figure 4.7 and Figure 4.8. In comparison to the Scatter,

[1] Only one communication channel is considered, shared memory, and hence superscripts are not used in this section.

TABLE 6.2: Recursive Doubling Allgather in LogGP

Stage	Message Size	Cost
1	m/P	$L + 2o + S\,G + (1\,(k-1)/P)\,(g + S\,G)$
2	$2\,m/P$	$L + 2o + S\,G + (2\,(k-1)/P)\,(g + S\,G)$
3	$4\,m/P$	$L + 2o + S\,G + (4\,(k-1)/P)\,(g + S\,G)$
. . .		
$\log_2 P$	$2^{\log P-1}\,m/P$	$L + 2o + S\,G + ((k-1)/2)\,(g + S\,G)$
Allgather total cost		$\log_2 P(L + 2o + S\,G)+$ $+((P-1)/P)\,(k-1)\,(g + S\,G)$

the initial size of the message is $m' = m/P$, doubling at each stage. Table 6.2 shows the Allgather cost modeling in LogGP, with $\log(P)$ stages and P concurrent transmissions in each stage.

Interestingly, both operations have the same cost under LogGP, a definitely inaccurate result. The whole amount of data moved is quite different: $\log_2 P \cdot (m/2)$ bytes in the Scatter versus $(P-1) \cdot m$ bytes in the Allgather, as can be deduced from Figure 4.18 and Figure 4.7, a fact that LogGP is unable to represent.

Next, the modeling and comparison of both algorithms are carried out with τ–Lop. Departing from the point-to-point segmented transmission defined in (3.2), the Binomial Scatter algorithm cost under τ–Lop defined in (4.20) is reproduced here for convenience:

$$\Theta_{BinSct}(m) = \log P\, o(m) + \sum_{i=0}^{\log P-1} \left\lceil \frac{k}{2^{i+1}}\, L\left(S, 2^{i+1}\right) \right\rceil \tag{6.1}$$

The factor $\frac{k}{2^{i+1}}$ shows clearly the decreasing of the message size per stage, while the 2^{i+1} parameter of L shows the increasing of the contention in the Binomial tree.

The cost of the Recursive Doubling algorithm was discussed in the section 4.2, for a message of size m (see definition (4.9)). The cost modeling is reproduced next for a message of size m/P, which corresponds to the size of the message used in the invocation of the *MPI_Allgather*, compared to m in the *MPI_Scatter*:

$$\begin{aligned}
\Theta_{RDA}\left(\frac{m}{P}\right) &= \sum_{i=0}^{\log P-1} \left\lceil P \parallel T_{sr}\left(2^i\,\frac{m}{P}\right) \right\rceil \\
&= \sum_{i=0}^{\log P-1} \left\lceil P \parallel \left(o(m) + \frac{2^{i+1}\,k}{P}\, L\,(S,1)\right) \right\rceil \\
&= \sum_{i=0}^{\log P-1} \left\lceil o(m) + \frac{2^{i+1}\,k}{P}\, L\,(S,P) \right\rceil \\
&= \log(P)\, o(m) + \sum_{i=0}^{\log P-1} \left\lceil \frac{2^{i+1}\,k}{P}\, L\,(S,P) \right\rceil
\end{aligned} \tag{6.2}$$

For (6.1) and (6.2), they can be further expanded as (the overhead is omitted for clarity):

$$\Theta_{BinSct}(m) = \frac{k}{2} L(S,2) + \frac{k}{4} L(S,4) + \cdots + \frac{k}{P} L(S,P) \tag{6.3}$$

$$\begin{aligned}\Theta_{RDA}\left(\frac{m}{P}\right) &= \frac{2\,k}{P} L(S,P) + \frac{4\,k}{P} L(S,P) + \cdots + \frac{P\,k}{P} L(S,P) \\ &= 2\,k\,\frac{P-1}{P} L(S,P)\end{aligned}$$

Introducing additional contention in the Binomial Scatter (6.3) in all the stages except in the last one, an upper bound for $\Theta_{BinSct}(m)$ is obtained:

$$\begin{aligned}\Theta'_{BinSct}(m) &= \frac{k}{2} L(S,P) + \frac{k}{4} L(S,P) + \cdots + \frac{k}{P} L(S,P) \\ &= k\,\frac{P-1}{P} L(S,P)\end{aligned}$$

Now, we have that

$$\Theta_{RDA}\left(\frac{m}{P}\right) = 2\,\Theta'_{BinSct}(m) > 2\,\Theta_{BinSct}(m), \tag{6.4}$$

that shows that the cost of RDA (Θ_{RDA}) exceeds more than twofold that of the Binomial Scatter (Θ_{BinSct}).

Figure 6.1 shows both RDA and Binomial Scatter costs measured in the *Lusitania* platform for different number of processes. The relative average costs ($\Theta_{RDA}(m)/\Theta_{BinSct}(m)$) for the range of the message sizes shown are 2.61 for $P = 32$, 2.83 for $P = 64$ and 3.02 for $P = 128$, increasing with the number of processes because of the difference in the contention between operations. For any m and P, RDA cost more than doubles that of Scatter, what confirms the τ–Lop prediction in (6.4).

As a conclusion, this case of study illustrates why the modeling of the cost of a collective operation has to take into account the contention of the processes in accessing the communication channel, also in a NUMA architecture as *Lusitania*. As LogGP ignores this issue, it makes errors like giving the same cost estimation on two quite different collective algorithms with the same number of stages. Regarding the $\log_n P$ model, the cost modeling of both collectives in definitions (4.12) for RDA and (4.24) for the Binomial Scatter, shows that the RDA cost is near P times higher than Scatter, a result far from being correct:

$$\frac{T_{RDA}}{T_{BinSct}} = \frac{o_m + (P-1)\,2\,o}{\frac{P-1}{P}\,2\,o} \approx P \tag{6.5}$$

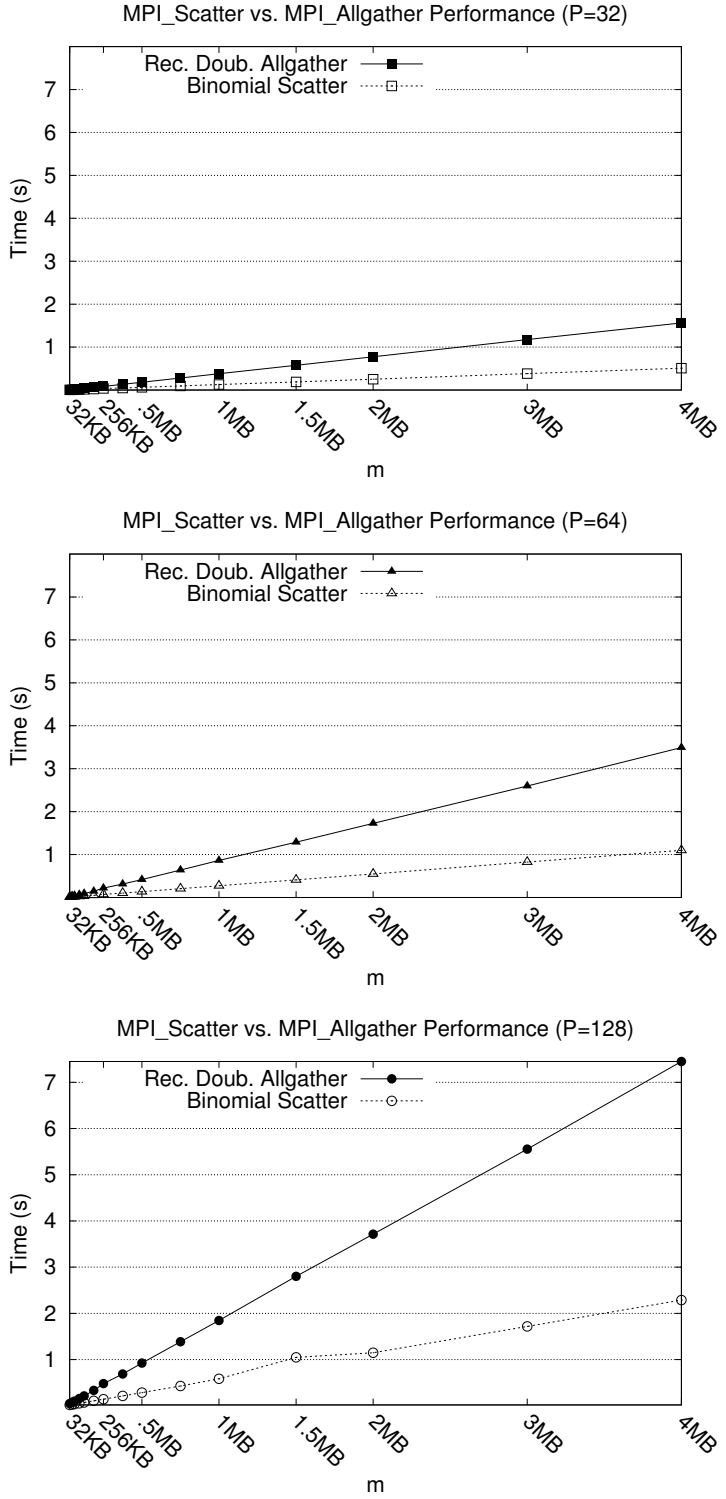


FIGURE 6.1: MPICH performance of Binomial Scatter tree and Recursive Doubling Allgather with different number of processes P in the *Lusitania* machine.

6.2 Case study 2: Comparing the costs of different mappings for the Ring Allgather algorithm

Parallel algorithms implementing collective operations schedule a communication graph between the ranks involved in the operation. Thus the process mapping determines the channels used for message transmissions over the multi-core cluster, and hence the contention in such channels. This section shows the capabilities of models to capture the mapping effect in the overall cost. As a conduit example, we discuss the Ring algorithm when used in the *MPI_Allgather* collective operation.

The *Metropolis* platform with the shared memory and network channels is considered (see section 7.1.2). Two commonly used mappings are evaluated. *Sequential* mapping deploys processes on cores in a way that a rank p runs in the node $p \div Q$ while *Round Robin* mapping runs a process p in the node $p \,\%\, M$.

The section 5.2.2 discussed the cost of the Ring algorithm under Sequential mapping, showed in Figure 5.7, and reproduced at the right side of the Figure 6.2. The algorithm executes in $P-1$ stages. In each stage, a process with rank p invokes the *MPI_Sendrecv* operation for communicating with the nearest ranks. The destination and the source ranks of both point-to-point transmissions composing the Send-receive could be in the same or in different nodes, therefore, transmissions could progress through shared memory or network. Under Sequential mapping, P_{15} receives from process P_{14} through shared memory and then sends through the network to P_0, while P_5 transmits only through shared memory. LogGPH models the cost of a stage as the cost of the most expensive Send-receive, that considering $Q \geq 2$, is the addition of a shared memory and a network transmissions. Hence:

$$T_{Ring}^{SEQ} = (P-1) \times \Big[\big(L + 2\,o + (m-1)\,G\big) + \\ + \Big(L^1 + 2\,o^1 + (m-1)\,G^1\Big) \Big] \tag{6.6}$$

where L is the latency of shared memory and L^1 that of the network, and likely for o and G. The cost for large messages approaches to $(P-1)\,m\,\left(G + G^1\right)$, with the latency and the overhead costs negligible. $G^1 \gg G$, i.e. the cost of the transmission of a byte is higher through the network than through shared memory, as can be seen in section 7.2.

Under Round Robin mapping (see left side of the Figure 6.3) all the messages travel through the network. Hence, each process sends and receives a message of size m with the cost:

$$T_{Ring}^{RR} = (P-1) \times 2 \times \Big(L^1 + 2\,o^1 + (m-1)\,G^1\Big) \tag{6.7}$$

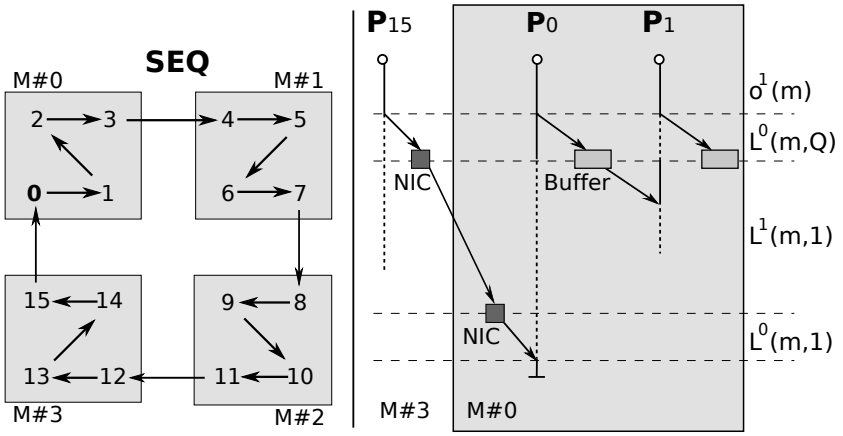


FIGURE 6.2: The Ring algorithm with $P = 16$ processes in $M = 4$ nodes with Sequential mapping, and a message transmission of P_0 deployed at right side.

The cost for long messages approaches to $(P-1)\,2\,m\,G^1$, nearly doubling (6.6), the Sequential mapping cost. Although LogGPH provides useful extensions to LogGP to reflect different communication channels in the system, an important fact is that under LogGPH both cost expressions (6.6) and (6.7) are independent of the number of processes per node Q, as showed later, a non plausible result.

$m\log_n P$ follows a similar approach. The costs for Sequential and Round Robin mappings are:

$$T_{Ring}^{SEQ} = (P-1) \times \left(o_{mw} + o'_{mw} + o'_{net}\right) \tag{6.8}$$

$$T_{Ring}^{RR} = (P-1) \times 2 \times \left(o'_{mw} + o'_{net}\right) \tag{6.9}$$

Again, a cost modeling independent of the number of processes in a node.

As LogGPH and $m\log_n P$, τ–Lop models the Send-receive operation (T_{sr}) as the addition of two point-to-point transmissions. The cost of each individual transmission depends on the channel it progresses and on the contention it suffers, this derived from the process mapping and represented by C^{map} in definition (5.10). Besides, the Ring algorithm behavior can be analyzed at transfer level. As discussed in section 5.2.2, the cost of the Ring algorithm under Sequential mapping is $\Theta_{Ring}^{SEQ}(m) = (P-1) \times \left[L^0(m,Q) + L^1(m,1) + L^0(m,1)\right]$. Note that Q affects only to shared memory transfers.

The behavior under Round Robin mapping is shown at the right size of the Figure 6.3. The three transfers composing the transmission progress concurrently for the Q processes in each node, leading to:

$$\Theta_{Ring}^{RR}(m) = (P-1) \times \left[L^0(m,Q) + L^1(m,Q) + L^0(m,Q)\right] \tag{6.10}$$

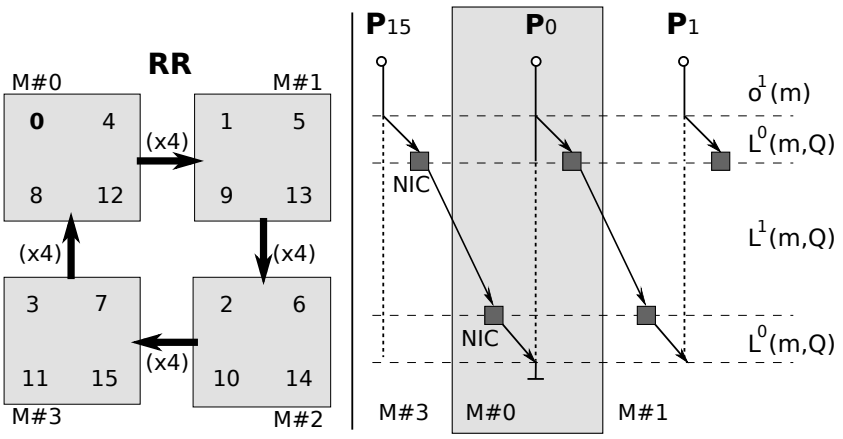


FIGURE 6.3: Ring algorithm with $P = 16$ processes in $M = 4$ nodes with Round Robin mapping, and a message transmission of P_0 deployed at right side with the costs of the concurrent transfers under τ–Lop.

Comparing both Sequential and Round Robin mappings, the difference of their costs is:

$$\Theta^{RR}_{Ring} - \Theta^{SEQ}_{Ring} = (P-1) \times \Big[\Big(L^0(m,Q) - L^0(m,1) \Big) + \Big(L^1(m,Q) - L^1(m,1) \Big) \Big]$$

The difference is the contention in the channels, determined by the number of processes per node Q, a fact not represented neither in LogGPH/$m\log_n P$ nor in any other model.

Figure 6.4 shows the execution costs in terms of the bandwidth of the Sequential and Round Robin mappings of the Ring algorithm implemented in the *MPI_Allgather* collective operation in MPICH. The number of processes is $P = 32$, deployed in $M = 4$ nodes.

In Figure 6.4, the upper plot shows the Sequential mapping cost figures. Here the influence of contention on the total cost is relatively small, due to it occurs in the faster shared memory channel. Notwithstanding, τ–Lop achieves a better fit than LogGPH and $m\log_n P$ in long messages, where the contention effects increase. In any event, to determinate the impact of the shared memory contention on the total cost is a complex issue. The LogGPH and $m\log_n P$ models underestimate this fact, leading to an increase in the estimation error in platforms with a high Q.

