Communication Performance Models for Heterogeneous Computational Clusters

Maureen O'Flynn

The thesis is submitted to University College Dublin
for the degree of Doctor of Philosophy

June 2009

Head of Department: Joe Carty
Supervisor of Research : Alexey Lastovetsky

School of Computer Science and Informatics,
College of Engineering, Mathematical and Physical Sciences,
University College Dublin,
Belfield, Dublin.

I would like to dedicate this thesis to my husband Tony.

Acknowledgements

I would like to thank my supervisor, Alexey Lastovetsky, for an unfailing support, help and advice. From the very beginning he provided a very focused and specific guidance that meant my research never drifted and was always rewarding towards our goals, in the vast and complex subject arena of parallel communications. I am especially grateful for his encouragement through the difficult times. I would like to give special thanks to Vladimir Rychkov who helped me with every aspect of this thesis with great interest and enthusiasm, spending many hours in discussions. I would like to thank Is-Haka Imwawa for his help and encouragement at the early stages of the project, he fostered our early ideas and provided me with guidance when starting out. I would like to thank Ravi Reddy for his ideas and help also.

I would like to thank Science Foundation Ireland for their financial support with my grant funding. I would like thank Brett Becker for his work with the cluster throughout the project, and to thank the other postgraduates of our group, Thomas, Robert and Michele, and the administrative and technical staff of the computer science department. Finally I would like to thank all the friends and family, in particular my father, for their constant faith and motivation. Most of all I would like to acknowledge my husband, who made this journey one of happiness.

Abstract

The parallel systems of the past are made up of specialist homogeneous processors dedicated to their task. These systems are evolved so they may now harness the power of ordinary switched networks. The networks of today are heterogeneous with a variety of computers with different power and communication speeds. The design of applications for parallel computing is greatly assisted by predictive models for estimation of communications costs. However these models are traditionally designed for the homogeneous systems, and no longer reflect the challenges presented by heterogeneous networks of computers.

We present a new intuitive performance model, the *LMO* model that is designed for both homogeneous and heterogeneous switched networks. The model is an innovative formulation that can map parts of parallel communication operations in a much more flexible way than traditional models to date. It has the advantage that it also includes empirical readings rather than a purely analytical approach as before. The model can adapt to each network and has highlighted previously uncharted regions of nonlinear response for some data sizes that give rise to severe performance degradation. Our model is unique in allowing the user to avoid such problem regions with its predictive assessment.

There are two key issues with a communications software model, its design and the estimation of its parameters. The new model for heterogeneity gives rise to a greater number of parameters that allow for its flexible mapping abilities. Previous models were limited to only a few parameters due to their simple estimation techniques. Therefore without suitable estimation methods the model has little applicability or use. We design an innovative estimation method and add efficiency with the calculations being performed in parallel. The method overcomes the limitations of the simple parameter estimation techniques of the past, these methods had restricted the design of the models to be non-intuitive and inflexible for the switched heterogeneous networks of today.

The ultimate usefulness of the predictive performance model is conditioned by the ability to accurately and efficiently measure the parameters of these models. Accurate timing methods are essential to ensure the performance model can accurately represent the overall execution time of the applications. Our first task was to address the shortcomings of current benchmarking methods. We address the problem of accurate and efficient timing of communications performance on a heterogeneous single switched network with a new software benchmarking library *MPIBlib*. This is a library that can be easily integrated with parallel applications.

We also present revised traditional models that we have extended to be heterogeneous. We implement a new software tool, the *CPM*, that allows the efficient and effective calculation process for the *LMO* model and the heterogeneous extended traditional models. The software provides an API for application developers that is easy to use and readily extensible in design. Finally we present some examples of applications that use the models we have developed. Using the *CPM* software we demonstrate the possibilities that these models present in assisting the applications designer to tune and optimize parallel computing applications.

Contents

List of Figures

Chapter 1

Introduction

Distributed systems of computers have with the advent of high-speed networks given rise to a platform of networks of computers. The power of parallel computing, that was previously restricted to specialist computers, can be applied to the widely available switched network systems. The traditional high-performance computing platforms made up of large dedicated systems of homogeneous processors can now be substituted by less expensive and widely available ordinary networks of computers. The heterogeneity of these clusters refers to networks of processors with computational diversity. These heterogeneous computational clusters have become more popular and accessible with the availability of low-cost computers.

Communication performance models are used by application developers to estimate speeds and execution times of parallel communications operations. The application can then be optimized and redesigned to allow for better communication cost. Many applications were originally designed for the more traditional homogeneous platforms used by parallel computing in the past. They used simple analytical models to predict performance of parallel applications. These models for prediction were designed for homogeneous systems with a uniformity that no longer exists with heterogeneous systems. The current models used to optimize the applications design are not fully representative of the diversity of the system and are therefore liable to be inaccurate in their estimation. This means that the efficiency of the application becomes less determined.

As today's networks can be made up of a variety of computers with different power and communication speeds, so the purpose of this thesis is to explore extending the current performance models from a point of view of heterogeneity, and to propose our own model and its innovative estimation methods as a better solution. Traditional models are limited in their abilities to accurately represent the parallel communications operations for a variety of reasons. The models are designed for specialist parallel systems of the past based on homogeneous

networks of dedicated processors. The models are too simplistic for new heterogeneous switched networks as they do not allow for adequate representation of MPI collective operations. This is because they are limited to a few simple parameters that cannot map the communications operations correctly or intuitively. The models are analytical in nature and cannot reflect the empirical irregularities that arise due to congestion for some message sizes.

The solution proposed in this thesis is a new heterogeneous communication performance model, *LMO*, (Lastovetsky, Mkwawa, O'Flynn) that allows for easy and intuitive expression of the execution time of collective operations, Lastovetsky *et al.* (2006a), Lastovetsky *et al.* (2006b), Lastovetsky & OFlynn (2007), Lastovetsky *et al.* (2009). We provide a software tool, *CPM*, Lastovetsky *et al.* (2008b), with innovative parameter estimation techniques, Lastovetsky *et al.* (2007) that allows for the deployment of heterogeneous extensions of traditional models and the new LMO model, for estimation and optimization of parallel communications. We also developed our own benchmarking software, *MPIBLib* Lastovetsky *et al.* (2008a), that is used for estimation of models and can be integrated as a library in parallel applications.

1.1 Project Scope

The platform for our performance model is a cluster of heterogeneous processors connected by a single network switch. Its key property is that processors in the cluster may not be identical, leading to heterogeneity. This is arguably the most common platform for parallel computing, with MPI (Message Passing Interface), Nagle (2005), Forum (2008), Gabriel *et al.* (2004), Gropp *et al.* (1996), as the principle programming system for parallel applications. MPI is a message-passing library interface specification, it is a language-independent communications protocol used to program parallel computers. Both point-to-point and collective communications are supported. The networks considered are Local Area Network (LAN) clusters of single processor computers. The network is connected by a switch with port-to-port connections between computers, with full duplex communications. We base our project on the most commonly available type of LAN, that is the single-switched network with computers of various speeds and power.

The prediction for parallel scientific computing applications uses a performance model to include the computation and communication costs. The subject of this thesis is only the communication costs, computational costs are beyond the scope of our modeling process. The model is for estimation of application level programming with communication operations across the switched cluster. The accuracy of this model depends on the accurate assessment of model's pa-

rameters. In this thesis the current models are revised and extended, and a new model is proposed, as well as new methods of assessing the parameters of the communication performance model.

1.2 A New Heterogeneous Communication Performance Model

Traditionally, communication performance models for high performance computing are analytical and built with the assumption that the processors of the cluster are homogeneous. There is another problem with these models however, and that is a lack of intuitiveness that is very significant, as these models need to be easy to use for applications developers.

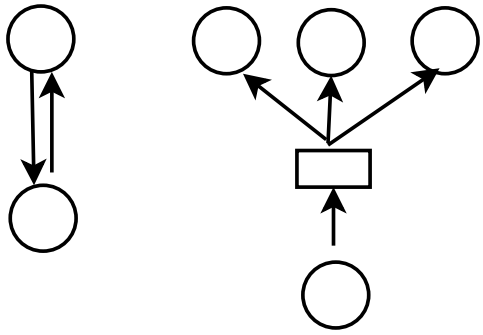


Figure 1.1: Point-to-point versus a switched network collective operation

Traditional communication performance models have a small number of parameters and are designed for homogeneous transfer. They provide a simple representation of point-to-point operations, but they do not intuitively represent the parallel collective communications in an easy way. In Figure 1.1 we can see how the simple point-to-point mapping differs from a more complex mapping to represent a collective communications operation over a switched network. The collective operation is in fact a serial communication of messages from the root processor that then becomes a parallel communication of messages from the switch to each destination node. A simple point to point representation from traditional models cannot map this combination of message transfers for collective operations with MPI on a switched network. This is a critical problem leading to significant inaccuracies that our model is designed to overcome. It separates the contributions of different origin, that is *constant* and *variable* parts of the communication. This allows the collective operations to be better modeled with our extra parameters in a clearer and more accurate way.

The basis of the traditional models is a set of integral point-to-point parameters, having the same value for each pair of processors. The execution time of other operations (which are, in fact, collective), is then expressed by the model as a combination of the point-to-point parameters, and is analytically predicted for different message sizes and numbers of processors involved. The core of this approach is the choice of such a point-to-point model that then allows a combination of parameters to be an expression of the different algorithms of collective operations. For homogeneous clusters, the point-to-point parameters are found statistically from communication experiments between any two processors. A communication model can be seen as consisting of two key issues, the design of the model itself and the method of its estimation. The analytical communication performance models currently available are limited in their design by their method of accurate estimation of its parameters on targets platforms. These models are restricted to a few simple parameters since only this small number of parameters can be estimated using traditional methods of point-to-point communication experiments. Thus the estimation part of the model is very important and can exert significant influence on the design.

The accuracy of the analytical prediction provided by a communication performance model depends on how easy and natural the execution time of collective operations can be expressed via a combination of the model's parameters. An ideal intuitive communication performance model for a homogeneous or heterogeneous cluster with a single switch should have the following features:

- It is based on the point-to-point parameters that reflect *constant* and *variable* contributions of *processors* and *network*. *Note:* The *constant* contribution of the processors and network is the fixed communication costs found using zero sized message transmission, while the *variable* contributions are those additional costs both from the processor and network as a result of increasing message size. This is important as the full separation of the contributions of a different nature that arise from different sources will lead to more intuitive analytical expressions of the communication execution time. In contrast, for the traditional communication performance models, the constant and variable contributions of processors and network are not fully separated.

- The execution time of any collective communication operation can be represented by a combination of *maximums* (parallel part) and *sums* (sequential part) of the point-to-point parameters. The formula of the execution time should also include *empirical* parameters that reflect possible *irregular* behavior of the collective operation. Current models are analytical only and they assume a linear response in execution time to message size. Non-linear

responses in execution time may be found empirically for a particular platform. Traditional models do not include such empirical parameters that can reflect these changes.

- There is *a set of communication experiments* that allows for the *accurate estimation of the parameters*. Traditional models are designed so that their parameters can be estimated from point-to-point communication experiments. The attempts to separate the contributions lead to a model whose parameters cannot be estimated from the point-to-point experiments only. We outline some of the most popular traditional models, and propose their straightforward heterogeneous extensions.

The above criteria form a problem definition and specification for our new model, Lastovetsky *et al.* (2009).

We demonstrate the importance of estimation methods as the main factor limiting the expressive power of the traditional communication models. They do not allow for parameters that can represent the *constant* and *variable* contributions separately, originating from different sources in the communication cost. To achieve this goal there is an innovative estimation strategy to estimate beyond the standard point-to-point communication experiments. Therefore the estimation of such communication models is a new and non-trivial problem. This thesis proposes a solution of this problem, as well as the intuitive and practical performance *LMO* model. The idea is to introduce additional collective communication experiments involving more than two processors. To make use of the results of these additional independent experiments, we proposed to extend the heterogeneous point-to-point communication model by a model of these collective operations. We can then use this extended model to obtain the additional independent equations for the estimated parameters.