Bottom plot shows that, under Round Robin mapping, the algorithm communication pattern saturates the links between nodes with Q concurrent transmissions. MPICH long messages performance decreases sixfold in *Metropolis* with only $M = 4$ nodes and $Q = 8$ cores per node. As LogGPH does not model the contention derived cost, its cost estimation is very optimistic. The cost nearly halves in the Round Robin mapping, as can be deduced from the estimations in definitions (6.6) and (6.7), with shared memory

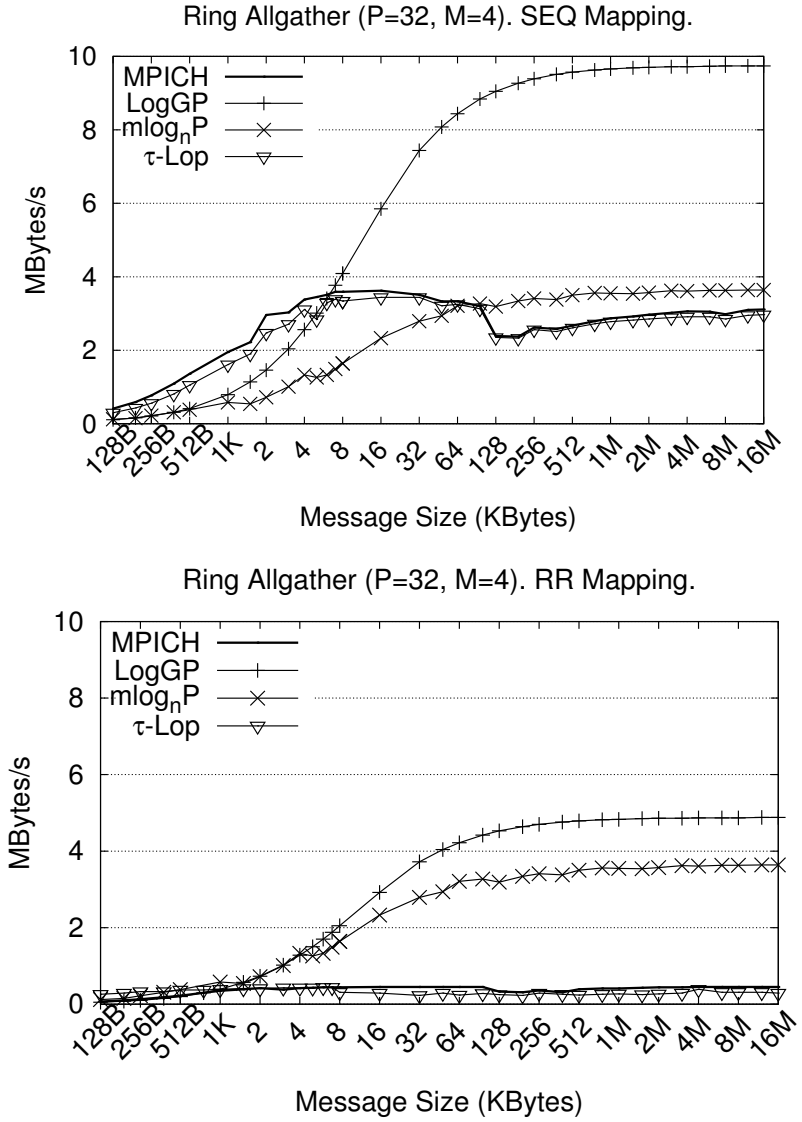


FIGURE 6.4: Estimation of the cost of the Ring Allgather with LogGP, $m\log_n P$ and τ–Lop models, compared to real MPICH measured cost bandwidth. The number of processes is $P = 32$, deployed on $M = 4$ nodes under two mappings, Sequential (above) and Round Robin (below).

transmission cost negligible. $m\log_n P$ costs from definitions (6.8) and (6.9) slightly vary between mappings, because the term o_{mw} has a low influence in the total cost. While, τ–Lop estimation remains close to the real value. It captures the high contention effect in the Ethernet link in the modeling of the costs Θ^{SEQ}_{Ring} and Θ^{RR}_{Ring}. Note that an increase in Q will lead to a wider error in LogGPH and $m\log_n P$, while τ–Lop shows itself more scalable. The development of the models leads to identical conclusions for all the range of message sizes, including the segmentation of messages in shared memory, as can be deduced from the performance in the figures.

6.3 Conclusions

This chapter discusses the analysis capabilities of τ–Lop through two cases of study involving common algorithms which are present in implementations such as MPICH or Open MPI.

The first example shows the influence of the contention on the representation of the cost of algorithms. The bandwidth of shared channels shrinks when several transmissions progress in parallel. τ–Lop represents concurrent transmissions through a shared communication channel using a simple but still powerful notation, with the overall result of improving the cost prediction. It is shown that two algorithms as different as the Binomial Scatter and the Recursive Doubling Allgather are allocated the same cost by LogGP, while it is quite different in practice, as τ–Lop predicts.

The second example discusses the influence of the process mapping on the cost of an algorithm. Such mapping decides the channels used for communicating in the multicore cluster, and hence, it has a high impact on the cost of the algorithm. A model has to capture the algorithms behavior derived from the mapping, to gain scalability and accuracy in their cost estimations.

In the author's view, a transfer-based model as τ–Lop offers a higher analysis capacity derived from a meaningful representation of the cost of the algorithms, that leads to a good accuracy of the cost prediction.

Chapter 7

Parameters measurement

An analytical model represents the cost of collective algorithms as a combination of the parameters of the model. Therefore, a correct measurement of the parameters is key to achieving accurate predictions. For measuring the parameters of LogGPH in the test platforms we follow the methodology proposed in [32], and in [42] for $m\log_n P$. The chapter contributes with a methodology for estimating the parameters of τ–Lop. First of all, the testing platforms are described.

7.1 Test platforms and software

The experimental work of this thesis has been carried out in two different platforms: *Lusitania* and *Metropolis*. *Lusitania* is used to represent and estimate the cost of algorithms in shared memory, under the contention effects. The multi-core cluster *Metropolis* is also used to evaluate the contention in networks and, in addition, to estimate the impact of mapping in the cost of the algorithms.

7.1.1 Lusitania

The *Lusitania* platform is a ccNUMA HP Superdome sx2000 machine installed at the Extremadura Supercomputing Center (CenitS) in Trujillo, Spain. It consists of 64 Dual-Core Intel Itanium2 Montvale 9140N processors. Each core has a private $9MB$ L3 cache, that make up a total of 128 cores, split into 16 UMA nodes. *Lusitania* is used as an SMP test platform due to similar intra-UMA and inter-UMA communication channels performance. The operating system is GNU/Linux 2.6.16, and the compiler used was GCC version 4.1.2.

7.1.2 Metropolis

The experimental platform *Metropolis*, installed at the Escuela Politécnica, in Cáceres, Spain, consists of four nodes connected by 10 Gigabit Ethernet Intel 82574L adapters through a 10 GbE switch Dell PowerConnect 8024F. Each node has two 2.4 GHz Intel Xeon E5620 (Nehalem) processors, making a total of eight cores per node, hyperthreading disabled. The cache figures are 12MB of shared L3, 256KB of L2 and 32 KB of Harvard L1. The operating system is GNU/Linux 2.6.32 and the compiler used was GCC version 4.4.6.

7.1.3 Software

The IMB (Intel MPI Benchmark) suite [58] version 3.2 is used to obtain the execution time and bandwidth data, by invoking the MPI point-to-point and collective operations in the MPI libraries and measuring their execution time and bandwidth. The measurements of a collective are based on a high number of executions, and the time measured is the time from the beginning of the collective to the moment when the last process ends its execution. NetPipe [67] version 3.7 is used to measure the time of point-to-point message transmissions and memory copies in the cases specified in this thesis. Both tools provide with the execution time data of communication operations. Note that, whatever benchmarking tools used for communication time measurement, they all require to deal with issues as cache effects and buffer reuse, to ensure the accuracy and reproducibility of such measurements ([14, 68, 69]).

MPICH version 1.4.1p1 and Open MPI version 1.7 are used as the reference MPI implementations. While MPICH is used upon Nemesis for the collective operations based on point-to-point messages, Open MPI is used mainly for the shared memory collective algorithms based on intermediate memory mapped to the processes involved using its SM component. Although not included in this document, Open MPI Tuned component implementing point-to-point based collectives has been evaluated, showing similar results than MPICH collectives cost predictions.

7.2 LogGPH and $\log_n P$ parameter estimation

For estimating the parameters of the LogGP model we follow the widely used procedure described by Kielmann et al. in [32]: First, the MPI LogP Benchmark is used for estimating the values of the parameters of the *Parametrized LogP* (PLogP) model, introduced at the end of the section 2.1. After that, a simple conversion table from

TABLE 7.1: Conversion from PLogP to LogGP parameter values.

LogGP Parameter	PLogP Parameter
L	$L + g(1) - o_s(1) - o_r(1)$
o	$\left(o_s(1) + o_r(1)\right)/2$
g	$g(1)$
G	$g(m)/m$, for a large m value

PLogP to LogGPH parameter values is applied. The table is provided in work [32] and reproduced in table 7.1.

Following, we discuss the MPI LogP Benchmark procedure to measure the parameters of the PLogP model: the overhead in the sender $o_s(m)$ and receiver $o_r(m)$, the latency L and the gap per message $g(m)$. The overhead in the sender $o_s(m)$ is measured as the average time of sending messages of size m. The overhead in the receiver $o_r(m)$ is measured using *Round Trip* messages[1], that is, the processor P_i sends a message of size 0 to the processor P_j, and receives a message of size m as the response. The processor P_i posts the receive after a time enough for the response message has reached it. The overhead $o_r(m)$ is exclusively the time measured for the posted receive. Both overheads are based on a high number of repetitions, and average is taken.

Regarding the gap per message parameter $g(m)$, the MPI PLogP benchmark provides with two methods of measuring it: *direct* and *optimized*, described in [32]. The *direct* method is expensive because of the high number of messages used. The *optimized* method is based on sending only one message, but it has been demonstrated as highly inaccurate by Lastovetsky and Dongarra [70]. In the *direct* method, Round Trip Time (RTT) of sets of increasing number of messages is measured until changes in $g(m)$ is under 1%, when channel saturation is assumed to be reached. The saturation of the channel makes the network latency negligible with respect to the bandwidth. If the set of n messages sent for a size m has a completion time of $s_n(m)$, then the gap per message is calculated as $g(m) = s_n(m)/n$.

Finally, the *Latency* parameter L is determined from the RTT of a message of size 0 as:

$$RTT(0) = 2 \times (L + g(0)) \implies L = \frac{RTT(0)}{2} - g(0) \tag{7.1}$$

Table 7.2 shows the LogGPH parameters yielded for the shared memory channel in *Lusitania* and for the shared memory and network channels in *Metropolis*. Times are provided in μsecs.

Following the instructions of the $m\log_n P$ authors in [42], the *Parametrized Round Trip Time* (PRTT) defined in [71] is used to measure the o'_{net} time for a range of message

[1] A *Round Trip* message is composed of a point-to-point message and the response from the receiver.

TABLE 7.2: LogGPH parameter values (μsecs) in Lusitania (shared memory) and Metropolis (shared memory and network).

Parameter	Lusitania SM	Metropolis SM	Metropolis NET
L	0.500	1.683	32.986
o	0.350	0.1095	2.8775
g	0.120	1.649	4.841
G	0.0009890	0.0001923	0.0092

sizes. The PRTT(n,d,m) time for a set of n messages sent, a delay of d between message transmissions and the size of message m, is used to calculate the network overhead as:

$$PRTT(n, 0, m) = 2 \times o'_{net}$$

Then, standard blocking *MPI_Send* and *MPI_Recv* are used to measure the Round Trip Time in shared memory RTT_{shm} and network RTT_{net}, and to infer o_{mw} and o'_{mw} as:

$$RTT_{shm}(m) = 2 \times o_{mw} \implies o_{mw} = \frac{RTT_{shm}(m)}{2}$$
$$RTT_{net}(m) = 2 \times \left(o'_{mw} + o'_{net}\right) \implies o'_{mw} = \frac{RTT_{net}(m)}{2} - o'_{net}$$

7.3 τ–Lop parameters

The methodology for measuring the τ–Lop parameters is introduced by Rico and Díaz in [20], and improved by Rico et al. in [23]. The authors devise an operation $Ring^c_\tau(m)$ to solve the complex issue of ensuring the contention of τ processes in the estimation of the transfer time parameter $L^c(m, \tau)$ for a range of message sizes. The operation is executed in shared memory to obtain the transfer time $L^0(m, \tau)$. Then, the operation execution in the network together with the shared memory transfer time are used to figure out the network transfer time $L^1(m, \tau)$.

A linear regression method is proposed here for improving the accuracy of the measurements of the transfer time L^c parameter. It is used for all the range of message sizes, but it is specially interesting for short messages. The short messages measurements usually have a lower accuracy, and the contention is complex to ensure, and indeed it has not a high influence in the final parameter value.

7.3.1 Overhead (o)

$o^c(m)$ is the time elapsed from the invocation of the operation until the effective data transmission starts through the channel c. The overhead time is assumed not be affected

by the contention in the channels. It includes the software stack and protocol times.

Most MPI libraries implement two protocols for message passing, with an important impact in the overhead value. They are called *eager* and *rendezvous* and perform regardless of the communication channel. They are used for short and large messages respectively. When the message size reaches a threshold size H, the send primitive switches from eager to rendezvous. The value of H is implementation dependent. For instance, in MPICH the Ethernet/TCP network protocol default threshold is $H_{TCP} = 128KB$. The protocols work as follows. Eager requires no handshake between sender and receiver, hence data is sent as soon as possible. In the rendezvous protocol, before the data transmission starts, the sender process sends a RTS (Request to Send) message to the receiver, which responds with a CTS (Clear to Send) acknowledgment message when ready to receive. Waiting for this notification avoids the sender to flood the receiver.

Our goal is to estimate the overhead in the shared memory and network channels taking into account that it depends on the protocol used. In practice, libraries such as MPICH and Open MPI do not provide noticeable difference in the overhead value in shared memory whatever be the message size. The reason is that they have a common communication mechanism for the whole range of message sizes.

The two operations shown in Figure 7.1 are used to measure the overhead parameter. Both operations are applicable to MPICH, Open MPI and AzequiaMPI libraries. In both operations, messages of size $m = 0$ are used, hence with a transfer cost of $L^c(0,1) = 0$.

The first operation is RTT^c, shown at the left side of the Figure 7.1. It is used to estimate the shared memory overhead $o^0(m)$ for the whole range of messages, and the network overhead $o^1(m)$ under the eager protocol, i.e., for message sizes $m < H_{TCP}$. τ–Lop models the cost of the RTT^c operation using a Round Trip message composed of two point-to-point message transmissions. The cost of each message transmission is the addition of the overhead and the sequence of s transfers to reach the destination, as defined in (3.4). The cost of the operation is:

$$RTT^c(0) = 2 \times \left[o^c(m) + \sum_{j=0}^{s-1} L_j^c(0,1) \right] \Rightarrow o^c(m) = \frac{RTT^c(0)}{2} \tag{7.2}$$

The $Ping^c$ operation, shown at the right side of the Figure 7.1, is used to estimate the overhead in the network $o^1(m)$ under the rendezvous protocol ($m \geq H_{TCP}$). The MPI Standard defines the *synchronous* point-to-point *MPI_Ssend* primitive, which forces the

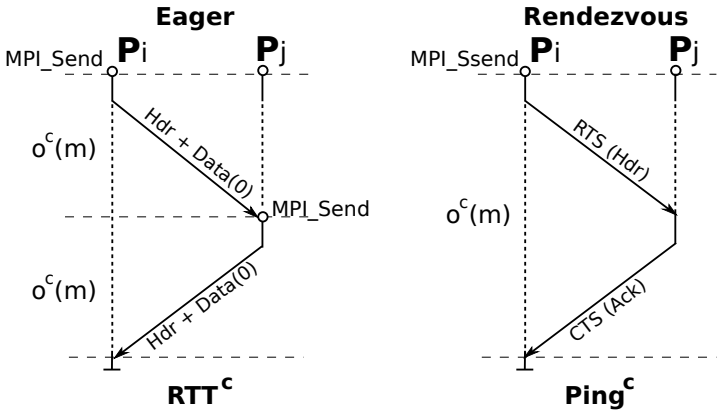


FIGURE 7.1: RTT^c and $Ping^c$ operations to measure the overhead $o^c(m)$ parameter under eager and rendezvous communication protocols respectively. Both operation can be used in shared memory and network communication channels, although, in practice, in shared memory the eager protocol is used exclusively.

use of the rendezvous protocol in a network message transmission, leading to a cost of:

$$Ping^c(0) = o^c(m) + \sum_{j=0}^{s-1} L_j^c(0,1) \Rightarrow o^c(m) = Ping^c(0) \tag{7.3}$$

A high number of iterations in both operations are executed and the average time is used.

RTT^c operation is not used for the rendezvous protocol because of the following reason. At the right side of the Figure 7.1, the RTT^c operation would add a second point-to-point response message by the process P_j. However, it can start such response message sending the RTS before the process P_i finishes the reception of the CTS. The overlapping of both messages would lead to a wrong overhead estimation.

7.3.2 Shared memory Transfer Time (L^0)

The communication between processes in shared memory progresses through shared intermediate buffers, so data needs two transfers to reach the destination buffer, as discussed in section 3.4.1.

A $Ring_\tau^0$ operation is defined as the sending and receiving of a message of size m between adjacent processes arranged in a ring of τ processes ([20, 21]). *MPI_Sendrecv*, with cost defined in (3.6) and (3.7) for short and segmented messages respectively, is used to execute the Ring operation. Every call to *MPI_Sendrecv* by process P_i entails a transmission to process P_{i+1}, and a transmission from process P_{i-1}, and then a wait operation until both complete. The wait provides a synchronization point in each transmission, which ensures that the τ processes transfer data concurrently.

When $m \leq S$, no segmentation takes place and the operation cost will be:

$$\begin{aligned} Ring_{\tau}^{0}(m) &= o^{0}(m) + 2\,L^{0}(m,\tau) \Rightarrow \\ L^{0}(m,\tau) &= \frac{Ring_{\tau}^{0}(m) - o^{0}(m)}{2} \end{aligned} \tag{7.4}$$

When $m > S$, messages are segmented and sent as a sequence of k segments of size S, with $k = m/S$. Every process sends k segments and in turn receives k segments, so the cost will be:

$$\begin{aligned} Ring_{\tau}^{0}(m) &= o^{0}(m) + 2\,k\,L^{0}(S,\tau) \Rightarrow \\ L^{0}(S,\tau) &= \frac{Ring_{\tau}^{0}(m) - o^{0}(m)}{2\,k} \end{aligned} \tag{7.5}$$

The size of a segment is $S = 8KB$ in both MPICH and Open MPI unless another value is specified. As a consequence f modeling around $L^{0}(S,\tau)$, only messages up to size S need to be measured using the $Ring_{\tau}^{0}$ operations. This is as well the case for the LogGP model, however, in $\log_n P$ the overhead parameter has to be estimated for the whole range of messages, because it is unable of capturing such a low level feature of the transmission as it is the segmentation.

7.3.3 Network Transfer Time (L^1)

As discussed in section 3.4.3, the message transmission between processes through the *Metropolis* Ethernet network consists of three intermediate transfers. The first and the last will progress through shared memory, hence with a cost already estimated in (7.4) and (7.5).

A $Ring_{\tau}^{1}$ operation is set up to measure the network transfer time L^1 ensuring concurrent access of τ processes to the channel. An example of the operation for $\tau = 3$ is shown in Figure 7.2. $2\,\tau$ processes are mapped in a Round Robin fashion in two nodes, so that even and odd processes will run in different nodes. $Ring_{\tau}^{1}$ operation has two stages. First, each even processes P_i sends to odd processes P_{i+1} a message through the network, and then each even process P_i receives from the odd processes P_{i-1}, resulting in the following cost:

$$\begin{aligned} Ring_{\tau}^{1}(m) &= 2 \times \left[o^{1}(m) + 2\,L^{0}(m,\tau) + L^{1}(m,\tau) \right] \Rightarrow \\ L^{1}(m,\tau) &= \frac{Ring_{\tau}^{1}(m)}{2} - o^{1}(m) - 2\,L^{0}(m,\tau) \end{aligned} \tag{7.6}$$

As the overlap of the copying to the NIC internal buffer and transmission through the network is unavoidable, it is not considered in the cost because of its random behavior,

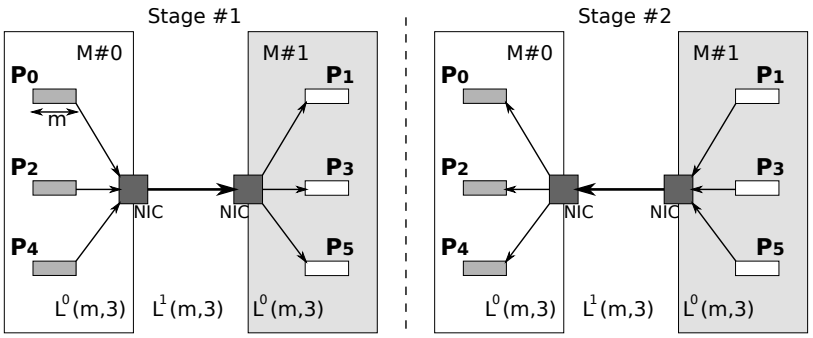


FIGURE 7.2: The $Ring_{\tau}^{1}$ operation used to estimate the $L^1(m,\tau)$ parameter for message of size m and $\tau = 3$. $2\tau = 6$ processes are mapped in Round Robin in two nodes. In the first stage, even processes send to odd processes, and in the second stage the communications revert.

as assumed in section 3.5. The message segmentation mechanism is not used for transmissions though a TCP network, so definition (7.6) can be applied for the whole range of message sizes m in MPICH, Open MPI and other libraries.