The heterogeneous models have a larger number of parameters and will require a significantly larger number of measurements. We can address this problem by performing most of the communication experiments in parallel, using the fact that the network switches provide no-contention point-to-point communications, appropriately forwarding packets between sources and destinations.

We also show how traditional models may be extended, Lastovetsky *et al.* (2009), to the heterogeneous switched network, and how our new model compares favorably with these attempts. Chapter 5 describes our heterogeneous versions of traditional models, this work is significant as to the best of the author's knowledge, there are no publications by others describing heterogeneous extensions of communication performance models. We determine new heterogeneous versions of the traditional homogeneous models, Hockney and PLogP. We demonstrate with experimental results that the LMO model is a more accurate predictor of

the execution time of collective operations than traditional models, even when extended for a heterogeneous switched network. The heterogeneous versions of current models are an improvement on their homogeneous counterparts, but our new LMO model is a far more versatile and accurate approach to the problem of estimation of performance for MPI collective communications.

1.3 A New Benchmarking Library

The use of advanced intuitive heterogeneous communication models for optimization of communication operations has a potential to improve their performance (and hence the performance of the corresponding applications) on heterogeneous computational clusters. However, utilization of this potential is conditioned by the ability to accurately and efficiently estimate the parameters of these models. Given the parameters are estimated inaccurately, implementation of the optimization algorithms in applications running on heterogeneous clusters will not have a positive effect on their performance. Accurate timing methods are therefore important to ensure an accurate performance model to estimate the overall execution time of the applications. Our first task was to address the shortcomings of current benchmarking methods. We designed a new integrative library, the *MPIBlib*, Lastovetsky *et al.* (2008a), for benchmarking as a solution.

The first problem with current benchmarking tools is that they restrict the user to one choice of timing method, our library provides a choice of timing methods to allow the user to select the most suitable for their needs. This is particularly relevant when the user wishes to tune applications at run-time for optimization of application performance. The second limitation is that previous benchmarking software is designed in the form of a standalone executable programs, there is a need for a benchmarking library that can be used and integrated easily and efficiently into application-level software.

Our new approach to benchmarking software addresses these limitations and offers a choice of timing that can be easily integrated with the application as a simple software library. This library is used as a basis to estimate parameters for our new model more accurately, and so assist its effectiveness as a performance estimator for parallel communications.

1.4 A Software Tool for Accurate Estimation of Heterogeneous Communication Performance Models

The goal of our work is to facilitate and assist applications designers with new heterogeneous models, therefore we present a software tool for easy deployment of the performance models. We designed and implemented this software tool to estimate our model and also our heterogeneous extended versions of the traditional models, Lastovetsky *et al.* (2008b),Lastovetsky *et al.* (2009). We described the design of the *CPM* (Communication Performance Modeling) software that automates the estimation of parameters of both traditional and the heterogeneous communication performance models, including our new *LMO* model. The software assists the designer by automating the estimation process for the model parameters, the heterogeneous models having a greater number of parameters than traditional homogeneous approaches. Our research has developed new estimation techniques, these new methods make the new *LMO* model possible, and allow a model to be built with much greater intuitive mapping of collective operations. The model building is also completely automated by the software. The process includes detection of platform specific features such as possible performance irregularity and performance degradation that found from empirical readings. The applications designers do not need to know of or adjust the underlying system, as the software will automatically find the model's representation on their behalf.

The *CPM* software tool consists of both a library and command-line utilities. The library provides an API for measurement of the execution time of communication experiments (Measurement Module) and for estimation of models and prediction of the execution time of collective algorithms with the models (Models Module). The library is implemented in C/C++ on the top of MPI with help of some third-party software. The third-party software is mainly used for statistical and regression analysis, and for estimation of traditional homogeneous models.

1.5 Model-based Optimization of Collective Algorithms

Our final area of new work is with the use of the models, and we present the *CPM* software tool with its use for optimization. The use of analytical predictive communication models in the design of applications with MPI collective operations can significantly improve their performance on homogeneous and heterogeneous clusters. We showed how our software is used to improve applications using collective operations. Our research has found from empirical observations of some

platforms that significant non-deterministic escalations of the execution time for medium-sized messages within some range were observed for different platforms and MPI implementations. The new *LMO* communication performance model is the only model that can address this issue, and allows the performance degradation from the non-deterministic escalations of the execution time to be avoided with application design adjustment. The software tool has implemented a choice of traditional models that have been extended to be heterogenous, as well as the new *LMO* model. We show by a number of examples how the models may be used to optimize MPI collective communications with different parallel applications.

1.6 New Work

This thesis presents the following new work to accurately assess the performance of collective communications for applications software on parallel systems:

- An accurate model depends on accurate measurement techniques. We address the problem of accurate and efficient timing for benchmarking of communications performance on a heterogeneous single switched network with a new library *MPIBlib*, Lastovetsky *et al.* (2008a). This is a benchmarking library that can be used in parallel applications for the accurate estimation of communication operations execution times. The *MPIBlib* can be linked to other applications and used at runtime. The library is used as the basis for the accurate estimation of performance model parameters.

- We design a new intuitive performance model, the *LMO* model that is designed for heterogeneous switched networks. The model is an innovative formulation that separates *constant* and *variable* parts of collective communication to allow for a new direct way of mapping the communications form. It is based on empirical readings that are adapted to each network for a new range of message sizes, previously uncharted, Lastovetsky *et al.* (2006a), Lastovetsky & OFlynn (2007), Lastovetsky *et al.* (2009). A model for heterogeneity gives rise to a greater number of parameters. We extend the traditional point-to-point estimation methods with innovative estimation methods. We add efficiency to the measurments with the option of calculations to be performed in parallel.

- We present revised traditional models extended to be heterogeneous, this work is previously unpublished by other authors to the best of our knowledge, Lastovetsky *et al.* (2009).

- We implement a new software tool, *CPM*, Lastovetsky *et al.* (2008b), that allows the efficient and effective calculation process for the *LMO* model and

heterogeneous extensions of traditional models. These include traditional Hockney-based and PLogP-based collective operations where the models are extended to be heterogeneous, and the new *LMO* model collective operations, Lastovetsky *et al.* (2007), Lastovetsky *et al.* (2009). The software provides an API for application developers that is easy to use and readily extensible in design.