7.3.4 Computing time (γ)

As showed in the section 4.7, the computing time has a high influence in collective operations as *MPI_Reduce*. The $\gamma_{op}(m,\tau)$ parameter represents the computation time in the execution of a collective operation, and its value depends on the operation performed, the types of the operands, the size of the message and other hardware and software factors. For measuring this parameter a micro-benchmark has been developed, using as a base the IMB. It works by creating τ processes operating on arrays of size m concurrently, without communication. The operation *op* to apply is one of the MPI Standard predefined operations that can be used in the reduction collectives. Extensions to cover the user-defined operations allowed by the Standard are direct. Barrier synchronization makes all processes to start the computation at the same time, to ensure the concurrency. We ensure the computed data is out of L3 cache, hence, providing an upper bound value for the parameter. This configuration meets the usual one in the reduction operations, where the data is received out of cache at the time of applying the operation.

The concurrency of the computing affects to the performance, as showed in Figure 4.26 and Figure 4.27 for the *MPI_SUM* operation (addition) applied to arrays of increased number of float operands.

7.3.5 Packing and unpacking time (p)

Non-contiguous data are represented by a *type map* in the MPI standard. Current MPI implementations pack these data in a contiguous buffer before the transmission according to its type map and, once arrived, the buffer is unpacked, mostly under the same map. The MPI standard provides with user packing and unpacking operations as well. τ–Lop considers that the *packing* cost is attributable to the collective operation (see section 3.4), which follows the common behavior of MPI libraries such as MPICH and Open MPI.

τ–Lop assumes the error in the cost derived from the transmission of part of the message during the packing algorithm. This issue is highly dependent on the library and hard to measure in practice, and it has not been addressed in any previous work. This thesis only considers contiguous messages. Therefore, the local costs associated with packing/unpacking of non-contiguous messages are not addressed.

7.4 Improving the accuracy in the $L(m,\tau)$ estimation

The discussed $Ring^c_\tau$ operation has been used to measure the L^c parameter in each communication channel c. This method of estimating L has shown itself accurate enough in modeling the complex collectives studied in this thesis, as shown in previous sections. Nevertheless, even higher accuracy in the estimation of L^c can be achieved by applying a linear regression procedure, specially for short messages. Besides $Ring^c_\tau$, linear regression procedure can involve the measurements done for other collectives. The target L^c terms will appear now in more than one equation and the best fitting value can be obtained.

Expression (7.4) $Ring^0_\tau(m) = o^0(m) + 2\,L^0(m,\tau)$ can be put as $Ring^0_\tau(m) - o^0(m) = 2\,L^0(m,\tau)$. Moving τ between 1 and 4, for instance, we can express the method used to determine $L^0(m,\tau)$ as the trivial linear system

$$\begin{pmatrix} Ring^0_{\tau=1}(m) - o(m) \\ Ring^0_{\tau=2}(m) - o(m) \\ Ring^0_{\tau=3}(m) - o(m) \\ Ring^0_{\tau=4}(m) - o(m) \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} L^0(m,1) \\ L^0(m,2) \\ L^0(m,3) \\ L^0(m,4) \end{pmatrix}, \tag{7.7}$$

which we express in a vector form as $d_m - o_m = R_m\, l_m$. Vector d_m contains the data obtained after measuring the execution times of the $Ring^c_\tau(m)$ operations. The system has P equations and P unknowns, four in this case. Matrix R_m contains the coefficient of the $L^c(m,\tau)$ in the column of number τ.

Beyond $Ring^c_\tau(m)$ operation, we know that the Binomial broadcast operation has a cost of $\Theta^0_{bcast}(m) = 2\,o^0(m) + 2\,L^0(m,1) + 2\,L^0(m,2)$ (see section 5.2.1). It is possible to enrich the former linear system (7.7) with this equation, so that:

$$\begin{pmatrix} Ring^0_{\tau=1}(m) - o(m) \\ Ring^0_{\tau=2}(m) - o(m) \\ Ring^0_{\tau=3}(m) - o(m) \\ Ring^0_{\tau=4}(m) - o(m) \\ \Theta^0_{bcast}(m) - 2\,o(m) \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \\ 2 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} L^0(m,1) \\ L^0(m,2) \\ L^0(m,3) \\ L^0(m,4) \end{pmatrix}$$

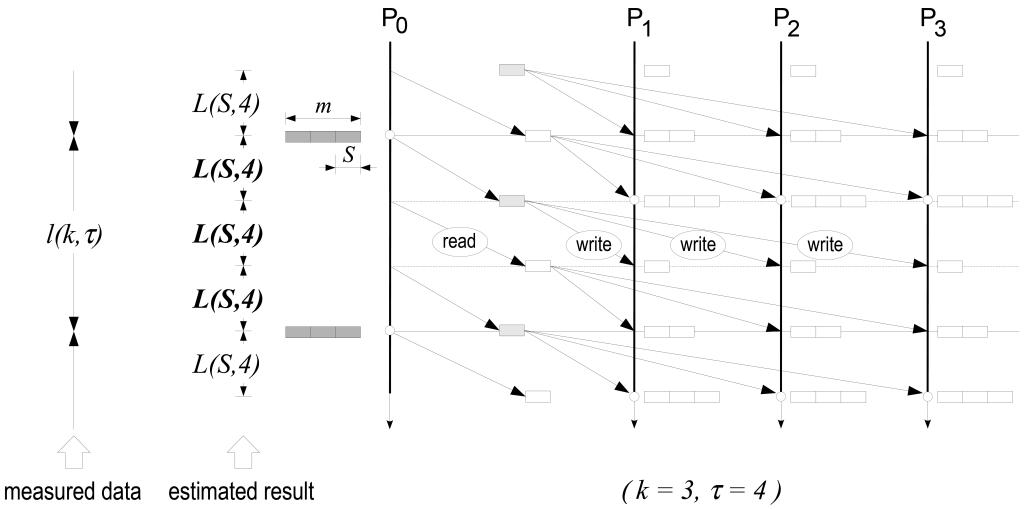
The method of ordinary least squares can be used to find an approximate solution to this overdetermined system of $P+1$ equations and P unknowns. It is well known that such solution is obtained from the problem of finding the l_m which minimizes $\|(d_m - o_m) - R_m\, l_m\|^2$.

Note that this is just an example. More and different equations can be added to the problem of estimating the parameters of τ–Lop. This methodology is mainly aimed at getting a higher level of accuracy in cost estimation of different applications, by adding the execution time of collectives actually invoked by the applications. This expected overall improvement in modeling the communication costs of applications comes hence at the expense of running benchmarks for their collectives.

7.5 On the estimation of $L^c(m,\tau)$: Further issues and description of the software tool.

This section addresses some issues which lead to inaccuracies in the estimation of the transfer time parameters. As well, we propose a tool for estimating the overhead and transfer time parameters based on the operations described in section 7.3.

Current memory systems are faster in reading than in writing. This difference is surprising, because from the perspective of the physical DRAM memory module, load and store operations take approximately the same amount of time. Writes take about twice as long as reads, because of prefetching and the way the cache is integrated into the memory subsystem. For instance, the authors' tests show that the read bandwidth overcomes the write bandwidth by a factor of 1,5 in *Lusitania*. Similarly, according to Molka et al. [72], the read bandwidth doubles the write bandwidth in the Nehalem architecture, a figure confirmed by the author's tests. This means that the cost $L(S,\tau)$ of transferring a segment from its user send buffer to the intermediate buffer (read) would



k (segments)	m (bytes)	l(k,4) (ns)	L(8k, 4) (ns)
1	8192	3793	3793
2	16384	7666	3822
...	...	...	...
8192	67108864	31205259	3809
12288	100663296	47056028	3829

FIGURE 7.3: The *broadcast* approach to the estimation of $L(S,\tau)$. The figure illustrates the case $\tau = 4$. Messages of three segments were used ($k = 3$). Note that $l(k,4)$ was measured in successive iterations (the average was taken). $L(S,4)$ was obtained from the hypothesis of $l(k,4) = kL(S,4)$, so that $L(S,4) = l(k,4)/k$. This setup was created by the command "./taulop -bcast -off_cache -local 2 8 4". For better precision, $L(S,\tau)$ was estimated for growing k (the figure illustrates the case $k = 3$), and so the final $L(S,\tau)$ figure is the average of all them. Table below shows the output of the command, with a row per each value of k. Column l in the first entry, for instance, shows the cost of broadcasting a buffer of size $m = 8$ KB (or $k = 1$). It is really the average of a loop of 262144 broadcasts, a number great enough to minimize the measurement error.

be half the cost from the intermediate buffer to the user receive buffer (write). Introducing these hardware dependent issues in τ–Lop would make the model over-detailed, and hence difficult to understand and use. However, keeping τ–Lop simple entails a degree of indetermination in the characterization of $L(S,\tau)$.

A suitable experimental design is essential to perform any parameter estimation, and to this end the software tool *tauLop* [73] was developed. It produces a variety of concurrent transfer setups, measures their performance and calculates $L(m,\tau)$ from the obtained data. Each adopted setup, however, leads to different estimations, and so three of them are discussed next. To guide the exposition, the goal is the determination of $L^0(8KB, 4)$.

The first setup is based on Figure 7.3, a broadcasting between τ processes ($\tau = 4$). *tauLop* creates this specific scenery by invoking the command *¨$ ./tauLop -bcast -off_cache -local 2 8 4¨*. One needs to note the *-bcast* parameter. Parameter *-local* binds processes to cores, so that P_0, acting as root, is bound to core 0 of NUMA 0.

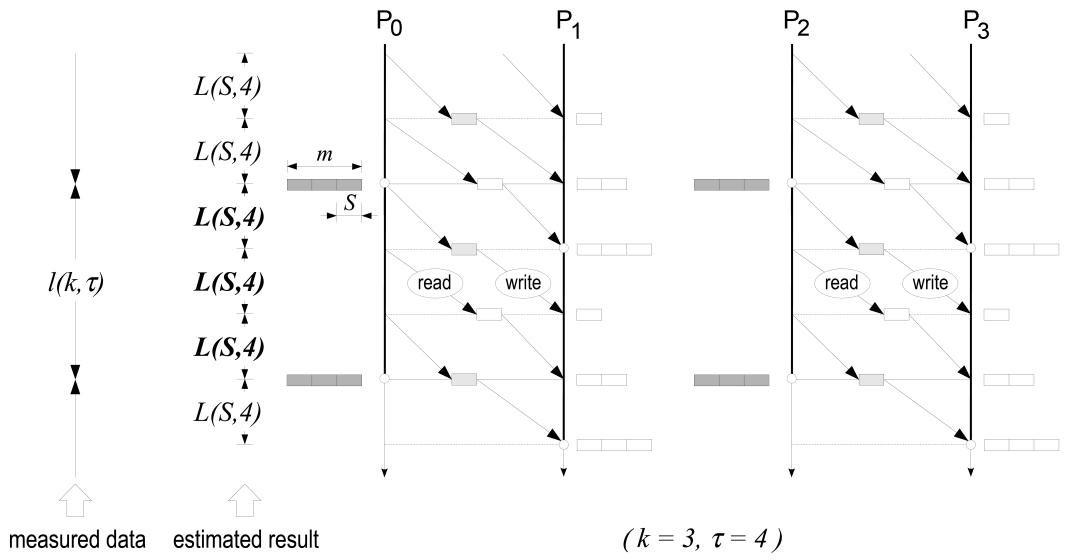


FIGURE 7.4: The *ping* approach to the estimation of $L(S,4)$.

Non-root processes first fill the rest of the cores of NUMA 0, and then the NUMAs 1, 2, 3, etc. The pair of parameters *[2,8]* after *-local* means that the transfers are performed through an intermediate buffer of two segments, each of size $S = 8$ KB. The last parameter 4 starts four threads. Finally, the parameter *-off_cache* keeps the send and receive user buffers out of cache, with the goal of determining L as the cost of a transfer between main memory (the user buffer) and L3 cache (the intermediate segments). Both send and intermediate buffers are allocated in the NUMA 0. Each receiving buffer was allocated to the NUMA where its corresponding process runs. The key issue of this setup is that Figure 7.3 assumes that $l(k,\tau) = kL(S,\tau)$, so that $L(S,\tau) = l(k,\tau)/k$. This broadcast scenery should perform as an upper bound on the estimation of $L(S,\tau)$. The $\tau-1$ simultaneous copies from the shared intermediate segments in NUMA 0 to the $\tau-1$ receiving user buffers in NUMA 0, 1, 2, etc, end up exhausting the bandwidth of NUMA 0. Effectively, $L(S,\tau)$ grows linearly with τ having a steep slope. It is to be noted that this broadcast technique is inefficient and hence not adopted for collective operations in current MPI implementations, that opt for hierarchical arrangements[2]. In other words, it characterizes the behavior of τ–Lop under these rather extreme conditions.

The second approach to estimate $L(S,\tau)$ was based on a pair of concurrent message transmissions ($\tau = 4$) (see Figure 7.4), started by the command *¨$ ./tauLop -ping -off_cache -local 2 8 4¨*. This time, the figure states that $l(k,\tau) = kL(S,\tau)$ for every k, so that $L(S,\tau) = l(k,\tau)/k$. The couples of processes first fill the NUMA 0 and then populate successive NUMA nodes. All buffers were allocated to the NUMA where their corresponding couple runs. On this occasion, no inter-NUMA communication takes

[2]The broadcast of Figure 5.2 is an example. Let it be supposed that processes 0 to 7 run in the NUMA 0 and processes 8 to 15 in NUMA 1. Once the inter-NUMA message transmission of stage 1 is done, both NUMAs work in parallel with just internal communication.

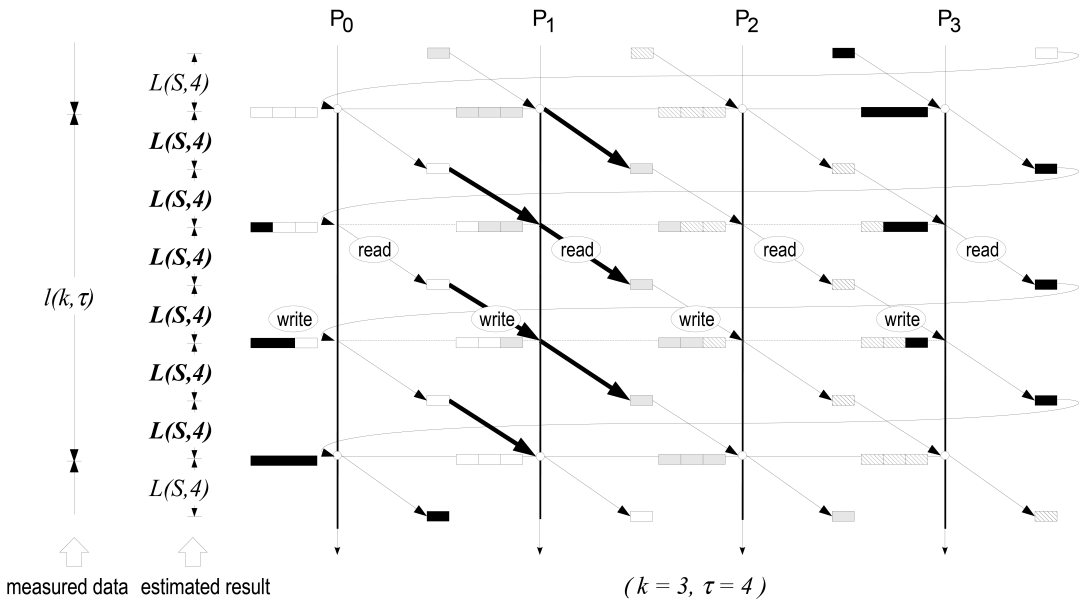


FIGURE 7.5: The *ring* approach to the estimation of $L(S, 4)$.

place, again an unrealistic situation, this time at the opposite extreme to $-bcast$. Another problem with *-ping* arrangement is that it lacks synchronization between couples. As a result, it becomes impossible to assure that couples run synchronized along the time frame of Figure 7.4, and thus the measurements are invalidated.

The third configuration was based on a ring of four processes ($\tau = 4$) created by the command *¨$ ./tauLop -ring -off_cache -local 2 8 4¨* (Figure 7.5). Note the parameter $-ring$. The figure states that $l(k, \tau) = 2kL(S, \tau)$ for every k, so that $L(S, \tau) = l(k, \tau)/2k$. Synchronization on the access to main memory is ensured by each pulsation of the ring: a transfer from the user buffer to the intermediate segment shared with the right neighbor, followed by a second transfer from the intermediate segment shared with the left neighbor to the user segment just copied out.

The foregoing discussion on the application of the *taulop* tool, for instance, has generated three different versions of $L(S, \tau)$, and many more are possible; so, which of them is the best of all? The *-ring* approach offers a good mix of intra- and inter-NUMA communication and reliability of measurements, and so the authors chose that method to obtain $L(S, \tau)$. Regardless of the present results, it is evident that each choice favors the accuracy of some algorithms to the detriment of others. For example, options $-bcast$ and $-ping$ could be useful in particular collective algorithms. It was detected here that $-bcast$ results achieve better accuracy on modeling the cost of Open MPI SM broadcast, because that algorithm is built upon the same type of collective transfers used in τ–Lop. However, $-ring$ configuration was used throughout this presentation. Figure 7.6 shows the estimations of $L^0(8KB, \tau)$ obtained in the target platforms.

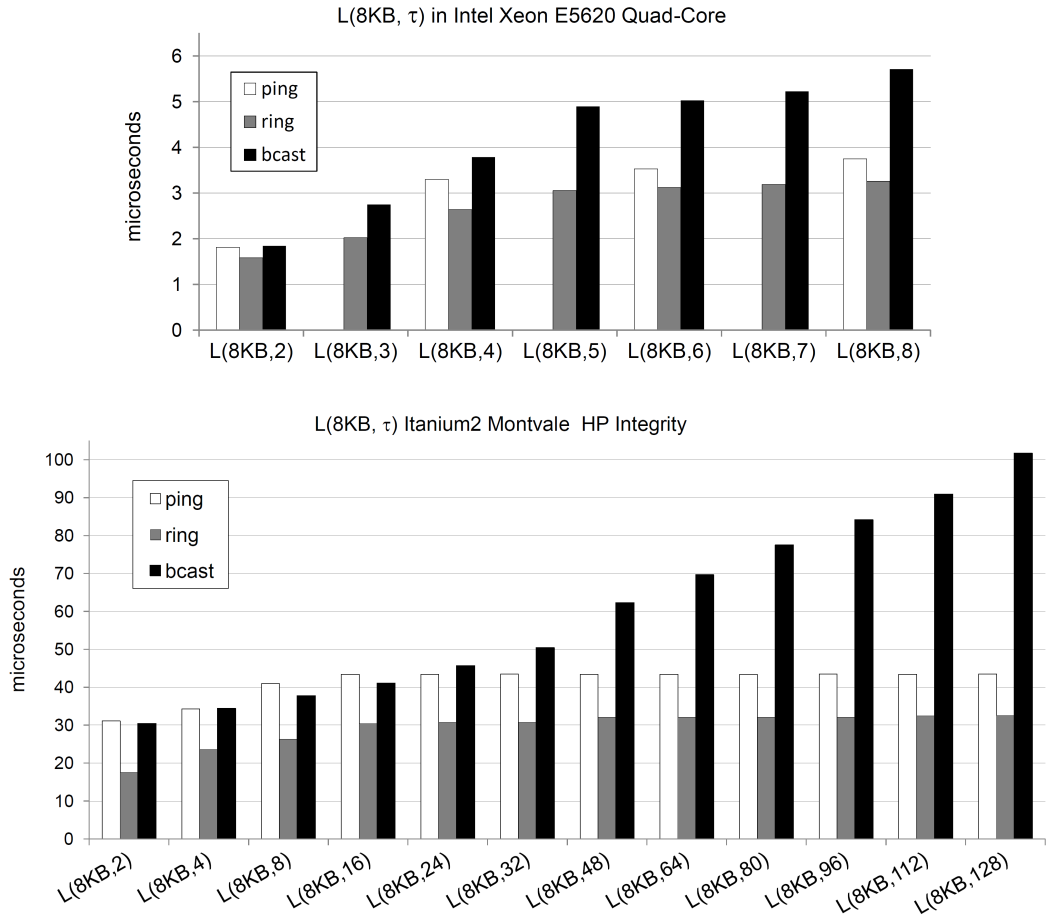


FIGURE 7.6: Estimations of $L(S, \tau)$ in *Metropolis* and *Lusitania* machines under three considered approaches.