1.7 Thesis Structure

This thesis is organized with the three main sections - the benchmarking software *MPIBlib* library to measure model parameters, the new *LMO* performance model and extended traditional models, and finally optimization using the models. First we describe the 'Related Works' in current research to date in Chapter 2. The chapter outlines the related work in the key three areas, beginning with timing and benchmarking methods, and then describing the traditional communication performance models and then the former work in the optimization of communication with models. In Chapter 3 we begin the main body of the thesis, the new benchmarking library software is presented. We demonstrate the *MPIBlib* library of routines for the measurement of execution time that can be integrated with parallel applications for communication performance modeling and the tuning communication operations. This benchmarking software is the basis for accurate measurement of parameters of the models, the next topic of the thesis. Chapter 4 presents the *LMO* model, a new model designed for heterogeneity for the assessment of execution time of MPI collective communications. The chapter also discusses the innovative estimation methods that are used to find the additional parameters of the heterogeneous model. In Chapter 5 we take the current popular traditional models and present our own heterogeneous versions of these models, extending the homogeneous models to a heterogeneous environment. Chapter 6 describes the new software tool, the *CPM*, that automates the estimation of the heterogeneous communication performance models. The last topic is how the models may be used, and therefore Chapter 7 shows how the models and the software tool are used to optimize MPI collective operations in parallel applications. The examples include using the implementation of the traditional model software extended to be heterogeneous. There are also further examples of an application using the new *LMO* model and the *CPM* software library tool. Finally conclusions are presented in Chapter 8.

Chapter 2

State of the Art

Three main areas of Related Works concerning the timing estimation by communication performance models are described in this chapter. The first relates to the implementation of our new software tool for benchmarking methods of measurement. The chapter begins by explaining the importance of the role that accuracy plays in the estimation of the execution time of MPI communication operations. The current benchmarking suites are discussed and compared to highlight areas of potential improvement.

The second area of innovative research in the thesis is that of a new performance model for MPI communications, therefore we give an overview of the current models that are used to date. We review the current models for collective MPI operations that allow a software developer to design a parallel application for better performance. By examining the limitations of current models we define a basis of criteria for design of the new model.

The third area of research reviewed is the use of the models and model-based optimization of MPI collective communication operations. The final section of this chapter describes current work in this area.

2.1 Benchmarking MPI Communications

Accurate estimation of the execution time of MPI communication operations plays an important role in optimization of parallel applications. A priori information about the performance of each MPI operation allows a software developer to design a parallel application in such a way that it will have maximum performance. This data can also be useful for tuning collective communication operations and for the evaluation of different available implementations. The choice of collective algorithms becomes even more important in heterogeneous environments. Before considering the performance models however, we examine

the timing methods and benchmarking they may use, as the accuracy and efficiency of the model depends directly on the measurements that are taken in the first place. MPI benchmarking suites use different timing methods to estimate the execution time of the MPI communications. Each of these methods provides a certain accuracy and efficiency. The efficiency of the timing method is particularly important in self-adaptable parallel applications using runtime benchmarking of communication operations to optimize their performance on the executing platform. In this case, less accurate results can be acceptable in favor of a rapid response from the benchmark.

Most of the MPI benchmarking suites are designed in the form of a standalone executable program that takes the parameters of communication experiments and produce a lot of output data for further analysis. As such, they cannot be integrated easily and efficiently into application-level software.

In this section, different timing methods are described, and the existing benchmarking suites are reviewed. We begin with a discussion of what timing methods mean and then describe the benchmarking suites currently available. We summarize their limitations, and this provides a basis for design of the MPIBlib benchmarking suite presented in the next chapter.

2.1.1 Timing Methods

In order to evaluate the accuracy of the estimation given by different suites, we first provide a unified definition of the execution time and we suggest the following as a natural definition.

The execution time of a communication operation is defined as the real (wall clock) time elapsed from the start of the operation, given all the participating processors have started the operation simultaneously, until the successful completion of the operation by the last participating processor.

Mathematically, this time can be defined as the minimum execution time of the operation, given that the participating processors do not synchronize their start and are not participating in any other communication operation. It is important to note that the definition assumes that we estimate the execution time for a single isolated operation.

Estimation of the execution time of the communication operation includes:

- Selection of two events marking the start and the end of the operation respectively.
- Measuring the time between these events.

First of all, the benchmarking suites differ in what they measure, which can be:

- The time between two events on a single designated processor.
- For each participating processor, the time between two events on a processor.
- The time between two events, but on different processors.

The first two approaches are natural for clusters as there is no global time in these environments where each processor has its own clock showing its own local hour. The local clocks are not synchronized and can have different clock rates, especially in heterogeneous clusters. The only way to measure the time between two events on two different processors is to synchronize their local clocks before performing the measurement. Therefore, the third approach assumes the local clocks to be regularly synchronized. Unlike the first two, this approach introduces a measurement error as it is impossible to keep the independent clocks synchronized all the time with absolute accuracy.

In order to measure time, most of packages rely on the *MPI_Wtime* function. This function is used to measure the time between two events on the same processor (the local time). For example, the execution time of a roundtrip can be measured on one process and used as an indication of the point-to-point communication execution time, Int (2007), and Worsch *et al.* (2002).

The execution time of a collective communication operation can also be measured at a designated process. For collective operations with a root, the root can be selected for the measurement. As for many collective operations the completion of the operation by the root does not mean its completion by all participating processes, short or empty messages can be sent by the processors to the root to confirm the completion. A barrier, reduce, or empty point-to-point communications can be used for this purpose. The final result must be corrected by the average time of the confirmation. The drawback of this approach is that the confirmation can be overlapped with the collective operation and hence it cannot simply be subtracted. As a result, this technique may give negative values of the execution time for very small messages. The accuracy of this approach (***root timing***) is strongly dependent on whether all processes have started the execution of the operation simultaneously. To ensure the more or less accurate synchronization of the start, a barrier, reduce, or empty point-to-point communications can be used. They can be overlapped with the collective operation to be measured and previous communications as well. To achieve even better synchronization, multiple barriers are used in the benchmarking suites Int (2007), Grove & Coddington (2001), and Worsch *et al.* (2002).