7.6 Comparing the models with the Proportional Error μ

The utility of a model is determined largely by the accuracy achieved in the estimation of its parameters, which results in the correct prediction of the cost of the operations. Several error metrics exist to express the accuracy of an estimation with respect to the real measurement. Usually, *Relative error* is used in the literature, based on the distance between estimate and real values. Lets call ρ to the Relative error of an estimated value e with respect to the real value r. It is defined [74] as:

$$\rho = \frac{|e - r|}{r} \tag{7.8}$$

The concept of relative error as an expression of error suffers an anomaly outlined hereafter. Figure 7.7 illustrates this point with an example. The first estimated value is e, with a Relative error with respect to the measured value r of $\rho = 1$. For the estimated

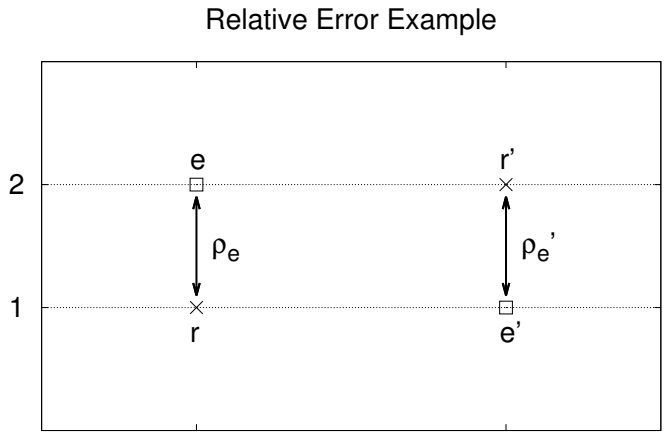


FIGURE 7.7: Example of the *Relative error* ρ of two estimated values with respect to their respective real measurements.

value e', the Relative error with respect to r' is $\rho' = 0.5$. Therefore, $\rho \gg \rho'$, even though the distance and proportion among estimated and real values are equal.

This fact has an impact in the communication performance modeling evaluation. A model that underestimates the cost of an algorithm will give a lower Relative error than a model that overestimates it in the same proportion. For this reason this thesis discards the Relative error and uses an error measurement based on the proportion between real and estimated values, which reflects the accuracy of the measures in a more meaningful way. We define the *Proportional error* μ of a estimated value e with respect to the real value r as:

$$\mu = \frac{max(r, e)}{min(r, e)} \tag{7.9}$$

Under the hypothesis that both e and r are positive numbers, the Proportional error is always greater than 1, equal when there is no error. Regarding the example in Figure 7.7, Proportional errors will be identical ($\mu = \mu' = 2$) because their predictions fail in the same proportion.

In this thesis, the Proportional error is applied to sets of estimated values obtained from different models in order to compare them, and with the real values. Being times, real and estimated values are positive numbers. The procedure is as follows. In starting from two sets of discrete values R and E^M, where R represents the measured communication times, and E^M represents the estimated values by a model M for an operation:

$$\begin{aligned} R &= \{r_0, r_1, r_2, ..., r_{n-1}\} \\ E^M &= \{e_0^M, e_1^M, e_2^M, ..., e_{n-1}^M\} \end{aligned} \tag{7.10}$$

That is, r_i is the measured value for the message size i, and e_i^M is the estimated value for the message size i by the model M of an operation. The proportion between the set of measured and estimated values is defined as:

$$\mu^M = \{\mu_0^M, \mu_1^M, \mu_2^M, ..., \mu_{n-1}^M\},$$

with each component μ_i^M as the Proportional error of the estimated value with respect to the real measurement for the message size i, defined in (7.9). However, neither the Proportional nor the Relative errors give information about which range of values (R or E^M) is higher, and it needs to be provided in each context.

The Average Proportional error of the model M for the whole range of message sizes in the set μ^M is:

$$\bar{\mu}^M = \frac{\sum_{i=0}^{n-1} \mu_i^M}{n} \tag{7.11}$$

There is a simple relation between Relative and Proportional error that allows the conversion from one to the other. Lets us consider the Relative error ρ defined in (7.8) of a single estimated value e with respect to the real value r, and the Proportional error μ defined in (7.9) for the same pair. We have two cases: $e > r$ and $e < r$. Note that when $e = r$ no error applies, that is, the Relative error is 0 and the Proportional error is 1.

1. If $e > r$, then $\mu = e/r$, and then:

$$\rho = \frac{e - r}{r} = \frac{(\mu\, r) - r}{r} = \mu - 1 \tag{7.12}$$

   And then:

$$\mu = \rho + 1 \tag{7.13}$$

2. If $e < r$, then $\mu = r/e$, and then:

$$\rho = \frac{r - e}{r} = \frac{r - (r/\mu)}{r} = 1 - \frac{1}{\mu} \tag{7.14}$$

   So that:

$$\mu = \frac{1}{1 - \rho} \tag{7.15}$$

7.7 Conclusions

A simple methodology for estimating the τ–Lop parameters has been proposed. The overhead is estimated in the shared memory and network channels taking into account

the protocol time. We defined two simple operations for estimating the overhead under the two widely used protocols in MPI communications, eager and rendezvous. The estimation of the transfer time parameter is solved through the definition of a *Ring* operation, that ensures the synchronization, and hence the contention of the processes in the access to the channel. This operation can be executed in shared communication channels as shared memory, as well as in multi-core clusters. *Ring* provides with a reasonable good accuracy, that can be improved by a simple linear regression procedure described in the chapter. This procedure is specially indicated for predicting the cost of algorithms communicating short messages, because of the complexity in ensuring the concurrency in the measurement of the transfer time. In addition, in an application with a set of known collective algorithms, the linear regression procedure can be enriched with the execution times of these algorithms for improving the accuracy of the communication cost prediction. The computation time parameter is estimated through a micro-benchmark as an upper bound of the time of concurrent processes operating arrays of values of different datatypes.

Finally, the definition of the *Proportional error* provides with more meaningful information about the accuracy of the predictions than the broadly used Relative error. We show that the Proportional error does no change when a model underestimates or overestimates the real values in the same proportion. Besides, conversion equations are provided between both errors.

Chapter 8

Conclusions and future work

Modern HPC platforms demand better communication performance models able of capturing their growing complexity, in particular the heterogeneity of the communication channels and the increasing number of cores per node. This work proposes τ–Lop, a formal model that predicts the cost and helps in the optimization of the communications in such systems, with focus on the collective operations defined in the MPI standard. The limitations of the precedent models in the multi-core arena have been identified and the way they are overcome with τ–Lop demonstrated.

τ–Lop provides with two key contributions with respect to the previous stand out. First, τ–Lop is able to capture the impact of the bandwidth shrink in shared channels when several transmissions progress in parallel, with the overall result of improving the accuracy of the cost estimation. Indeed, this work reveals that overlooking the representation of the contention leads to unacceptable estimation errors. τ–Lop is able to represent such concurrent transmissions through a simple but still powerful notation. As an example, it has been formally demonstrated that the Binomial Scatter algorithm is at least twice cheaper than Recursive Doubling Allgather, while LogGP model assigns them equal cost. In practice, the programmers of current MPI libraries often choose between collective algorithms based on these wrong predictions.

The second contribution of τ–Lop is the modeling of the influence of the process mapping on the cost of an algorithm, due to a hierarchy of communications based on channels of uneven performance. A given mapping determines the channels used by a collective algorithm in the multi-core cluster, and under-representing it in a model leads to wrong cost predictions. For instance, it has been shown that the cost of the Ring algorithm in a small cluster could vary up to six-fold with opposite mappings. Previous models do not capture correctly the cost of the mapping because such cost is affected by the

contention, in addition to the communication channels used by the processes executing the algorithm.

Algorithm analysis under τ–Lop is compared to other representative models in shared memory (*Lusitania*) and in multi-core hierarchical (*Meropolis*) platforms. LogGP and its extension LogGPH models are representative of the broadly used postal models which use network-related latency and bandwidth parameters, and estimate the cost of an algorithm as the cost of the longest path of the involved point-to-point transmission. $\log_n P$ and its extension $m\log_n P$ models change the approach of the point-to-point transmissions models using middleware-related parameters and introduce a decomposition on transfers through the communication channels hierarchy. However, LogGP and $\log_n P$ main limitation is the modeling of algorithms as the largest path for a process transmissions.

In the author's view, τ–Lop offers a more faithful picture of an algorithm behavior based on the concept of *concurrent transfers* and a simple but still powerful notation for representing their cost. Indeed, the expressive power of τ–Lop makes possible the representation of the sophisticated communication techniques underlying current implementations of MPI in shared memory, such as collective transfers or message segmentation. As a result, τ–Lop offers a higher analysis capacity, resulting in a more scalable and accurate cost prediction. The accuracy of τ–Lop was evaluated by applying it to a diversity of *MPICH* and *Open MPI* algorithms. The algorithms discussed, being fairly common, have been chosen because their complexity significantly exceeds that of the algorithms evaluated in previous related works, and they cover a wide spectrum of the known collective communication techniques, specially in shared memory.

Finally, an exhaustive methodology for estimating the τ–Lop parameters is proposed. It is based in a *Ring* operation ensuring the processes contention in the access to the channel for measuring the *transfer time*. The accuracy in the estimation of this parameter can be improved by a linear regression procedure, specially in short messages, where contention and accuracy in the measurements is harder to achieve, due to the fact that synchronization costs are significant enough to be ignored. The *overhead* parameter is estimated using two defined operations, which are used for each combination of a communication channel and protocol. The *tauLop* software for parameter estimation based on the former operations is just a seminal work on this issue, which needs further contributions, application to other platforms and, most important, new insights.

The *Proportional error* is formally defined and it is used in place of the more common *Relative error* in the comparison of the algorithm costs. In the author's view, it provides with more meaningful information on the error derived from the estimation of the models.

As well, the conversion of the Proportional error to the Relative error and vice-versa is provided.

Current work focus on the prediction of the cost of communications in scientific hybrid applications on heterogeneous platforms. High performance heterogeneous computing platforms include processing elements with different missions, for instance, a combination of multi-core CPU and GPUs. The performance goal in this kind of platforms is maximize the computing load balancing. However, the communication cost optimization in this kind of platforms is addressed experimentally. τ–Lop has been applied in this field to facilitate the optimization of the communications of the, now widespread, hybrid applications running on heterogeneous platforms.

Appendix A

Estimation of the communication cost of hybrid applications in heterogeneous platforms

Ensuring the efficient increase in the performance of modern HPC systems, in terms of both cost and energy consumption, is the main cause for high performance computing becoming heterogeneous. The systems composed of nodes with multi-core processors and accelerators, and communicating through uneven performance communication channels, are mainstream. Scientific applications running on these heterogeneous computing platforms usually include a set of *kernels*. A kernel is a representative piece of code inside the application, as matrix multiplication and Fourier transform.

A *hybrid application* is composed of kernels targeted for being executed by a set of processes running on processors of different types. Hence, each process has different performance capabilities. Achieving the optimal performance of such hybrid scientific applications requires to unevenly distribute each kernel workload between the processes. The optimization goal is the overall computational load balancing, avoiding faster processes to wait for slower ones at synchronization or communication points. Such uneven distribution of a kernel workload affects to the global communication performance. The reason is that it introduces imbalances in the communication, because each process has to communicate a different volume of data. A holistic formal approach to the performance optimization in heterogeneous platforms has not attracted much attention in the literature, probably due to its complexity.

In practice, specially for large multi-kernel applications, the communication optimization is usually carried out through a set of thorough tests of a shortened version of the

application (for instance, individual kernels) on a subset of the target platform. This approach has two main drawbacks: the first is the use of quite expensive computational resources for testing, and the second is that often it is not possible to correctly extrapolate estimations from such simplification of the application and the platform. We introduce a model-based methodology for the estimation of the communication cost of hybrid applications on heterogeneous platforms. The costs are derived from, first, a given distribution of the workload to the processes, and second, the process mapping on the platform, which decides the channels used for their communication. This analytical approach, based on the τ–Lop model, offers the advantage of drastically reducing the experimental use of the resources and contributes to clarify hidden issues in the communications.

The rest of the appendix is organized as follows. Section A.1 reviews some of the approaches to the performance optimization of applications in heterogeneous platforms, focusing on computational load balancing. Section A.2 provides with assumptions in τ–Lop in order to model communication in heterogeneous platforms. Section A.3 introduces the SUMMA matrix multiplication algorithm, the kernel evaluated on the platform described in the section A.4. Section A.5 performs the communication modeling in two different layouts of processes and compares both configurations. Section A.6 proposes mapping related optimizations. Finally, section A.7 concludes.

A.1 Related work on performance optimization on heterogeneous platforms

Two main approaches face the problem of optimizing the performance of applications on heterogeneous platforms. The first is to characterize the application as a graph and apply techniques of *graph partitioning*. The second is to characterize the application processes with a performance model and to distribute the computation among the processes in proportion to their performance.

Graph partitioning is a technique extensively used to achieve computation and communication optimization. The application is modeled as a graph composed of a set of vertices representing the computation, and a set of edges representing the communication dependencies between vertices. Both vertices and edges can be labeled with a weight. The graph is partitioned in subsets of the same amount of vertices. The computation in each subset is assigned to a process. The partitioning objective is minimizing the number of edges connecting the vertices subsets, and hence, minimizing the communication. Indeed, alternative partition objectives can be introduced. For instance, minimizing the

volume of the communication in graph with weighted edges, or the number of messages. As well, the partitioning of the vertices in subsets can be non-homogeneous, using a pre-specified different fraction of the total number of vertices (or weights) per vertices subset.

Methods to accomplish the partition problem are generally classified in *combinational* [84, 85], *spectral* [86–88], *multilevel* [89–91] and *geometric* [92, 93], with an extensive number of algorithms in the literature. A problem of the graph partitioning methods is that computation and communication are represented as a constant (weights in the graph). The user is responsible of in advance providing the computation and communication weights. Aubanel and Wu [94] propose to incorporate the latency cost in addition to the transmission rate usually characterizing the communication volume in the graph edges. There exist several tools implementing graph partition algorithms as METIS [95, 96], JOSTLE [97, 98] and SCOTCH [99, 100].

In addition to the graph partitioning problem, it is the mapping of the resultant graph to the underlying non-homogeneous platform to achieve optimal performance communications. Pellegrini discusses different methods for solving this problem in [101]. Related to the mapping problem, the matching of the communication pattern of an application to the underlying hierarchical network represented as a tree is accomplished by the Jeannot et al. *TreeMatch* algorithm [102].

The second approach faces the problem of the distribution of the application workload among the processes on the heterogeneous platform. The computational load balancing is formulated as a *partitioning* problem [70]. It departs from n independent *computational units* of equal size. A computational unit is kernel dependent. In a matrix multiplication, for instance, the computational unit can be the computation of an element in the resultant matrix. The goal is to distribute these n computational units among a set of p ($p < n$) processes[1] $P = \{P_0, P_1, \ldots, P_{p-1}\}$, in a way that the workload is best balanced. The processes are characterized by $S = \{s_0, s_1, \ldots, s_{p-1}\}$, where s_i is the *speed* of the process P_i, a positive constant indicating the number of computational units it performs by time unit. Such modeling of the performance of a process is referred as *Constant Performance Model*.

Lets suppose $N = \{n_0, n_1, \ldots, n_{p-1}\}$, with n_i the number of computational units assigned to the process P_i. Each process P_i has an execution time $t_i = \frac{n_i}{s_i}$. The execution time of the application is given by the slower process as $\max_{i=0}^{p-1} t_i$. An optimal workload distribution minimizes this expression. Beaumont et al [75] develops an algorithm for

[1] Although usually the problem is defined in terms of processors, we define it in terms of processes, which can be single or multi-threaded.

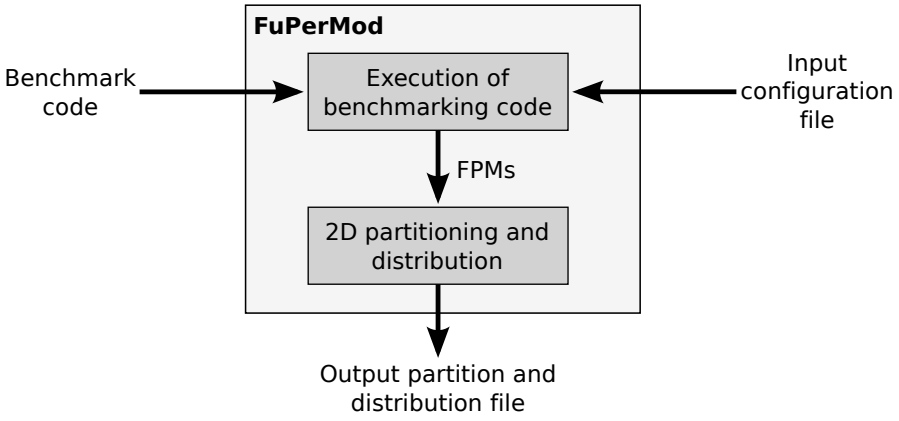


FIGURE A.1: Outline of the FuPerMod procedure for 2D partitioning and distribution of workload between processes in an application. The inputs to FuPerMod are the configuration file (layout and type of the processes) and the benchmark code. The output is the file containing the partition and distribution of the data for the processes.

achieving optimal load balancing using a Constant Performance Model and applies it to the example of matrix multiplication and the ScaLAPACK library [76].

A key point is to determine the speed of a process. It depends on several factors, including physical and software, as the memory size and hierarchy, processor type, size of the application, processor clock rate or even the operating system and its paging policy. These factors overwhelm the representation of the speed as a constant. For instance, a fast process can become slower if the amount of the data it processes exceeds the size of the cache memory. Lastovetsky and Reddy [77, 78] propose a *Functional Performance Model* (FPM), that is the speed of a process represented by a continuous function of the *task size*. The task size is defined as one or more application specific parameters characterizing the amount and layout of the data stored and computed by the processes during the execution of a kernel. For instance, regarding the matrix multiplication of two square matrices of size $x \times x$, the parameter characterizing the task size is x, which defines the amount of data stored as $3 \times x^2$ and the amount of operations to do as $\big(x + (x-1)\big) \times x^2 \approx 2\,x^3$.