The local times can be measured on all processes involved in the communication and the maximum can be taken as the communication execution time.

This approach ***(maximum timing***)is also dependent on synchronization of the processes before communication, e.g. with a barrier.

To measure the time between two events on different processors, the local clocks of the processors have to be synchronized. Such synchronization can be provided by the MPI global timer if the *MPI_WTIME_IS_GLOBAL* attribute is defined and is true. Alternatively, local clocks of two processors A and B can be synchronized by the following simple algorithm (implemented in MPIBench Grove & Coddington (2001)). Processor A sends a message to processor B, which contains the current time, plus half of the previously observed minimum roundtrip time. Processor B receives the message and returns it to A, which calculates the total time that the roundtrip took to complete. If the roundtrip time is the fastest observed so far, then the estimated time of arrival of the initial message is the most accurate yet. If so, processor B calculates the current approximation of the time offset as the message's value received in the next iteration. The processors repeat this procedure until a new minimum roundtrip time has not been observed for a prearranged number of repetitions. Given A being a base processor, this synchronization procedure is performed sequentially for all pairs *(A, B)*. A similar procedure is implemented in SKaMPI, Worsch *et al.* (2002), to find offsets between local times of the root and the other processes.

As local clocks can run at different speeds, especially in heterogeneous environments, the synchronization has to be regularly repeated. These synchronization procedures are quite costly and introduce a significant overhead in benchmarking when used. As soon as the global time has been set up, the time between two events on different processors can be measured, Grove & Coddington (2001), Worsch *et al.* (2002). The accuracy of this approach will depend on the accuracy of the clock synchronization and on whether processors start the communication simultaneously. The ***global timing*** usually gives a more accurate estimate because its design is closer to the natural definition of the communication execution time given in the beginning of the first chapter. However, as well as being more time-efficient, the other methods based on local clocks can also provide quite accurate results for many popular platforms and MPI implementations.

In some cases the ***operation-specific methods*** may work faster than their universal counterparts, and can be used as time-efficient alternatives. The efficiency of timing methods is particularly important in self-adaptable parallel applications that optimize their performance using runtime benchmarking of communication operations. Some particular collective operations and their implementations allow for the use of more accurate and efficient methods that cannot be applied to other collective operations. One example is the method of measurement of linear and binomial implementations of the MPI broadcast on heterogeneous platforms proposed in Supinski & Karonis (1999). It is based on measuring individual tasks rather than the entire broadcast and therefore it does not need the

global time. An individual task is a part of the broadcast communication between the root and the *i-th* process. In each individual task, the broadcast is followed by sending an acknowledgement message from the *i-th* process to the root. The execution time of the task is then corrected by the value of the point-to-point execution time.

Practically, the execution time of the communication operation is estimated from the results of an experiment that in addition to the operation, includes other communications and computations. As parallelism introduces an element of non-determinism, there is a problem of reproducibility of such experiments. The methodology of designing reproducible communication experiments is described in Gropp & Lusk (1999). It includes:

- Repeating the communication operation multiple times to obtain the reliable estimation of its execution time,
- Selecting message sizes adaptively to eliminate artifacts in a graph of the output of the communication operation, and
- Testing the communication operation in different conditions: cache effects, communication and computation overlap, communication patterns, non-blocking communication, etc.

To obtain a statistically reliable estimate of the execution time, a series of the same experiments are typically performed in the benchmarking suites. If the communications are not separated from each other in this series, the successive executions may overlap, resulting in a so-called pipeline effect (Bernaschi & Iannello (1998)), when some processes finish the current repetition earlier and start the next repetition of the operation before the other processes have completed the previous operation. The pipeline affects the overall performance of the series of the operations, resulting in inaccurate averaged execution time.

In order to find the execution time of a communication operation that is not distorted, it should be measured in isolation from other communications. A barrier, reduce, or point-to-point communications with short or empty messages can be used between successive operations in the series. The approach with isolation gives results that are more accurate.

2.1.2 MPI Benchmarking Suites

There are several commonly used MPI benchmarking suites **mpptest** (Gropp & Lusk (1999)), **Netpipe** (Turner *et al.* (2003)), **IMB** (Int (2007), **MPIBench** Grove & Coddington (2001), and **SkaMPI** Worsch *et al.* (2002)). Some of them include tests for collective operations.

In the **mpptest** suite, Gropp & Lusk (1999), a suite of performance measurement programs is developed for MPI benchmarking of point-to-point communications in a reproducible way. The execution time of communication operations depends on the MPI library, native software, and hardware configurations. **NetPIPE** (Turner *et al.* (2003)) provides benchmarks for different layers in the communication stack. It is based on the ping-pong communication experiments that are implemented over **memcpy**, TCP, MPI etc. In addition to evaluation of communication performance, this suite helps to identify where inefficiencies may be.

Regarding both the reproducibility of communication experiments and the dependency on communication layers, we focus on benchmarking not only point-to-point operations but also collective ones. Therefore, we analyzed several MPI benchmarking suites that include tests for collective operations (**IMB, MPIBench** and **SkaMPI**). Despite the different approaches to what and how to measure, they have several common features:

- Computing an average, minimum, maximum execution time of a series of the same communication experiments to get accurate results;
- Measuring the communication time for different message sizes – the number of measurements can be fixed or adaptively increased for messages when time is fluctuating rapidly;
- Performing simple statistical analysis by finding averages, variations, and errors.

The MPI benchmarking suites are also very similar in terms of the software design. Usually, they provide a single executable that takes a description of communication experiments to be measured and produces an output for plotting utilities to obtain graphs.

As more than two processors are involved in collective communications and connected in different ways (communication trees), there are two main issues concerned with the estimation of execution time of MPI collective operations:

- Measuring the execution time, and
- Scheduling the communication experiments.

In the next sections we describe the **IMB, MPIBench** and **SkaMPI** benchmarking suites in more detail.

2.1.2.1 Intel MPI Benchmarks

IMB (Int (2007)) is widely used in parallel processing to measure the performance of important MPI functions. Benchmarks are written in ANSI C using a message-passing paradigm comprising 10,000 lines of code. The IMB 2.0 version has three parts (a) IMB for MPI-1, (b) MPI-2 one sided communication, and (c) MPI-2 I/O. In standard mode, the message size can be set to *0,1, 2, 4, 8,... 4194304 bytes*. There are three classes of benchmarks, namely single transfer, parallel transfer and collective benchmarks.