The speed of a process P_i is represented as a real function of the task size $s_i(x)$, and hence the speed of a process depends on the size of the problem to solve. The partition problem is reformulated with the speeds of the p processes as $S = \{s_0(x), s_1(x), \ldots, s_{p-1}(x)\}$. The algorithm discussed by Lastovetsky and Reddy in [77] achieves $\frac{n_i}{s_i(x)} \approx const$. This means that the algorithm distributes the computational units among the processes in proportion of their speeds for an specific task size x, i..e., an optimal load balancing.

FuPerMod [79, 80] is a software tool that implements the whole procedure of workload partition and distribution for a set of processes running on a heterogeneous platform. Figure A.1 shows the FuPerMod procedure. First, FuPerMod generates the per-process

Functional Performance Model[2] executing a benchmarking code[3] provided by the user. The user also provides with a description of the layout of the processes and their assigned resources in the platform. Afterwards, FuPerMod performs the kernel data partitioning and assignation to the processes [81–83] following a geometrical approach [77]. For instance, following with the matrix multiplication example, a 2D partitioning of the matrices is generated from the FPMs of the processes.

Departing from a specific layout of processes with their workload assigned, a formal model can be used for estimating the cost of the application in heterogeneous platforms. Chan et al. [103] propose a specific performance model that predicts execution times of iterative mesh-based applications running on heterogeneous multi-core clusters. Ogata et al. [104] propose a model for the optimization of a FFT library on a GPU/CPU heterogeneous platform. The performance model divides the FFT computation into subsets and predicts the execution time of each step for a particular GPU/CPU combination of the workload distribution. García et al. [105] propose a model for the partition and evaluation of different schedules of the kernel composing and application in order to maximize the resource exploitation and reduce the global execution time. More generic communication performance models as LMO [53] are specifically designed for heterogeneous platforms (see section 2.3.3), although other models as τ–Lop can be adapted to accomplish this task, as we will see in the next section.

The communication performance modeling using τ–Lop along this appendix departs from a balanced workload achieved using the FuPerMod tool. The goal of this appendix is to evaluate the impact in the communication cost of different workload distributions and layouts of a set of processes running a kernel in a platform, minimizing the testing time.

A.2 Extensions of τ–Lop for heterogeneous communication modeling

As mentioned above, our methodology estimates the communication cost of hybrid applications on heterogeneous platforms. It addresses data parallel applications[4] which execute by repeating two stages: computation and communication. The communication modeling using τ–Lop departs from the assumption that all processes arrive at the same time to the communication stage (this requires to balance the computation stage using

[2] A table of real values representing the speed of the process for the different task sizes.

[3] The FPM reflects the speed of a process in executing a particular kernel, thus the benchmarking code has to be as similar to the kernel code as possible.

[4] Applications which allow the parallel processing of the data distributed among a set of processes.

a partitioning mechanism as a Functional Performance Model). τ–Lop is then used to predict the communication cost for the specific workload distribution and the layout of the processes in the platform[5].

τ–Lop was initially conceived to model the cost of collective operations executing on systems with hierarchical organization of the communication channels. It provides cost expressions in the forms $n \parallel T^c(m)$ and $T^c(m_1) + T^c(m_2)$, where $T(m)$ is the cost of a point-to-point message transmission between two processes. The first represents the cost of n concurrent transmissions of a message of size m through a communication channel c. The second represents the cost of a sequence of two transmissions of different message sizes through the same communication channel. More complex cost expressions, as those in the form $T^{c_1}(m_1) \parallel T^{c_2}(m_2)$ for any c_1 and c_2 communication channels, appear commonly in the modeling of communications in heterogeneous environments. Though τ–Lop allows to further subdivide a message transmission into smaller units known as transfers, the cost expressions are discussed at transmission level to simplify the exposition and focus on the main ideas. The decomposition of the transmissions in individual transfers is required in some modeling situations, at the expense of increasing the modeling complexity. Following, we extend τ–Lop with a set of assumptions to cover the evaluation of this type of expressions with the purpose of adapting the model to heterogeneous environments.

A1 A sequence of transmissions through the same communication channel has the cost of a transmission of the sum of the message sizes.

$$T^c(m_1) + T^c(m_2) + T^c(m_3) = T^c(m_1 + m_2 + m_3) \tag{A.1}$$

This is an extension of the transfers *linearity principle* (see section 3.3) applied to transmissions. The modeling assumes the error of disregarding the overhead costs of the added transmissions.

A2 Two message transmissions through the same communication channel progress concurrently during the transmission time of the shorter:

$$T^c(m_1) \parallel T^c(m_2) = 2 \parallel T^c(m_1) + T^c(m_2 - m_1), m_2 \geq m_1 \tag{A.2}$$

A3 Two transmissions progressing through different communication channels do not interfere. The total cost is the maximum of their costs:

$$T^{c_1}(m_1) \parallel T^{c_2}(m_2) = max\{T^{c_1}(m_1), T^{c_2}(m_2)\} \tag{A.3}$$

[5]We use the term *configuration* for the combination of a workload distribution and a layout of processes in a heterogeneous platform along this appendix.

A4 A process sends two messages sequentially, with a total cost of:

$$T^{c_1}(m_1) + T^{c_2}(m_2) \tag{A.4}$$

If $c_1 = c_2$, $A1$ can be applied. In the reception, there are not assumptions about sequentiality, due to the capabilities of the hardware technology, for instance, using remote memory access.

Derived from the previous assumptions, complex expressions usually appearing in the modeling procedure are transformed as

$$\begin{aligned} &\left[T^{c_0}(m_a) + T^{c_1}(m_b)\right] \parallel \left[T^{c_0}(m_c) + T^{c_1}(m_d)\right] = \\ &\left[T^{c_0}(m_a) \parallel T^{c_0}(m_c)\right] + \left[T^{c_1}(m_b) \parallel T^{c_1}(m_d)\right], \end{aligned} \tag{A.5}$$

i.e., two concurrent sequences of transmissions through different communication channels have the cost of the sequence of transmissions contending in the communication channels. Then, assumption $A2$ can be applied to simplify each resulting sequence of transmissions. As well,

$$\begin{aligned} &\left[T^{c_0}(m_a) \parallel T^{c_1}(m_b)\right] + \left[T^{c_0}(m_c) \parallel T^{c_1}(m_d)\right] = \\ &max\{T^{c_0}(m_a), T^{c_1}(m_b)\} + max\{T^{c_0}(m_c), T^{c_1}(m_d)\}, \end{aligned} \tag{A.6}$$

i.e., a sequence of concurrent transmissions through different channels has the cost of the addition of the maximum cost of the transmissions, hence applying assumption $A3$.

A.3 The SUMMA algorithm: a matrix multiplication kernel

The Scalable Universal Matrix Multiplication Algorithm (SUMMA) [106] is a state-of-the-art computational kernel which is present in many scientific applications. It can be found, for example, in the numerical linear algebra ScaLAPACK library. We discuss it here as a prototype of other kernels widely used in HPC applications.

The SUMMA algorithm computes the dense matrix multiplication $C = A \times B$. For simplicity, square matrices are supposed. Elements of the matrices are grouped in blocks of size $b \times b$ elements, and a block is the unit of computation and communication. The granularity of the block size is dependent on the system and it is adjusted prior to the multiplication. It remains constant for the whole algorithm execution. As a result, the size of the matrices is $N \times N$ blocks.

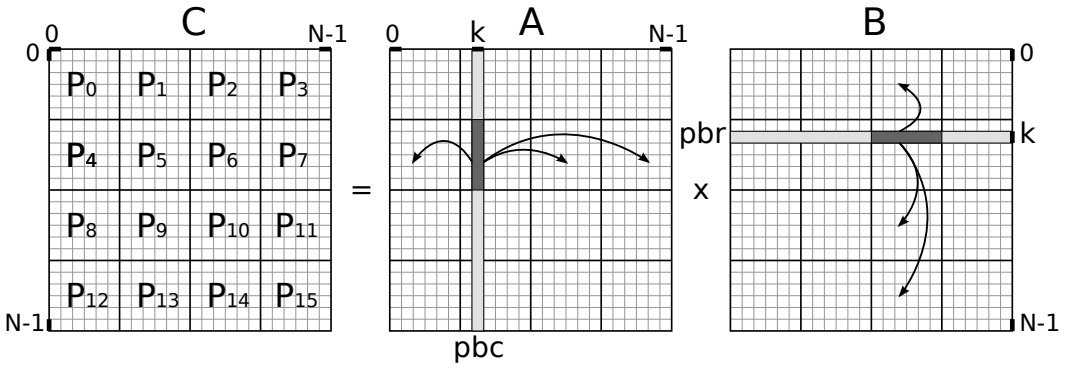


FIGURE A.2: Matrix multiplication with the SUMMA algorithm in a homogeneous platform. The number of processes is $P = 16$ in a grid layout. The matrix size is $N \times N$ blocks, with $N = 24$. A rectangle of 6×6 blocks is assigned to each process (elements are not showed). The figure shows the $k-th$ iteration in the execution of the algorithm. Processes with blocks in the pivot block column (pbc) of matrix A (P_1, P_5, P_9 and P_{13}) and pivot block row (pbr) of matrix B (P_4, P_5, P_6 and P_7) transmit their blocks to the rest of processes. For instance, in the iteration showed k, process P_5 sends its part of the pbc to the processes in the same row (P_4, P_6 and P_7), and process P_6 sends its part of the pbr to the processes in the same column (P_2, P_{10} and P_{14}).

Let us suppose a homogeneous system. The blocks are distributed evenly between the processes following a 2D arrangement. Each process is assigned with a rectangle of blocks of the same size of each matrix. Figure A.2 shows an example for $P = 16$ processes arranged in a grid. The size of the matrices is $N \times N$, with $N = 24$ blocks. Each process is assigned with a rectangle of 6×6 blocks of each matrix. This assignation balances the computational load, and also the communication volume of each process.

The SUMMA algorithm executes in N number of iterations. In each iteration k, the processes compute partial results for all of its assigned blocks of the matrix C. The process owning the block c_{ij}, computes in the iteration k $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$. As a consequence, the process has to receive the $k-th$ column block of the matrix A and the $k-th$ row block of the matrix B. After N iterations, each block will have the value $c_{ij} = \sum_{k=0}^{N-1} a_{ik} \times b_{kj}$.

Thus, each iteration k is composed of 3 stages:

1. The processes owning the k-th pivot block column (pbc) of the matrix A send the blocks to the processes in the same row. A broadcast operation on a per-row communicator can be used for the communication. The processes owning the blocks in the pbc are the roots of the broadcasts in each row.

2. The processes owning the k-th pivot block row (pbr) of the matrix B send the blocks to the processes in the same column. A broadcast operation on a per-column communicator can be used for the communication. The processes owning the blocks in the pbr are the roots of the broadcasts in each column.

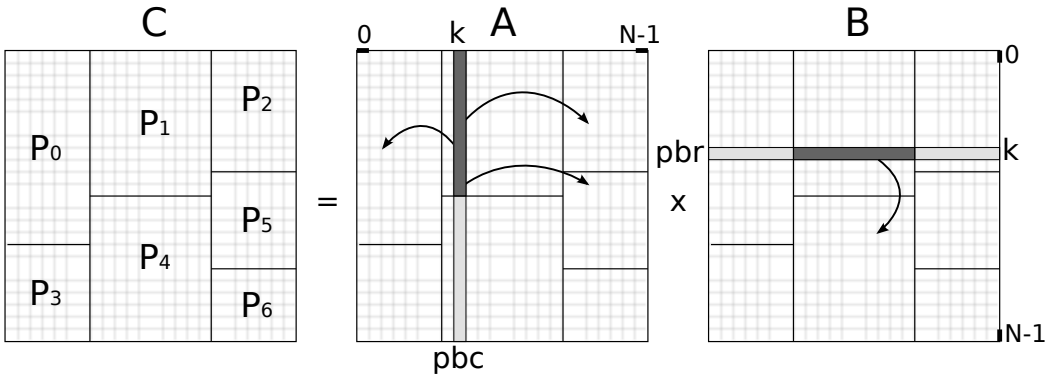


FIGURE A.3: Matrix multiplication with the SUMMA algorithm on a heterogeneous platform. The number of processes is 7. A rectangle of different size is assigned to each process. The figure shows an iteration in the execution of the algorithm. *Pbc* in the matrix A and *pbr* in the matrix B are transmitted to the rest of processes. For instance, process P_1 sends its part of the *pbc* to the processes in the same row (P_0, P_2 and P_5), and then it sends its part of the *pbr* to the processes in the same column (P_4).

3. P_i updates the blocks in its assigned rectangle of the matrix C.

The communication performance could be improved using non-blocking communication operations, allowing the overlapping of communication and computation. In addition, other improvements have been proposed, as the HSUMMA (Hierarchical SUMMA) algorithm by Hasanov et al. [107], which proposes a hierarchical approach for the algorithm communications.

In heterogeneous platforms, the number of blocks assigned to each process depends on its speed. Hence, the size of the rectangles assigned to the processes is not homogeneous, and the communication pattern changes. Figure A.3 shows an example. Following the 2D matrix geometrical partitioning algorithm ([108]), processes are arranged in columns of the same width. The communication of the *pbc* is achieved using point-to-point communication. Broadcasting is not possible because the amount of blocks sent to each processes in the same row is different. Nevertheless, the communication of the *pbr* can be achieved using broadcasts.

A.4 The heterogeneous test platform

Following, the communication cost of the SUMMA algorithm is evaluated on a heterogeneous test platform called *Fermi*. It is equipped with two nodes, each connected to two GPUs. Each node has 2×six-core Intel Xeon E5649 processors running at 2.53 GHz. Each GPU is a NVidia Tesla M2075 (dual-slot S2070 module). Nodes are connected by a QDR Infiniband (40 Gbps) and an Ethernet (1-GBit) networks.

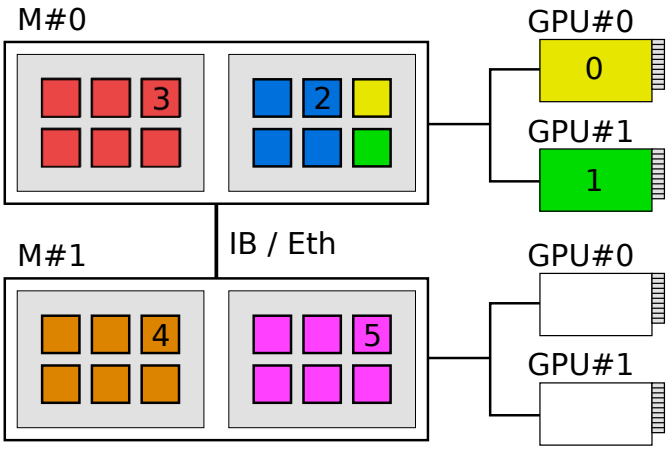


FIGURE A.4: Process deployment labeled as *Config1*. Node $M = 0$ has two processes running on GPUs, a multi-threaded process running on four cores and a multi-threaded process running on a socket with six cores. On node $M = 1$, two six-core multi-threaded processes execute. GPUs in node $M = 1$ are not used.

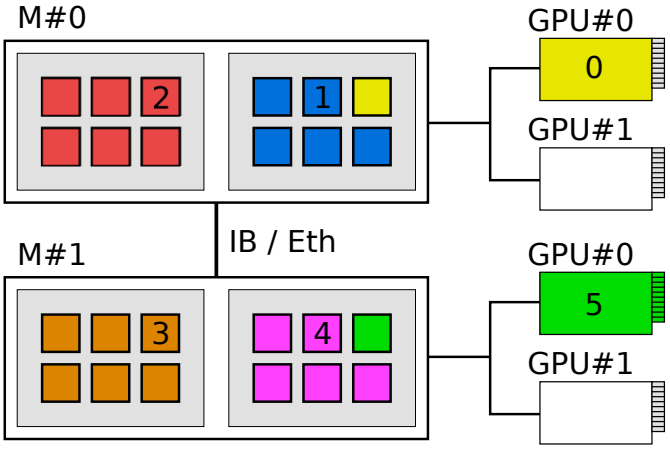


FIGURE A.5: Process deployment labeled as *Config2*. Node $M = 0$ has one process running on a GPU, a multi-threaded process running on five cores and a multi-threaded process running on a socket with six cores. Node $M = 1$ replicates the same layout of the node $M = 0$. Note that one GPU in each node is not used.

Operating system is CentOS 6.5, and a number of libraries are used for computation and communication. A process uses the function *dgemm* of the Intel MKL library to compute a rectangle of double precision elements. In the GPUs, cuBlas library is used with the same purpose. Open MPI 1.8.1 is used for communicating the processes in the application. Column communication uses the *MPI_Bcast* broadcasting function. Row communication uses the point-to-point non-blocking primitives *MPI_Isend* and *MPI_Irecv*. Non-blocking communication is used to minimize the wait times of the processes in the application due to the imbalances in the kernel execution.

FuPerMod utility is used for the 2D partitioning of the matrices based on a Functional Performance Model. The number of processes is $P = 6$, disposed in two configurations. *Config1* is shown in Figure A.4, and *Config2* is shown in Figure A.5. The goal is to

demonstrate that their modeling using τ–Lop provides with valuable information on the cost of the communications for comparing them. Note that both configurations contain multi-threaded, single-threaded and GPU processes. They run in two nodes and two GPUs, with slight differences in the mapping and resources used by the processes.

Two channels are considered in the cost modeling, *intra-node* and *inter-node*, respectively involving communications through shared memory ($c = 0$) and network ($c = 1$).

FuPerMod is used as well to compute the kernel execution time. The standard deviation of the execution time measured in each configuration is always under 1.35 %.

A.5 Matrix multiplication communications modeling

This section uses τ–Lop to model the communication cost of the SUMMA algorithm running on the two configurations described in section A.4. The procedure to obtain a balanced data partition using FuPerMod is described in section A.1, and summarized in Figure A.1. FuPerMod executes the user provided benchmark for the layout of processes described in a configuration file. Such file includes the number, type and mapping of the processors composing the hybrid application. We repeat the procedure for the two configurations (*Config1* and *Config2*) described in section A.4 (Figure A.4 and Figure A.5). Benchmarking is based either on a subset or a similar code of that executed by the kernel. In this work, the SUMMA algorithm through a range of matrix sizes is used as a benchmarking code. Resultant FPMs are then used to generate the 2D partition of the matrices. FuPerMod assigns the resulting rectangles to the processes. The test matrices have a size of $N \times N$ blocks, with $N = 256$. Each block has a size of $b \times b$ double precision elements, with $b = 64$.