The **IMB** measures the communication execution times locally on each process, and the minimum, maximum, and average times are then returned. The communication experiments in a series are not isolated. Figure 2.2 shows the results returned by the **IMB** on a 16-node heterogeneous cluster for scatter and gather operations when single and multiple repetitions are used in the experiments. One can see that for the scatter experiments with a single repetition, the minimum time represents the execution time of a non-blocking send on the root and is therefore relatively small. In the gather experiments with a single repetition, the maximum time is observed on the root, reflecting the communication congestion. The difference between the minimum and maximum times decreases with an increase in the number of repetitions. In both cases, we observe a clear impact of the pipeline effect on the measured execution time of the operation:

- *Scatter*: For small and large messages, the execution time of a repetition in the series is smaller than that measured in a single experiment. For medium-sized messages, escalations of the execution time are observed that do not happen in single experiments.

- *Gather*: Escalations of the execution time for medium-sized messages, observed for single experiments, disappear with the increase of the number of repetitions due to the pipelining.

Thus, the pipeline effect can significantly distort the actual behavior of the communication operation, given that we are interested in accurate estimation of the time of its single and isolated execution.

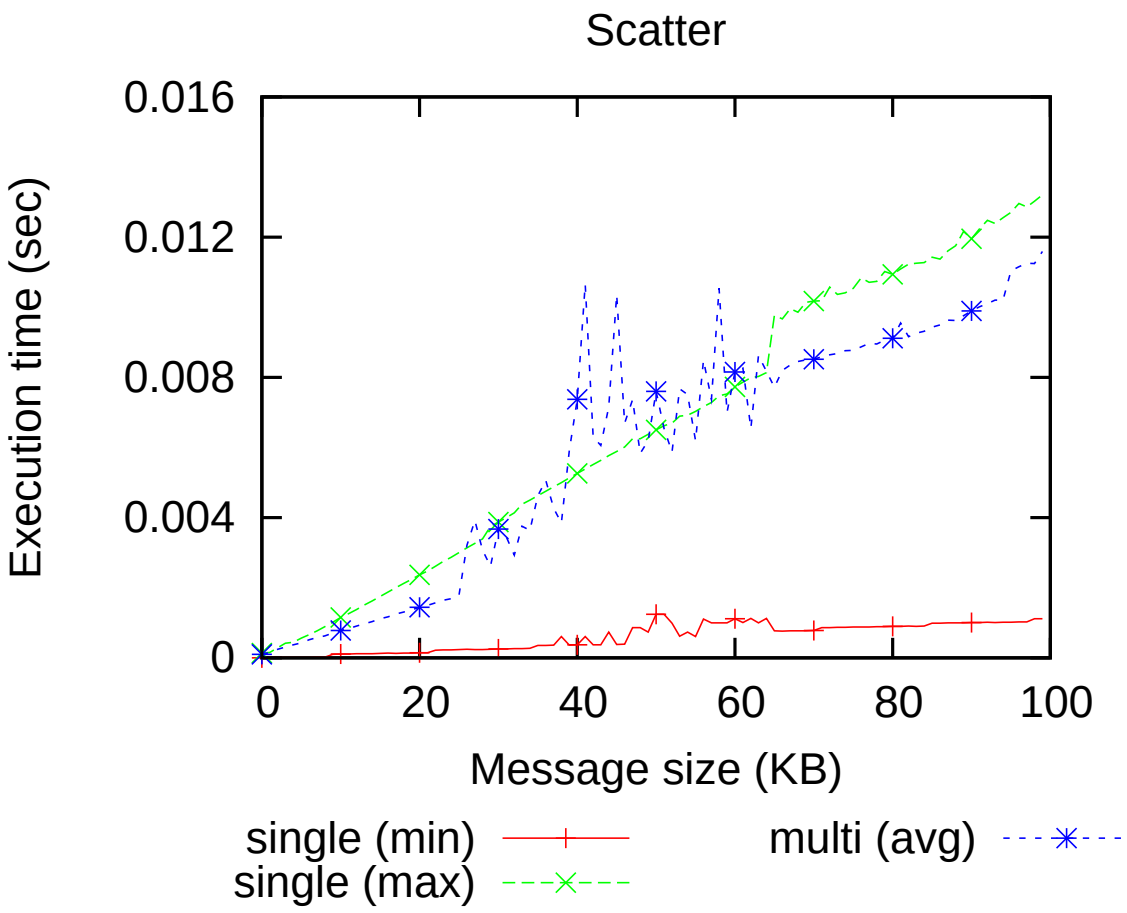


Figure 2.1: IMB benchmark on a 16-node heterogeneous cluster: single/multiple scatter measurements

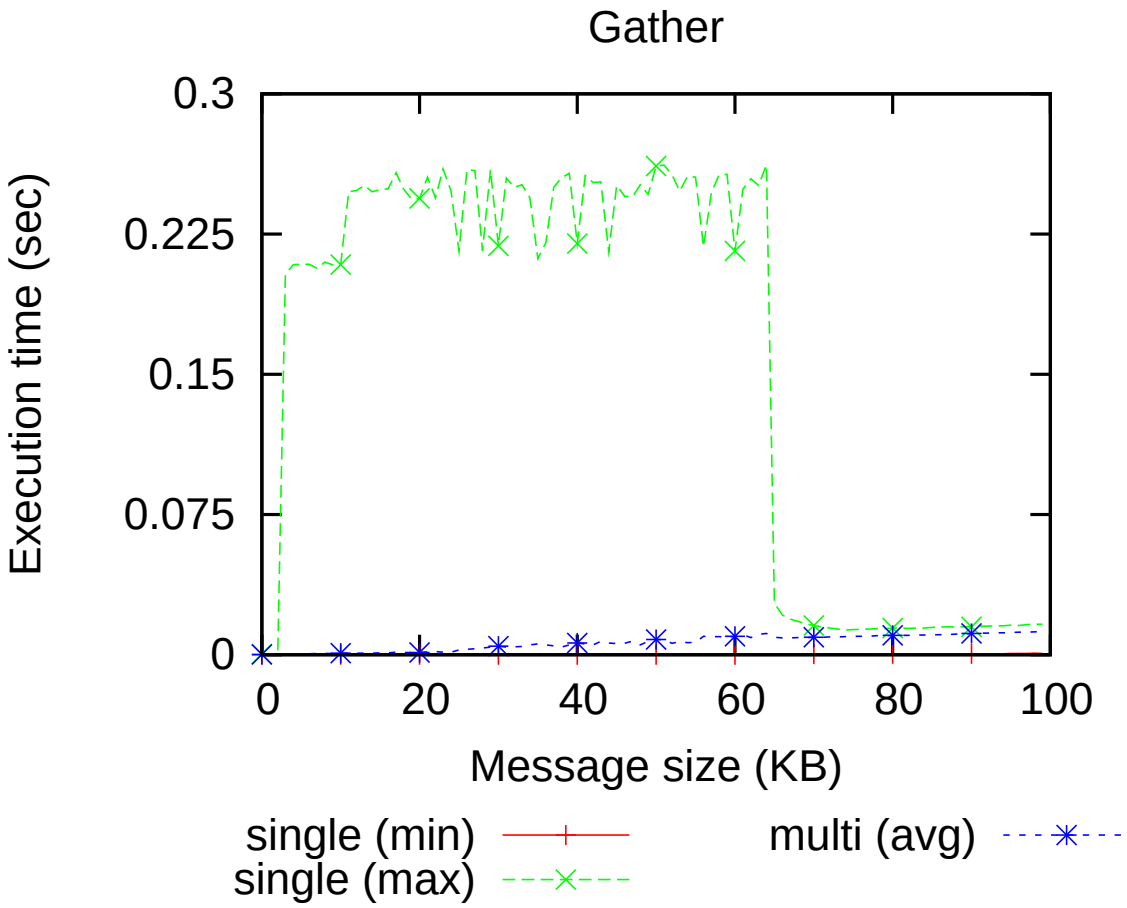


Figure 2.2: IMB benchmark on a 16-node heterogeneous cluster: single/multiple gather measurements

2.1.2.2 MPIBench

The MPI benchmarking package called **MPIBench** (Grove & Coddington (2001)) overcomes some of the drawbacks of other existing MPI benchmark tools. It uses a portable and accurate global clock to obtain timing data on many individual MPI communications calls in order to produce timing distributions that show the performance variability of these routines. **MPIBench** has built-in support for measuring the end-end and local completion times of the most common message passing primitives such as *MPI_Send, MPI_Isend, MPI_Recv, MPI_Sendrecv, MPI_Bcast* and *MPI_Reduce*. The compound communications patterns can be easily inserted within the timing framework. The main limitations of **MPIBench** are that it is a stand-alone form rather than a library that can be integrated with applications software. The use of a global clock may be too sophisticated and inefficient for simpler user requirements at run-time.

2.1.2.3 SkaMPI

SkaMPI software by Worsch *et al.* (2002) is a publicly available benchmarking tool that is a database providing performance data for operations of MPI that can be measured on several different platforms. **SKaMPI** is comprehensive as it covers most of MPI including point-to-point communication collective communications derived datatypes one-sided communication. **SKaMPI** can be configured and tuned in many ways - operations, measurement precision, communication modes, packet sizes, number of processors used etc. The **SKaMPI** benchmark package consists of three parts:

- The benchmarking program itself,
- A postprocessing program,
- A report generation tool.

For ease of portability the benchmarking and postprocessing program are both ANSI C programs. The report calls gnuplot and LaTeX. The *.skampi* runtime parameter file describes all measurements with specific parameters specified by the user in order to identify the measurement configuration. Many other customizations are also possible without changing the source code. The benchmark program produces an ASCII text file *skampi.out* in a documented format. The adjustable report generator reads the output file and generates a postscript file containing a graphical representation of the results

SKaMPI is easy to compile, once any MPI application is run with it repeatedly it uses a human readable configuration file to select the measurements to

be get readable benchmark results. **SKaMPI** is extensible, if there are missing measurements for a certain MPI feature, they may be added with some C programming.

The performance measurement of MPI operations using **SkaMPI** while designing programmes allows the software developer:

- To select the fastest implementation,
- To write performance portable software, that is showing high performance on several platforms without platform-specific tuning, and
- To quantify the tradeoff between performance and portability for different platforms.

The software uses a simple ping-pong experiment, with repetition for accuracy.

The main limitations are similar to **MPIBench** in that it is a stand-alone form rather than a library that can be integrated with applications software. A large amount of data is generated that is not easily assimilated by applications software. The use of a global clock may be too inefficient for some networks that require a quicker more efficient solution provided by simpler timing methods.

2.1.3 Summary

The existing MPI benchmarking suites that are currently available have several significant restrictions that prevent them from a wider use in applications and programming systems. In summary, the following limitations need to be overcome with the current benchmarking suites:

- They restrict the user to one choice of timing method. A choice of timing methods provides a balance of accuracy and efficiency. The efficiency of timing methods will be particularly important in self-adaptable parallel applications using run-time benchmarking of communication operations to optimize their performance on the executing platform.
- They are designed in the form of a standalone executable program and cannot be integrated easily and efficiently into application-level software. Therefore, there is a need for a benchmarking library that can be used in parallel applications or programming systems for communication performance modeling and tuning communication operations.

We developed a benchmarking library, called *MPIBlib*, described in Chapter 3 that provides a wide range of efficient methods of measurement of MPI communication operations, both universal and operation specific. The variety of methods

available with the library allows users to optimize the cost of benchmarking by choosing the most efficient method for a given operation and a required accuracy of estimation. In contrast to existing MPI benchmarking suites, implemented as standalone applications together with some plotting scripts for graphical representation of results, *MPIBlib* is designed as a library that can be used as an integral part of parallel applications.

2.2 Traditional Communication Performance Models

The communication performance model plays a pivotal role in the design and optimization of applications using MPI collective communications. This section reviews the current communication performance models and compares their merits, first we introduce ideas forming the basis for their most significant design limitations.