A.5.1 *Config1* modeling

The FuPerMod input configuration file for *Config1* is:

```
# conf_file  Configuration 1. P = 6
node0 0 10   gpu id=0
node0 1 11   gpu id=1
node0 2 6-9  cpu OMP_NUM_THREADS=4
node0 3 0-5  cpu OMP_NUM_THREADS=6
node1 0 0-5  cpu OMP_NUM_THREADS=6
node1 1 6-11 cpu OMP_NUM_THREADS=6
```

The file contains a line per process, with the node where the process executes, the rank inside the node and the type of processor. In the case of multi-threaded processes,

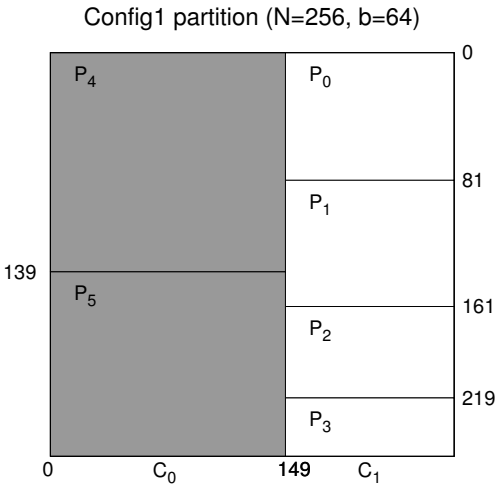


FIGURE A.6: *Config1* resultant partition for matrices A, B and C, and assignation to the $P = 6$ processes, with $b = 64$ and $N = 256$. White rectangles are assigned to processes running on the node 0, while grey rectangles correspond to processes on the node 1. C_i represents the number of blocks (width) of the i column.

$OMP\_NUM\_THREADS$ is used to indicate the number of threads of the process. After the execution of the FuPerMod procedure described before, the resulting matrices partition is shown in the Figure A.6. Note that the partition is the same for matrices C, A and B and only one is represented.

The costs of the communications for the SUMMA algorithm is the addition of the transmissions of the *pbc* of the matrix A and the *pbr* of the matrix B for each one of the N iterations. Both column and row communications are evaluated separately, and finally their costs are added.

A.5.1.1 Block row communication

Figure A.6 allows to visualize the processes which will interchange the *pbc* data of the matrix A. The communication is among processes in the same row. The communication cost is discussed per column. Table A.1 details the number of blocks of the *pbc* transmitted in an iteration of the $C_0 = 149$ blocks on the first column. Processes P_4 and P_5

TABLE A.1: *Config1* column C_0 transmissions.

Sender	Receiver	Blocks	Channel
P_4	P_0	81	T^1
P_4	P_1	58	T^1
P_5	P_1	22	T^1
P_5	P_2	58	T^1
P_5	P_3	37	T^1

send their respective parts of the *pbc* to the rest of the processes. The transmissions

of both processes progress concurrently through the network communication channel, shrinking the available bandwidth. Process P_4 sends to P_0 and P_1 its part of the *pbc*. The cost of the communications performed per iteration by P_4 is $T^1(81) + T^1(58)$, that reduces to $T^1(139)$ applying the $A1$ assumption in the section A.2. The corresponding communication cost of the process P_5 is $T^1(22) + T^1(58) + T^1(37) = T^1(117)$. The first column total communication cost is:

$$C_0 \times \left[T^1(139) \parallel T^1(117)\right] = C_0 \times \left[2 \parallel T^1(117) + T^1(22)\right] \tag{A.7}$$

Note that definition (A.7) is simplified using the assumption $A2$. The communication cost of the *pbc* for each of the $C_1 = 256 - 149 = 107$ iterations in the second column is represented in the table A.2.

TABLE A.2: *Config1* column C_1 transmissions.

Sender	Receiver	Blocks	Channel
P_0	P_4	81	T^1
P_1	P_4	58	T^1
P_1	P_5	22	T^1
P_2	P_5	58	T^1
P_3	P_5	37	T^1

The transmissions of the processes progress concurrently. Only the process P_1 has two sends in sequence, to processes P_4 and P_5, with cost $T^1(58) + T^1(22) = T^1(80)$. The second column communication cost of the $C_1 = 107$ blocks is:

$$\begin{aligned} &C_1 \times \left[T^1(81) \parallel T^1(80) \parallel T^1(58) \parallel T^1(37)\right] = \\ &C_1 \times \left[4 \parallel T^1(37) + 3 \parallel T^1(21) + 2 \parallel T^1(22) + T^1(1)\right] \end{aligned} \tag{A.8}$$

Note that assumption $A2$ is repeatedly applied to simplify the definition (A.8). The total cost for the row communication stage is:

$$\begin{aligned} &C_0 \times \left[2 \parallel T^1(117) + T^1(22)\right] + \\ &C_1 \times \left[4 \parallel T^1(37) + 3 \parallel T^1(21) + 2 \parallel T^1(22) + T^1(1)\right] = \\ &4 \parallel T^1(37\,C_1) + 3 \parallel T^1(21\,C_1) + 2 \parallel T^1(117\,C_0 + 22\,C_1) + T^1(22\,C_0 + 1\,C_1) = \\ &4 \parallel T^1(3589) + 3 \parallel T^1(2247) + 2 \parallel T^1(19787) + T^1(3385) \end{aligned} \tag{A.9}$$

A.5.1.2 Block column communication

Figure A.6 allows to visualize the processes that communicate *pbr* by columns. The columns have the same width, so the matrix B column communications are achieved

using the *MPI_Bcast* collective operation. A communicator is created for each column and the transmissions in the columns progress concurrently. We use the *Binomial tree* algorithm for the broadcast. The *root* (sender) process for the broadcast changes with the displacement of the *pbr*, so the cost of the collective operation can change significantly.

The communication cost is discussed per column. The first column is formed by two processes in the same *Node 1*, thus the broadcast derives in a point-to-point transmission, not affected by the root change. The message size is $C_0 = 149$ blocks. The final cost for the $N = 256$ iterations is:

$$N \times T^0(C_0) \tag{A.10}$$

A broadcast with $P = 4$ processes is executed in the second column. All communications progress through the shared memory in the *Node 0*. The Binomial tree cost changes with the root process during the displacement of the *pbr*. The cost is discussed next in a per root basis.

For the first $R_0 = 81$ rows, root process is P_0. It sends a message of size $C_1 = 107$ blocks to the process P_1, and then a message of the same size to the process P_3 while the process P_1 sends to the process P_2. Both transmissions progress concurrently. The cost is:

$$R_0 \times \left[T^0(C_1) + 2 \parallel T^0(C_1)\right] \tag{A.11}$$

When the root is P_1 the binomial tree deployment changes, although the cost is similar. P_1 sends to P_0, and then to process P_2 in concurrence with the send of P_0 to P_3. The cost for the $R_1 = 80$ blocks is:

$$R_1 \times \left[T^0(C_1) + 2 \parallel T^0(C_1)\right] \tag{A.12}$$

The same procedure is applied to calculate the cost of the $R_2 = 58$ blocks with P_2 as root:

$$R_2 \times \left[T^0(C_1) + 2 \parallel T^0(C_1)\right] \tag{A.13}$$

Finally, the last $R_3 = 37$ blocks with P_3 as root has a cost of:

$$R_3 \times \left[T^0(C_1) + 2 \parallel T^0(C_1)\right] \tag{A.14}$$

The total cost for the second column communication is the addition of the costs, with $N = R_0 + R_1 + R_2 + R_3$:

$$N \times \left[T^0(C_1) + 2 \parallel T^0(C_1)\right] \tag{A.15}$$

For the matrix B, the total cost of the column communications is:

$$N \times \Big[T^0(C_0)\Big] \parallel N \times \Big[T^0(C_1) + 2 \parallel T^0(C_1)\Big] =$$
$$N \times \Big[T^0(C_0) \parallel \Big(T^0(C_1) + 2 \parallel T^0(C_1)\Big)\Big]$$

Using the assumption $A2$, and the fact that $C_0 > C_1$, the cost is simplified to:

$$N \times \Big[2 \parallel T^0(C_1) + T^0(C_0 - C_1) \parallel \Big(2 \parallel T^0(C_1)\Big)\Big] =$$
$$N \times \Big[2 \parallel T^0(C_1) + 3 \parallel T^0(C_0 - C_1) + 2 \parallel T^0(C_1 - (C_0 - C_1))\Big] =$$
$$N \times \Big[2 \parallel T^0(C_1) + 3 \parallel T^0(C_0 - C_1) + 2 \parallel T^0(2\,C_1 - C_0)\Big] =$$

Summing up the terms, the cost becomes:

$$\begin{aligned} & N \times \Big[3 \parallel T^0(C_0 - C_1) + 2 \parallel T^0(3\,C_1 - C_0)\Big] = \\ & 3 \parallel T^0(N\,C_0 - N\,C_1) + 2 \parallel T^0(3\,N\,C_1 - N\,C_0) = \\ & 3 \parallel T^0(10752) + 2 \parallel T^0(44032) \end{aligned} \tag{A.16}$$

The total cost of the kernel communication is the addition of the row (A.9) and column (A.16) costs:

$$\begin{aligned} & 3 \parallel T^0(10752) + 2 \parallel T^0(44032) + \\ & 4 \parallel T^1(3589) + 3 \parallel T^1(2247) + 2 \parallel T^1(19787) + T^1(3385) \end{aligned} \tag{A.17}$$

The measured execution time in seconds for the *Config1* is shown in table A.3. Total time includes the computation and communication time, and the volume is the addition of the size of the messages sent by all process in MBytes.

TABLE A.3: *Config1* measured performance (in seconds)

Network	Total time	Comm. time	Comm. Volume
Ethernet	332.80	99.66	4952
Infiniband	310.22	75.30	4952

Note that the communication time using the Infiniband network is lower than in Ethernet (about 22 seconds), reducing the total execution time in almost the same time difference. Thus, an improvement in the communication cost directly results in an improvement of the total execution cost, because the computational cost remains constant.

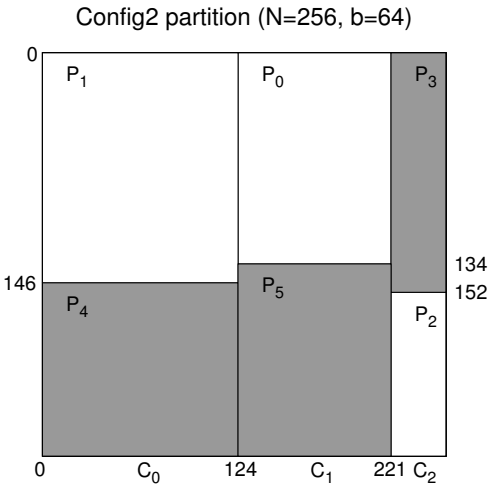


FIGURE A.7: *Config2* partition and assignation to the $P = 6$ processes, with $b = 64$ and $N = 256$. White rectangles are assigned to processes running on the node 0, while grey rectangles correspond to processes on the node 1.

A.5.2 *Config2* modeling

The methodology of cost estimation introduced in the former section is applied here for the *Config2* layout shown in Figure A.5. The FuPerMod input configuration file is:

```
# conf_file  Configuration 2. P = 6
node0 0 11   gpu id=0
node0 1 6-10 cpu OMP_NUM_THREADS=5
node0 2 0-5  cpu OMP_NUM_THREADS=6
node1 0 0-5  cpu OMP_NUM_THREADS=6
node1 1 6-10 cpu OMP_NUM_THREADS=5
node1 2 11   gpu id=0
```

The partition of the matrices A, B and C, and the assignation of the rectangles of blocks to the processes is represented in Figure A.7.

A.5.2.1 Block row communication

Pbc data transmissions are discussed by columns in the matrix A. There are three columns with $C_0 = 124$, $C_1 = 97$ and $C_2 = 35$ blocks. Table A.4 details the data transmissions between processes in each row for the first column C_0. Processes P_1 and P_4 perform their own transmissions in sequence. The cost of the transmissions of P_1 is $T^0(134) + T^1(158)$. Process P_4 transmissions cost is $T^0(116) + T^1(104)$. The total cost for the $C_0 = 124$ blocks of the first column, taking into account the concurrency of communications, is:

$$C_0 \times \left[\left(T^0(134) + T^1(158) \right) \parallel \left(T^0(116) + T^1(104) \right) \right]$$

TABLE A.4: *Config2* column C_0 transmissions.

Sender	Receiver	Blocks	Channel
P_1	P_0	134	T^0
P_1	P_3	146	T^1
P_1	P_5	12	T^1
P_4	P_2	104	T^1
P_4	P_3	6	T^0
P_4	P_5	110	T^0

Using the definition (A.5), the cost becomes:

$$C_0 \times \left[\Big(T^0(134) \parallel T^0(116)\Big) + \Big(T^1(158) + T^1(104)\Big)\right]$$

Finally, applying the $A2$ assumption, the cost will be:

$$\begin{aligned} &C_0 \times \Big[2 \parallel T^0(116) + T^0(18) + 2 \parallel T^1(104) + T^1(54)\Big] = \\ &2 \parallel T^0(116\,C_0) + T^0(18\,C_0) + 2 \parallel T^1(104\,C_0) + T^1(54\,C_0) \end{aligned} \tag{A.18}$$

The second column C_1 transmissions are represented in the table A.5. Processes P_0 and

TABLE A.5: *Config2* column C_1 transmissions.

Sender	Receiver	Blocks	Channel
P_0	P_1	134	T^0
P_0	P_3	134	T^1
P_5	P_1	12	T^1
P_5	P_2	104	T^1
P_5	P_3	18	T^0
P_5	P_4	110	T^0

P_5 transmissions contend for the communication channel. P_0 communication cost per block is $T^0(134) + T^1(134)$. P_5 communication cost is $T^0(128) + T^1(116)$. Then, the total cost for the $C_1 = 97$ blocks of the second column is:

$$\begin{aligned} &C_1 \times \left[\Big(T^0(134) + T^1(134)\Big) \parallel \Big(T^0(128) + T^1(116)\Big)\right] = \\ &C_1 \times \left[\Big(T^0(134) \parallel T^0(128)\Big) + \Big(T^1(134) \parallel T^1(116)\Big)\right] = \\ &C_1 \times \Big[2 \parallel T^0(128) + T^0(6) + 2 \parallel T^1(116) + T^1(18)\Big] = \\ &2 \parallel T^0(128\,C_1) + T^0(6\,C_1) + 2 \parallel T^1(116\,C_1) + T^1(18\,C_1) \end{aligned} \tag{A.19}$$

The third column communications are represented in table A.6. P_3 transmissions cost per block is $T^0(24) + T^1(280)$. P_2 transmission cost per block is $T^1(208)$. The total cost

TABLE A.6: *Config2* column C_2 transmissions.

Sender	Receiver	Blocks	Channel
P_3	P_0	134	T^1
P_3	P_1	146	T^1
P_3	P_4	6	T^0
P_3	P_5	18	T^0
P_2	P_4	104	T^1
P_2	P_5	104	T^1

of the $C_2 = 35$ blocks of the third column is:

$$\begin{aligned} &C_2 \times \Big[\Big(T^0(24) + T^1(280)\Big) \parallel T^1(208)\Big] = \\ &C_2 \times \Big[2 \parallel T^1(208) + T^1(72) + T^0(24)\Big] = \\ &2 \parallel T^1(208\,C_2) + T^1(72\,C_2) + T^0(24\,C_2) \end{aligned} \tag{A.20}$$

The total cost for the block row communications is the addition of the costs of the three columns, that is $C_0 + C_1 + C_2$, and hence:

$$\begin{aligned} &2 \parallel T^0(116\,C_0) + T^0(18\,C_0) + 2 \parallel T^1(104\,C_0) + T^1(54\,C_0) + \\ &2 \parallel T^0(128\,C_1) + T^0(6\,C_1) + 2 \parallel T^1(116\,C_1) + T^1(18\,C_1) + \\ &T^0(24\,C_2) + 2 \parallel T^1(208\,C_2) + T^1(72\,C_2) = \\ &2 \parallel T^0(116\,C_0 + 128\,C_1) + T^0(18\,C_0 + 6\,C_1 + 24\,C_2) + \\ &2 \parallel T^1(104\,C_0 + 116\,C_1 + 208\,C_2) + T^1(54\,C_0 + 18\,C_1 + 72\,C_2) = \\ &2 \parallel T^0(26800) + T^0(3654) + 2 \parallel T^1(31428) + T^1(10962) \end{aligned} \tag{A.21}$$

A.5.2.2 Block column communication

The broadcast collective operations in each column of the matrix B progress concurrently. Note that in the *Config2* each of the column communication progresses through the network channel and involves two processes per column, so they derive in point-to-point transmissions.

The first column have a communication cost of $N \times T^1(C_0)$, the second column has a cost of $N \times T^1(C_1)$, and the third column cost is $N \times T^1(C_2)$. The total cost including the contention of the columns is:

$$\begin{aligned} N \times \Big[T^1(C_0) \parallel T^1(C_1) \parallel T^1(C_2)\Big] = \\ N \times \Big[T^1(124) \parallel T^1(97) \parallel T^1(35)\Big] \end{aligned}$$

Applying the $A2$ assumption, the cost becomes:

$$\begin{aligned} N \times \left[3 \parallel T^1(35) + 2 \parallel T^1(62) + T^1(27)\right] = \\ 3 \parallel T^1(35\,N) + 2 \parallel T^1(62\,N) + T^1(27\,N) = \\ 3 \parallel T^1(8960) + 2 \parallel T^1(15872) + T^1(6912) \end{aligned} \tag{A.22}$$

The *Config2* total communication cost is the addition of the row (A.21) and column (A.22) costs:

$$2 \parallel T^0(26800) + T^0(3654) + 3 \parallel T^1(8960) + 2 \parallel T^1(47300) + T^1(17874) \tag{A.23}$$

The measured cost for this configuration is shown in the table A.7. As in *Config1*, the difference in the communication time between network types is directly reflected in the total execution time.

TABLE A.7: *Config2* measured performance (in seconds)

Network	Total time	Comm. time	Comm. Volume
Ethernet	366.29	148.32	6144
Infiniband	314.55	98.91	6144

A.5.3 Comparing the costs of both configurations

This section compares the cost of the *Config1* and *Config2* layouts exclusively from the models generated. The measurement of the parameters of τ–Lop in the target platform is not carried out. The goal is to find out from a formal analysis which is the more efficient layout.

Cost expression (A.16) shows that *Config1* has not network communication in the *pbc* communication. This fact leads to the hypothesis that the cost of the *Config1* is lower than that of the *Config2*. We call $\mathcal{C}_{cfg1}$ to the cost of the *Config1* layout, and $\mathcal{C}_{cfg2}$ to the cost of the *Config2* layout. The hypothesis could be expressed as $\mathcal{C}_{cfg2} > \mathcal{C}_{cfg1}$, that is:

$$\begin{aligned} &\left[2 \parallel T^0(26800) + T^0(3654) + 3 \parallel T^1(8960) + 2 \parallel T^1(47300) + T^1(17874)\right] > \\ &\left[3 \parallel T^0(10752) + 2 \parallel T^0(44032) + 4 \parallel T^1(3589) + 3 \parallel T^1(2037) + 2 \parallel T^1(19567) + T^1(3375)\right] \end{aligned} \tag{A.24}$$

Subtracting the terms in the same channel and contention factor, i.e., applying $n \parallel T^c(m_1) - n \parallel T^c(m_2) = n \parallel T^c(m_1 - m_2)$:

$$\begin{aligned} &T^0(3654) + 3 \parallel T^1(6923) + 2 \parallel T^1(27733) + T^1(14499) > \\ &3 \parallel T^0(10752) + 2 \parallel T^0(17232) + 4 \parallel T^1(3589) \end{aligned} \tag{A.25}$$

Because of $n \parallel T^{c_1}(m_1) - n \parallel T^{c_0}(m_2) \geq n \parallel T^{c_1}(m_1 - m_2), \forall c_1 \geq c_0 \; and \; m1 > m2$. This fact allows to simplify the terms $2 \parallel T^1(27733) - 2 \parallel T^0(17232) \geq 2 \parallel T^1(10501)$. And then:

$$\begin{aligned} &T^0(3654) + 3 \parallel T^1(6923) + 2 \parallel T^1(10501) + T^1(14499) > \\ &3 \parallel T^0(10752) + 4 \parallel T^1(3589) \end{aligned} \tag{A.26}$$

From the axiom in definition (3.5) in section 3.5, it is derived than an upper limit for $4 \parallel T^1(3589)$ is $T^1(3589 \times 4)$. Then:

$$\begin{aligned} &T^0(3654) + 3 \parallel T^1(6923) + 2 \parallel T^1(10501) + T^1(14499) > \\ &3 \parallel T^0(10752) + T^1(14356) \implies \\ &T^0(3654) + 3 \parallel T^1(6923) + 2 \parallel T^1(10501) + T^1(143) > 3 \parallel T^0(10752) \end{aligned} \tag{A.27}$$

Finally, $n_1 \parallel T^c(m_1) + n_2 \parallel T^c(m_2) \geq n_2 \parallel T^c(m_1 + m_2), \forall n_1 \geq n_2$, i.e. a lower limit is chosen, and then:

$$T^0(3654) + 2 \parallel T^1(17424) + T^1(143) > 3 \parallel T^0(10752) \tag{A.28}$$

In the evaluated architecture, with $T^1 \gg T^0$, even in presence of concurrency, the direct conclusion is that the hypothesis is correct.

A.6 Comparing different mappings for a configuration

The layout of processes shown in Figure A.7 poses the performance issue of the inter-node communication of the *pbc* in matrix A. The cost can be reduced by a simple change in the layout in the third column. By changing the order of the P_3 and P_2 processes, the inter-node communication could be reduced in favor of the more efficient intra-node one. The new *Config2_mod* layout of processes is shown in Figure A.8. While, the column communication cost of the *pbr* in matrix B will not change. This section evaluates this new layout of processes *Config2_mod* compared to the *Config2*, and gives experimental values that confirm the prediction. The goal is to apply the modeling methodology to evaluate the cost based on the mapping.

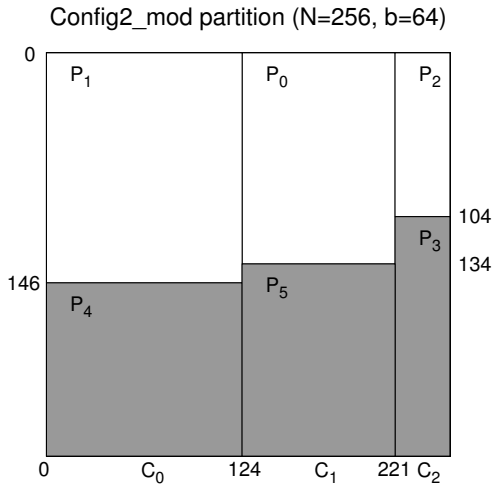


FIGURE A.8: *Config2_mod* partition and assignation to the $P = 6$ processes, with $b = 64$ and $N = 256$. White rectangles are assigned to processes running on the node 0, while grey rectangles correspond to processes on the node 1. P_2 and P_3 processes layout is changed with respect to *Config2*.

A.6.1 Block row communication

Again, the row communication is modeled separately by columns for the matrix A. Table A.8 details the interchange of data between processes in the row communication for the first column C_0. The cost per block of the transmissions of P_1 is $T^0(238) + T^1(54)$. The

TABLE A.8: *Config2_mod* column C_0 transmissions.

Sender	Receiver	Blocks	Channel
P_1	P_0	134	T^0
P_1	P_2	104	T^0
P_1	P_3	42	T^1
P_1	P_5	12	T^1
P_4	P_3	110	T^0
P_4	P_5	110	T^0

process P_4 transmission cost is $T^0(220)$. The total cost for the $C_0 = 124$ blocks of the first column, having into account the concurrency of communications, is:

$$\begin{aligned} & C_0 \times \left[\left(T^0(238) + T^1(54) \right) \parallel T^0(220) \right] = \\ & C_0 \times \left[\left(T^0(238) \parallel T^0(220) \right) + T^1(54) \right] = \\ & 2 \parallel T^0(220\, C_0) + T^0(18\, C_0) + T^1(54\, C_0) \end{aligned} \tag{A.29}$$

Note that definitions (A.5) and (A.6) are applied for simplifying definition (A.29). The second column transmissions are represented in the table A.9. The P_4 per block communication cost is $T^0(238) + T^1(30)$. The per block P_5 communication cost is

TABLE A.9: *Config_mod* column C_1 transmissions.

Sender	Receiver	Blocks	Channel
P_0	P_1	134	T^0
P_0	P_2	104	T^0
P_0	P_3	30	T^1
P_5	P_1	12	T^1
P_5	P_3	122	T^0
P_5	P_4	110	T^0

$T^0(232) + T^1(12)$. Then, the total cost for the $C_1 = 97$ blocks of the second column is:

$$\begin{aligned} &C_1 \times \left[\Big(T^0(238) + T^1(30)\Big) \parallel \Big(T^0(232) + T^1(12)\Big)\right] = \\ &C_1 \times \left[\Big(T^0(238) \parallel T^0(232)\Big) + \Big(T^1(30) \parallel T^1(12)\Big)\right] = \\ &C_1 \times \left[2 \parallel T^0(232) + T^0(6) + 2 \parallel T^1(12) + T^1(18)\right] = \\ &2 \parallel T^0(232\,C_1) + T^0(6\,C_1) + 2 \parallel T^1(12\,C_1) + T^1(18\,C_1) \end{aligned} \tag{A.30}$$

The third column communications is represented in table A.10. The P_2 per block com-

TABLE A.10: *Config_mod* column C_2 transmissions.

Sender	Receiver	Blocks	Channel
P_2	P_0	104	T^0
P_2	P_1	104	T^0
P_3	P_0	30	T^1
P_3	P_1	42	T^1
P_3	P_4	110	T^0
P_3	P_5	122	T^0

munication cost is $T^0(208)$. The P_3 per block communication cost is $T^0(232) + T^1(72)$. The total cost of the $C_2 = 35$ blocks of the third column is:

$$\begin{aligned} &C_2 \times \left[T^0(208) \parallel \Big(T^0(232) + T^1(72)\Big)\right] = \\ &C_2 \times \left[\Big(T^0(208) \parallel T^0(232)\Big) + T^1(72)\right] = \\ &C_2 \times \left[2 \parallel T^0(208) + T^0(24) + T^1(72)\right] = \\ &2 \parallel T^0(208\,C_2) + T^0(24\,C_2) + T^1(72\,C_2) \end{aligned} \tag{A.31}$$

Total communication for the row communication is the addition of the costs for the three columns in (A.29), (A.30) and (A.31), and hence:

$$\begin{aligned}
&2 \parallel T^0(220\,C_0) + T^0(18\,C_0) + T^1(54\,C_0) + \\
&2 \parallel T^0(232\,C_1) + T^0(6\,C_1) + 2 \parallel T^1(12\,C_1) + T^1(18\,C_1) + \\
&2 \parallel T^0(208\,C_2) + T^0(24\,C_2) + T^1(72\,C_2) = \\
&2 \parallel T^0(220\,C_0 + 232\,C_1 + 208\,C_2) + T^0(18\,C_0 + 6\,C_1 + 24\,C_2) + \\
&2 \parallel T^1(12\,C_1) + T^1(54\,C_0 + 18\,C_1 + 72\,C_2) = \\
&2 \parallel T^0(57064) + T^0(3654) + 2 \parallel T^1(1164) + T^1(10962)
\end{aligned} \tag{A.32}$$

A.6.2 Block column communication

The column communication does no change with respect to the *Config2* in the definition (A.22), because the communication is achieved in columns with the same processes in each column. Hence, the total cost of the matrix multiplication communication for the *Config2_mod* is the addition of the row (A.32) and column (A.22) communication costs:

$$\begin{aligned}
&2 \parallel T^0(57064) + T^0(3654) + 2 \parallel T^1(1164) + T^1(10962) + \\
&3 \parallel T^1(8960) + 2 \parallel T^1(15872) + T^1(6912) = \\
&2 \parallel T^0(57064) + T^0(3654) + 3 \parallel T^1(8960) + 2 \parallel T^1(17036) + T^1(17874)
\end{aligned} \tag{A.33}$$

Experimental cost values for this configuration are shown next:

TABLE A.11: *Config2_mod* measured performance (in seconds)

Network	Total time	Comm. time	Comm. Volume
Ethernet	348.75	131.73	6144
Infiniband	314.46	99.23	6144

Note that the number of bytes transmitted is the same of that in *Config2*. The difference between both layouts affects only to the channels used for communicating the data. Following, formal comparison of both layout is developed. Let $\mathcal{C}_{cfg2}$ be the cost of *Config2* communications and $\mathcal{C}'_{cfg2}$ the cost of *Config2_mod*. The departing hypothesis is that $\mathcal{C}_{cfg2} > \mathcal{C}'_{cfg2}$, that, from the definitions (A.23) and (A.33), becomes:

$$\begin{aligned}
&2 \parallel T^0(26800) + 2 \parallel T^1(47300) > 2 \parallel T^0(57064) + 2 \parallel T^1(17036) \implies \\
&2 \parallel T^1(30264) > 2 \parallel T^0(30264)
\end{aligned} \tag{A.34}$$

Note that, due to the change in the communication channel but not in the configuration, the number of blocks sent is the same, but through a different channels. The fact is captured by the modeling procedure. From the expression (A.34) the improvement of the communication performance is demonstrated.

A.7 Conclusions

This appendix introduces an on-going work on the communication performance modeling of hybrid applications. The target platforms are heterogeneous clusters for parallel computing composed of multi-core processors and accelerators connected by a hierarchical network.

The execution cost of a hybrid application on a heterogeneous platform highly depends on the distribution of the workload between the processes. The processes have to be assigned with an amount of computation in proportion to their different performance capabilities. Also, the communications have a significant influence in the final cost. The uneven computational load balancing carries out an unbalance in the communications, because some processes have to communicate a higher volume of data. In addition, due to the uneven communication channels, the deployment of the processes on the platform impacts the overall cost because it establishes the channel used by a process for communicating with the rest.

An extensive amount of work have been done in load balancing of computation and optimization of communications for parallel processing. These problems have been faced using Graph Partitioning techniques and more recently using Functional Performance Models. The FPMs better describe the performance of a process by integrating performance characteristics of both the platform and the application. However, FPMs lack of the capability of accurately including the communication cost in the optimization procedure. The τ–Lop model is applied to fill this gap. The model is extended to heterogeneous platforms under some basic and meaningful assumptions. The performance modeling methodology departs from a load balanced scenery using FPMs. The workload distribution between the processes conceptually transforms the heterogeneous system in homogeneous, in which processes balances their capabilities with the different assigned workloads. τ–Lop estimates the communication cost for a given process distribution in the platform, and it compares different layouts in order to choose that of minimum cost.

The SUMMA matrix multiplication kernel, representative for many HPC kernels, has been evaluated and the estimations of the cost of communications using τ–Lop match

the real execution costs. Application of the performance modeling methodology to other data parallel kernels is direct.

Our current work is focused on covering two open issues. The first is the estimation of the parameters in a heterogeneous platform. Such estimation will follow the methodology described in chapter 7. The parameter estimation will allow to quantitatively compare the estimated cost with the real value to fully asses the utility and adaptability of the model in heterogeneous computing. Nevertheless, this point is not important in the cases in which the goal is to find the best layout of processes in a platform comparing the cost expressions. The second issue is the implementation of the cost modeling procedure to automatically estimate the cost associated to the execution of a kernel for an specific layout of processes in a heterogeneous platform.

Bibliography

[1] Roger W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.*, 20(3):389–398, March 1994. ISSN 0167-8191. doi: 10.1016/S0167-8191(06)80021-9. URL http://dx.doi.org/10.1016/S0167-8191(06)80021-9.

[2] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM. ISBN 0-89791-589-5. doi: http://doi.acm.org/10.1145/155332.155333. URL http://doi.acm.org/10.1145/155332.155333.

[3] David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken. LogP: a practical model of parallel computation. *Commun. ACM*, 39:78–85, November 1996. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/240455.240477. URL http://doi.acm.org/10.1145/240455.240477.

[4] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. In *Proc. of the seventh annual ACM symposium on Parallel algorithms and architectures*, SPAA '95, pages 95–105, NY, USA, 1995. ISBN 0-89791-717-0. doi: http://doi.acm.org/10.1145/215399.215427. URL http://doi.acm.org/10.1145/215399.215427.

[5] MPI Forum. MPI: A Message-Passing Interface Standard. Version 3.0, September 21th 2012. available at: http://www.mpi-forum.org.

[6] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi, Alok Choudhary,

Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Zhong Jin, Yutaka Ishikawa, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias S. Mueller, Wolfgang E. Nagel, Hiroshi Nakashima, Michael E Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad Van Der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick. The International Exascale Software Project Roadmap. *Int. J. High Perform. Comput. Appl.*, 25 (1):3–60, February 2011. ISSN 1094-3420. doi: 10.1177/1094342010391989. URL http://dx.doi.org/10.1177/1094342010391989.

[7] Georges Da Costa, Thomas Fahringer, Juan Antonio Rico Gallego, Ivan Grasso, Atanas Hristov, Helen Karatza, Alexey Lastovetsky, Fabrizio Marozzo, Dana Petcu, Georgios Stavrinides, Domenico Talia, Paolo Trunfio, and Hrachya Astsatryan. Exascale Machines Require New Programming Paradigms and Runtimes. *Supercomputing frontiers and innovations*, 2(2):6–27, 2015. ISSN 2313-8734. URL http://superfri.org/superfri/article/view/44.

[8] Gregory D. Benson, Cho wai Chu, Qing Huang, and Sadik G. Caglar. A Comparison of MPICH Allgather Algorithms on Switched Networks. In *In Proceedings of the 10th EuroPVM/MPI 2003 Conference*, pages 335–343. Springer, 2003.

[9] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGP in theory and practice: An in-depth analysis of modern interconnection networks and benchmarking methods for collective operations. *Simulation Modelling Practice and Theory*, 17(9):1511–1521, 2009. ISSN 1569-190X. doi: 10.1016/j.simpat.2009.06.007. URL http://www.sciencedirect.com/science/article/pii/S1569190X09000811. Advances in System Performance Modelling, Analysis and Enhancement.

[10] Torsten Hoefler, William Gropp, William Kramer, and Marc Snir. Performance Modeling for Systematic Performance Tuning. In *State of the Practice Reports*, SC '11, pages 6:1–6:12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1139-7. doi: 10.1145/2063348.2063356. URL http://doi.acm.org/10.1145/2063348.2063356.

[11] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Accurately measuring overhead, communication time and progression of blocking and nonblocking collective operations at massive scale. *Int. J. Parallel Emerg. Distrib. Syst.*, 25(4):

241–258, August 2010. ISSN 1744-5760. doi: 10.1080/17445760902894688. URL http://dx.doi.org/10.1080/17445760902894688.

[12] Rolf Rabenseifner. Automatic Profiling of MPI Applications with Hardware Performance Counters. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 35–42, London, UK, 1999. Springer-Verlag. ISBN 3-540-66549-8. URL http://dl.acm.org/citation.cfm?id=648136.748802.

[13] Jelena Pješivac-Grbović, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing*, 10:127–143, June 2007. ISSN 1386-7857. doi: 10.1007/s10586-007-0012-0. URL http://dl.acm.org/citation.cfm?id=1265235.1265248.

[14] Sascha Hunold, Alexandra Carpen-Amarie, and Jesper Larsson Träff. Reproducible MPI micro-benchmarking isn't as easy as you think. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 69:69–69:76, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2875-3. doi: 10.1145/2642769.2642785. URL http://doi.acm.org/10.1145/2642769.2642785.

[15] Argonne National Laboratory: MPICH2 project., 2013. URL http://www.mcs.anl.gov/mpi/mpich2.

[16] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, DavidJ. Daniel, Richard L. Graham, and Timothy S. Woodall. *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23163-9. doi: 10.1007/978-3-540-30218-6{_}19. URL http://dx.doi.org/10.1007/978-3-540-30218-6_19.

[17] Kirk W. Cameron and Xian-He Sun. Quantifying Locality Effect in Data Access Delay: Memory logP. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 48.2–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1926-1. URL http://dl.acm.org/citation.cfm?id=838237.838468.

[18] K W Cameron, R Ge, and X H Sun. $\log_m P$ and $\log_3 P$: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems. *Computers IEEE Transactions on*, 56(3):314–327, 2007. doi: 10.1109/TC.2007.38".

[19] Kirk W. Cameron and Rong Ge. Predicting and Evaluating Distributed Communication Performance. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 43–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2153-3. doi: http://dx.doi.org/10.1109/SC.2004.40. URL http://dx.doi.org/10.1109/SC.2004.40.

[20] Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín. τ–Lop: Modeling performance of shared memory MPI. *Parallel Computing*, 46:14 – 31, 2015. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2015.02.006. URL http://www.sciencedirect.com/science/article/pii/S0167819115000447.

[21] Juan-Antonio Rico-Gallego, Juan-Carlos Díaz-Martín, and Alexey L. Lastovetsky. Extending τ–Lop to Model Concurrent MPI Communications in Multicore Clusters. *Paper submitted to Future Generation Computer Systems journal*, 2015.

[22] Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín. On the Performance of Concurrent Transfers in Collective Algorithms. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 143–144, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1903-4. doi: 10.1145/2488551.2488601. URL http://doi.acm.org/10.1145/2488551.2488601.

[23] Juan-Antonio Rico-Gallego, Juan-Carlos Díaz-Martín, and Alexey L. Lastovetsky. Modeling Contention and Mapping Effects in Multi-core Clusters. In Springer, editor, *Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar)*, volume 9523 of *Lecture Notes in computer Science*. Springer International Publishing, 2015. In press.

[24] E.W. Chan, M.F. Heimlich, A. Purkayastha, and R.A. van de Geijn. On optimizing collective communication. *Cluster Computing, 2004 IEEE International Conference on*, pages 145–155, Sept. 2004. ISSN 1552-5244. doi: 10.1109/CLUSTR.2004.1392612.

[25] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective Communication: Theory, Practice, and Experience: Research Articles. *Concurr. Comput. : Pract. Exper.*, 19(13):1749–1783, September 2007. ISSN 1532-0626. doi: 10.1002/cpe.v19:13. URL http://dx.doi.org/10.1002/cpe.v19:13.

[26] Rajeev Thakur and William D. Gropp. Improving the Performance of Collective Operations in MPICH. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *Recent Advances in Parallel Virtual Machine and*

Message Passing Interface, volume 2840 of *Lecture Notes in Computer Science*, pages 257–267. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20149-6. doi: 10.1007/978-3-540-39924-7{_}38. URL http://dx.doi.org/10.1007/978-3-540-39924-7_38.

[27] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005. doi: 10.1177/1094342005051521. URL http://hpc.sagepub.com/content/19/1/49.abstract.

[28] Jesper Larsson Träff and Andreas Ripke. *An Optimal Broadcast Algorithm Adapted to SMP Clusters*, volume 3666 of *Lecture Notes in Computer Science*, pages 48–56. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29009-4. doi: 10.1007/11557265{_}11. URL http://dx.doi.org/10.1007/11557265_11.

[29] Rolf Rabenseifner and Jesper Larsson Träff. *More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems*, volume 3241 of *Lecture Notes in Computer Science*, pages 36–46. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23163-9. doi: 10.1007/978-3-540-30218-6{_}13. URL http://dx.doi.org/10.1007/978-3-540-30218-6_13.

[30] L.V. Kale, S. Kumar, and K. Varadarajan. A framework for collective personalized communication. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 9 pp.–, April 2003. doi: 10.1109/IPDPS.2003.1213166.

[31] Khalid Al-Tawil and Csaba Andras Mortíz. Performance Modeling and Evaluation of MPI. *J. Parallel Distrib. Comput.*, 61(2):202–223, February 2001. ISSN 0743-7315. doi: 10.1006/jpdc.2000.1677. URL http://dx.doi.org/10.1006/jpdc.2000.1677.

[32] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, IPDPS '00, pages 1176–1183, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67442-X. URL http://dl.acm.org/citation.cfm?id=645612.662667.

[33] T. Kielmann, H.E. Bal, and S. Gorlatch. Bandwidth-efficient collective communication for clustered wide area systems. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 492–499, 2000. doi: 10.1109/IPDPS.2000.846026.

[34] LuizAngelo Barchet-Estefanel and Grégory Mounié. *Fast Tuning of Intra-cluster Collective Communications*, volume 3241 of *Lecture Notes in Computer Science*, pages 28–35. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23163-9. doi: 10.1007/978-3-540-30218-6{_}12. URL http://dx.doi.org/10.1007/978-3-540-30218-6_12.

[35] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. LogGPS: a parallel computational model for synchronization analysis. *SIGPLAN Not.*, 36:133–142, June 2001. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/568014.379592. URL http://doi.acm.org/10.1145/568014.379592.

[36] WenGuang Chen, JiDong Zhai, Jin Zhang, and WeiMin Zheng. LogGPO: An accurate communication model for performance prediction of MPI programs. 52 (10):1785–1791, 2009. doi: 10.1007/s11432-009-0161-2. URL http://dx.doi.org/10.1007/s11432-009-0161-2.

[37] T. Hoefler, T. Mehlan, F. Mietke, and Wolfgang Rehm. LogfP - a model for small messages in InfiniBand. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 6 pp.–, 2006. doi: 10.1109/IPDPS.2006.1639624.

[38] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *Cluster Computing, 2008 IEEE International Conference on*, pages 116–125, 2008. doi: 10.1109/CLUSTR.2008.4663762.

[39] Sang Cheol Kim and Sunggu Lee. Measurement and Prediction of Communication Delays in Myrinet Networks. *J. Parallel Distrib. Comput.*, 61(11):1692–1704, November 2001. ISSN 0743-7315. doi: 10.1006/jpdc.2001.1761. URL http://dx.doi.org/10.1006/jpdc.2001.1761.

[40] Liang Yuan, Yunquan Zhang, Yuxin Tang, Li Rao, and Xiangzheng Sun. LogGPH: A Parallel Computational Model with Hierarchical Communication Awareness. In *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, pages 268 –274, dec. 2010. doi: 10.1109/CSE.2010.40.

[41] Hao Zhu, David Goodell, William Gropp, and Rajeev Thakur. Hierarchical Collectives in MPICH2. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 325–326, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03769-6. doi: http://dx.doi.org/10.1007/978-3-642-03770-2_41. URL http://dx.doi.org/10.1007/978-3-642-03770-2_41.

[42] Bibo Tu, Jianping Fan, Jianfeng Zhan, and Xiaofang Zhao. Performance analysis and optimization of MPI collective operations on multi-core clusters. *The Journal of Supercomputing*, 60(1):141–162, 2012. ISSN 0920-8542. doi: 10.1007/s11227-009-0296-3. URL http://dx.doi.org/10.1007/s11227-009-0296-3.

[43] Maxime Martinasso and Jean-François Méhaut. *Prediction of Communication Latency over Complex Network Behaviors on SMP Clusters*, volume 3670 of *Lecture Notes in Computer Science*, pages 172–186. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28701-8. doi: 10.1007/11549970{_}13. URL http://dx.doi.org/10.1007/11549970_13.

[44] Maxime Martinasso and Jean-François Méhaut. A Contention-aware Performance Model for HPC-based Networks: A Case Study of the InfiniBand Network. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par'11, pages 91–102, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23399-9. URL http://dl.acm.org/citation.cfm?id=2033345.2033357.

[45] V. Jerome, M. Maxime, V. Jean-Marc, and M. Jean-Francois. Predictive models for bandwidth sharing in high performance clusters. In *Cluster Computing, 2008 IEEE International Conference on*, pages 286–291, 2008. doi: 10.1109/CLUSTR.2008.4663783.

[46] Jun Zhu, Alexey Lastovetsky, Shoukat Ali, and Rolf Riesen. Communication Models for Resource Constrained Hierarchical Ethernet Networks. In *11th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2013)*, Aachen, Germany, 26 August 2013.

[47] Csaba Andras Moritz and Matthew I. Frank. LoGPC: Modeling Network Contention in Message-Passing Programs. *IEEE Trans. Parallel Distrib. Syst.*, 12 (4):404–415, April 2001. ISSN 1045-9219. doi: 10.1109/71.920589. URL http://dx.doi.org/10.1109/71.920589.

[48] L.A. Steffenel. Modeling Network Contention Effects on All-to-All Operations. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–10, 2006. doi: 10.1109/CLUSTR.2006.311889.

[49] Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. *J. Parallel Distrib. Comput.*, 63(3):251–263, March 2003. ISSN 0743-7315. doi: 10.1016/S0743-7315(03)00008-X. URL http://dx.doi.org/10.1016/S0743-7315(03)00008-X.

[50] J. Hatta and S. Shibusawa. Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In *Parallel Processing, 2000. Proceedings. 2000 International Workshops on*, pages 173–180, 2000. doi: 10.1109/ICPPW.2000.869101.

[51] F. Ooshita, S. Matsumae, and T. Masuzawa. Efficient gather operation in heterogeneous cluster systems. In *High Performance Computing Systems and Applications, 2002. Proceedings. 16th Annual International Symposium on*, pages 196–204, 2002. doi: 10.1109/HPCSA.2002.1019155.

[52] J.L. Bosque and L.P. Perez. HLogGP: a new parallel computational model for heterogeneous clusters. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 403–410, April 2004. doi: 10.1109/CCGrid.2004.1336594.

[53] A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn. An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, volume 2, pages 6 pp.–, 2006. doi: 10.1109/ICPADS.2006.24.

[54] Alexey Lastovetsky and Vladimir Rychkov. Accurate and efficient estimation of parameters of heterogeneous communication performance models. *Int. J. High Perform. Comput. Appl.*, 23(2):123–139, May 2009. ISSN 1094-3420. doi: 10.1177/1094342009103947. URL http://dx.doi.org/10.1177/1094342009103947.

[55] A. Lastovetsky, V. Rychkov, and M. O'Flynn. Revisiting communication performance models for computational clusters. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–11, 2009. doi: 10.1109/IPDPS.2009.5160918.

[56] J.C. Díaz Martín, J.A. Rico Gallego, J.M. Álvarez Llorente, and J.F. Perogil Duque. An MPI-1 Compliant Thread-Based Implementation. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 5759 of *Lecture Notes in Computer Science*, pages 327–328. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03769-6. doi: 10.1007/978-3-642-03770-2_42. URL http://dx.doi.org/10.1007/978-3-642-03770-2_42.

[57] Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín. Performance Evaluation of Thread-Based MPI in Shared Memory. In Yiannis Cotronis, Anthony Danalis, DimitriosS. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances*

in the Message Passing Interface, volume 6960 of *Lecture Notes in Computer Science*, pages 337–338. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24448-3. doi: 10.1007/978-3-642-24449-0_42. URL http://dx.doi.org/10.1007/978-3-642-24449-0_42.

[58] Intel. Intel MPI Benchmarks. URL https://software.intel.com/en-us/articles/intel-mpi-benchmarks/.

[59] Darius Buntinas, Guillaume Mercier, and William Gropp. Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem. In *Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 521–530, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2585-7. doi: http://dx.doi.org/10.1109/CCGRID.2006.31. URL http://dx.doi.org/10.1109/CCGRID.2006.31.

[60] Hyun wook Jin, Sayantan Sur, Lei Chai, and Dhabaleswar K. Panda. LiMIC: Support for high-performance MPI intra-node communication on Linux cluster. In *In International Conference on Parallel Processing (ICPP*, pages 184–191, 2005.

[61] Darius Buntinas, Brice Goglin, David Goodell, Guillaume Mercier, and Stéphanie Moreaud. Cache-Efficient, Intranode, Large-Message MPI Communication with MPICH2-Nemesis. In *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, pages 462–469, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3802-0. doi: 10.1109/ICPP.2009.22. URL http://dx.doi.org/10.1109/ICPP.2009.22.

[62] Teng Ma, G. Bosilca, A. Bouteiller, B. Goglin, J.M. Squyres, and J.J. Dongarra. Kernel Assisted Collective Intra-node MPI Communication among Multi-Core and Many-Core CPUs. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 532–541, Sept 2011. doi: 10.1109/ICPP.2011.29.

[63] Juan Antonio Rico Gallego, Juan Carlos Díaz Martín, Carolina Gómez-Tostón Gutiérrez, and Álvaro Cortés Fácila. Improving Collectives by User Buffer Relocation. In Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra, editors, *Recent Advances in the Message Passing Interface*, volume 7490 of *Lecture Notes in Computer Science*, pages 287–288. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33517-4. doi: 10.1007/978-3-642-33518-1_35. URL http://dx.doi.org/10.1007/978-3-642-33518-1_35.

[64] Marc Pérache, Patrick Carribault, and Hervé Jourdren. MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 5759 of *Lecture Notes in*

Computer Science, pages 94–103. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03769-6. doi: 10.1007/978-3-642-03770-2_16. URL http://dx.doi.org/10.1007/978-3-642-03770-2_16.

[65] Jing Chen, Linbo Zhang, Yunquan Zhang, and Wei Yuan. Performance Evaluation of Allgather Algorithms On Terascale Linux Cluster with Fast Ethernet. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, HPCASIA '05, pages 437–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2486-9. doi: 10.1109/HPCASIA.2005.75. URL http://dx.doi.org/10.1109/HPCASIA.2005.75.

[66] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 8(11):1143–1156, 1997. ISSN 1045-9219. doi: 10.1109/71.642949.

[67] D. Turner and Xuehua Chen. Protocol-dependent message-passing performance on Linux clusters. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pages 187–194, 2002. doi: 10.1109/CLUSTR.2002.1137746.

[68] Sascha Hunold and Alexandra Carpen-Amarie. On the impact of synchronizing clocks and processes on benchmarking mpi collectives. In *Proceedings of the 22Nd European MPI Users' Group Meeting*, EuroMPI '15, pages 8:1–8:10, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3795-3. doi: 10.1145/2802658.2802662. URL http://doi.acm.org/10.1145/2802658.2802662.

[69] Sascha Hunold and Alexandra Carpen-Amarie. MPI benchmarking revisited: Experimental design and reproducibility. *CoRR*, abs/1505.07734, 2015. URL http://arxiv.org/abs/1505.07734.

[70] Jack Dongarra and Alexey L. Lastovetsky. *High Performance Heterogeneous Computing*. Wiley-Interscience, New York, NY, USA, 2009. ISBN 0470040394, 9780470040393.

[71] T. Hoefler, A. Lichei, and Wolfgang Rehm. Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, 2007. doi: 10.1109/IPDPS.2007.370593.

[72] Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S. Muller. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System. In *Proceedings of the 2009 18th International Conference on Parallel*

Architectures and Compilation Techniques, PACT '09, pages 261–270, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3771-9. doi: 10.1109/PACT.2009.22. URL http://dx.doi.org/10.1109/PACT.2009.22.

[73] *Media Engineering Group*. tauLop tool. http://gim.unex.es/taulop, 2013.

[74] Milton Abramowitz and Irene A. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55 of *National Bureau of Standards Applied Mathematics Series*. For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 1964.

[75] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *Computers, IEEE Transactions on*, 50(10):1052–1070, Oct 2001. ISSN 0018-9340. doi: 10.1109/12.956091.

[76] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997. doi: 10.1137/1.9780898719642. URL http://epubs.siam.org/doi/abs/10.1137/1.9780898719642.

[77] Alexey Lastovetsky and Ravi Reddy. Data Partitioning with a Functional Performance Model of Heterogeneous Processors. *Int. J. High Perform. Comput. Appl.*, 21(1):76–90, February 2007. ISSN 1094-3420. doi: 10.1177/1094342006074864. URL http://dx.doi.org/10.1177/1094342006074864.

[78] Alexey Lastovetsky and Ravi Reddy. Data Distribution for Dense Factorization on Computers with Memory Heterogeneity. *Parallel Comput.*, 33(12):757–779, December 2007. ISSN 0167-8191. doi: 10.1016/j.parco.2007.06.001. URL http://dx.doi.org/10.1016/j.parco.2007.06.001.

[79] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky. FuPerMod: A Framework for Optimal Data Partitioning for Parallel Scientific Applications on Dedicated Heterogeneous HPC Platforms. In *12th International Conference on Parallel Computing Technologies (PaCT-2013)*, pages 182–196, St. Petersburg, Russia, 30 Sept - 4 Oct 2013. Lecture Notes in Computer Science 7979, Springer, Lecture Notes in Computer Science 7979, Springer.

[80] David Clarke, Ziming Zhong, V. Rychkov, and Alexey Lastovetsky. FuPerMod: a software tool for the optimization of data-parallel applications on heterogeneous platforms. *The Journal of Supercomputing*, 69:61– 69, 2014. ISSN

0920-8542. doi: 10.1007/s11227-014-1207-9. URL http://dx.doi.org/10.1007/s11227-014-1207-9.

[81] David Clarke, Alexey Lastovetsky, and Vladimir Rychkov. *Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models*, volume 7155 of *Lecture Notes in Computer Science*, pages 450–459. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29736-6. doi: 10.1007/978-3-642-29737-3{_}50. URL http://dx.doi.org/10.1007/978-3-642-29737-3_50.

[82] Ziming Zhong, V. Rychkov, and A. Lastovetsky. Data Partitioning on Heterogeneous Multicore Platforms. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 580–584, Sept 2011. doi: 10.1109/CLUSTER.2011.64.

[83] Ziming Zhong, V. Rychkov, and A. Lastovetsky. Data Partitioning on Heterogeneous Multicore and Multi-GPU Systems Using Functional Performance Models of Data-Parallel Applications. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 191–199, Sept 2012. doi: 10.1109/CLUSTER.2012.34.

[84] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x. URL http://dx.doi.org/10.1002/j.1538-7305.1970.tb01770.x.

[85] B. W. Kernighan S. Lin. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/169020.

[86] David F. Gleich. Hierarchical Directed Spectral Graph Partitioning. Information Networks, Stanford University, Final Project, 2005, 2006. URL http://www.cs.purdue.edu/homes/dgleich/publications/Gleich2005-hierarchicaldirectedspectral.pdf. Cited over 6 times.

[87] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM. ISBN 0-89791-816-9. doi: 10.1145/224170.224228. URL http://doi.acm.org/10.1145/224170.224228.

[88] Bruce Hendrickson. *Graph partitioning and parallel solvers: Has the emperor no clothes?*, volume 1457 of *Lecture Notes in Computer Science*, pages 218–225. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64809-3. doi: 10.1007/BFb0018541. URL http://dx.doi.org/10.1007/BFb0018541.

[89] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998. ISSN 1064-8275. doi: 10.1137/S1064827595287997. URL http://dx.doi.org/10.1137/S1064827595287997.

[90] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–13, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-89791-984-X. URL http://dl.acm.org/citation.cfm?id=509058.509086.

[91] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129, January 1998. ISSN 0743-7315. doi: 10.1006/jpdc.1997.1404. URL http://dx.doi.org/10.1006/jpdc.1997.1404.

[92] H. D. Simon. Parallel methods on large-scale structural analysis and physics applicationspartitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2):135–148, 1991. doi: http://dx.doi.org/10.1016/0956-0521(91)90014-V. URL http://www.sciencedirect.com/science/article/pii/095605219190014V.

[93] Roy D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–481, 1991. ISSN 1096-9128. doi: 10.1002/cpe.4330030502. URL http://dx.doi.org/10.1002/cpe.4330030502.

[94] E. Aubanel and X. Wu. Incorporating Latency in Heterogeneous Graph Partitioning. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007. doi: 10.1109/IPDPS.2007.370577.

[95] G. Karypis and K. Schloegel. METIS: Serial graph partitioning and fill-reducing matrix ordering, Version 5.1. URL http://glaros.dtc.umn.edu/gkhome/metis/metis.

[96] G. Karypis and K. Schloegel. ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 4.0. URL http://glaros.dtc.umn.edu/gkhome/metis/parmetis.

[97] C. Walshaw, M. Cross, and M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, 1997.

[98] C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview. In F. Magoules, editor, *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pages 27–58. Civil-Comp Ltd., 2007. ISBN 978-1-874672-29-6. (Invited chapter).

[99] C. Chevalier and F. Pellegrini. PT-Scotch: A Tool for Efficient Parallel Graph Ordering. *Parallel Comput.*, 34(6-8):318–331, July 2008. ISSN 0167-8191. doi: 10.1016/j.parco.2007.12.001. URL http://dx.doi.org/10.1016/j.parco.2007.12.001.

[100] François Pellegrini and Jean Roman. SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, HPCN Europe 1996, pages 493–498, London, UK, UK, 1996. Springer-Verlag. ISBN 3-540-61142-8. URL http://dl.acm.org/citation.cfm?id=645560.658570.

[101] François Pellegrini. *Static Mapping of Process Graphs*, pages 115–136. John Wiley & Sons, Inc., 2013. ISBN 9781118601181. doi: 10.1002/9781118601181.ch5. URL http://dx.doi.org/10.1002/9781118601181.ch5.

[102] Francois Tessier, Guillaume Mercier, and Emmanuel Jeannot. Process placement in multicore clusters:algorithmic issues and practical techniques. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):993–1002, 2014. ISSN 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.104.

[103] Siew Yin Chan, Teck Chaw Ling, and Eric Aubanel. Performance Modeling for Hierarchical Graph Partitioning in Heterogeneous Multi-core Environment. *Parallel Comput.*, 46(C):78–97, July 2015. ISSN 0167-8191. doi: 10.1016/j.parco.2014.05.001. URL http://dx.doi.org/10.1016/j.parco.2014.05.001.

[104] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka. An efficient, model-based CPU-GPU heterogeneous FFT library. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–10, April 2008. doi: 10.1109/IPDPS.2008.4536163.

[105] J. Daniel García, Rafael Sotomayor, Javier Fernández, and Luis Miguel Sánchez. Static partitioning and mapping of kernel-based applications over modern heterogeneous architectures. *Simulation Modelling Practice and Theory*, 58, Part

1:79–94, 11 2015. doi: http://dx.doi.org/10.1016/j.simpat.2015.05.010. URL http://www.sciencedirect.com/science/article/pii/S1569190X15000933.

[106] Robert A. van de Geijn and Jerrell Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. Technical report, Austin, TX, USA, 1995.

[107] Khalid Hasanov, Jean-Noel Quintin, and Alexey Lastovetsky. Hierarchical Approach to Optimization of Parallel Matrix Multiplication on Large-Scale Platforms. *The Journal of Supercomputing*, 71:3991–4014, 11/2015 2015.

[108] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix-matrix multiplication on heterogeneous platforms. In *Parallel Processing, 2000. Proceedings. 2000 International Conference on*, pages 289–298, 2000. doi: 10.1109/ICPP.2000.876144.