The basis of these models is a set of integral point-to-point parameters, based on the homogeneous network. The values of parameters are found from point-to-point experiments between the homogeneous processors, so each pair has the same values. Typical experiments include sending and receiving messages of different sizes, with the communication execution time being measured on one side. The models are an analytical prediction for different message sizes and numbers of processors involved. The execution time of collective operations are expressed as a combination of these point-to-point parameters. The approach has shortcomings when considering the heterogeneity of the network with processors of different types. The homogeneous communication model can be applied to a cluster of heterogeneous processors by averaging values obtained for every pair of processors. If some processors or links in the heterogeneous cluster significantly differ in performance, predictions based on the homogeneous communication model may become inaccurate.

One limitation of the traditional communication performance models, preventing them from the accurate prediction of the execution time of collective communication operations on homogeneous and, especially, heterogeneous clusters, is that they combine the contributions in the execution time that have different nature and arise from different sources. Therefore, they do not allow for intuitive analytical expressions of the execution time of the most of efficient algorithms of collective communication operations. In the case of heterogeneous clusters, the intuitive models are of the utmost importance as the heterogeneous communication performance models have much larger a number of parameters. If the most of these parameters are not intuitive then the model will be absolutely useless. There is general lack of software automating the estimation of commu-

nication performance models. The *logp_mpi* library Kielmann *et al.* (2000) is a rare exception. It estimates the PLogP parameters for a pair of processors and, therefore can only be used directly for homogeneous platforms.

2.2.1 Hockney Model

Let us start with a traditional model proposed by Hockney (1994). The parameters of the Hockney model combine the processor and network contributions. The execution time of point-to-point communication is expressed as $\alpha + \beta M$, where α is the latency (constant contributions from processors and network), β is the bandwidth (variable contributions from processors and network) and M is the message size. The Hockney parameters are estimated with help of series of the point-to-point communications in one of two ways:

- Two series of roundtrips with empty messages (to get the latency parameter from the average execution time), and with non-empty ones (to get the bandwidth), or
- A series of roundtrips with messages of different sizes (to perform a linear regression, which fits the execution time into a linear combination of the Hockney parameters and a message size).

The Hockney model is the simplest of traditional models that uses only two parameters to describe communication between two processors. These parameters represent an accumulation of the contributions of the processors timings together with the communication layer timings. This makes it non-intuitive to model the communication operations for the single-switched platform, where the operation is split between a serialization and parallel transfer of messages.

2.2.2 LogP Model

The LogP model, Culler *et al.* (1993), is a more elaborate model that predicts the time of network communication for small fixed-sized messages in terms of the latency, L, the overhead, o, the gap per message, g, and the number of processors, P. The latency, L, is an upper bound on the time to transmit a message from its source to destination; it reflects the constant contribution of network. The overhead, o, is the time period during which the processor is engaged in sending or receiving a message (a constant processor contribution). The gap, g, is the minimum time between consecutive transmissions or receptions; it is the reciprocal value of the end-to-end bandwidth between two processors, so that the network bandwidth can be expressed as L/g. According to LogP, the time of point-to-point communication can be estimated by $L + 2o$. The LogP model

assumes that a large message is decomposed to a series of short messages. In the formula for a series the gap parameter will be used: $L+2o+Mg$. Therefore the gap can be attributed to the variable contributions of processors and the network.

The method of estimation of the LogP parameters is presented in Culler *et al.* (1996), with the sending time, o_s, and receiving time, o_r, so that the overheads being sent and received are distinguished. The set of experiments used for estimation of the LogP parameters is as follows:

- Finding o_s, a small number of messages are sent consecutively in one direction in order to estimate the sending overhead parameter. The average time is then measured on the sender side for the value of o_s.
- Finding receiving overhead, o_r, directly from the time of receiving a message in the roundtrip between source and destination. There is a time delay at the sender side to allow for the posting of each receive, and the average of the receive operation is o_r.
- The latency is found from the execution time of the roundtrip with a small message: $L = RTT/2 - o_s - o_r$.
- To find the gap parameter, g, a large number of messages are sent in one direction consecutively, that is one-direction consecutive sendings with a confirmation, and g is estimated as $(i \xleftarrow[0]{\overbrace{M \ldots M}^{2^x}} j)$ with $g = T_s/s$, where $s = 2^x$ is a number of messages sent and T is the total execution time measured on the sender processor.

The number of messages sent consecutively is chosen to be large to ensure that the communication time is dominated by the factor of bandwidth rather than latency. This saturation method has the disadvantage that it takes a long time. In contrast to the Hockney model, LogP is not designed for the communications with arbitrary message sizes, so there are some derivatives, such as the LogGP and PLogP model, that address this issue next.

2.2.3 LogGP

The LogGP model by Alexandrov *et al.* (1995), an extension of LogP, takes into account the message size by introducing the gap per byte parameter, G. The point-to-point communication time is estimated by $L + 2o + (M - 1)G$. The original gap parameter, g, is also used in the model to represent the delays between consecutive communications. For example, the execution time of m

sendings of M bytes is estimated as follows: $L + 2o + (m-1)G + (m-1)g$. The g gap parameter combines the contributions of processors and network, the G parameter is network only. The gap g represents the constant and variable contribution, while the gap per byte G represents the variable contribution.

2.2.4 PLogP Model

In the PLogP (parameterized LogP) model by Kielmann *et al.* (2000), all parameters except for latency are piecewise linear functions of the message size, and the meaning of parameters differs slightly from LogP. The meaning of latency, L, is not intuitive; it is a constant that combines all fixed contribution factors such as copying to/from the network interfaces and the transfer over the network. The send, $o_s(M)$, and receive, $o_r(M)$ overheads are the times that the source and destination processors are busy for the duration of communication (variable contributions of processors). They can be overlapped for sufficiently large messages. The gap, $g(M)$, is the minimum time between consecutive transmissions or receptions; it is the reciprocal value of the end-to-end bandwidth between two processors for messages of a given size M. The gap is assumed to cover the overheads ($g(M) \geq o_s(M), g(M) \geq o_r(M)$) and represents mixed processor-network variable contributions. According to the PLogP model, the point-to-point execution time is equal to $L + g(M)$.

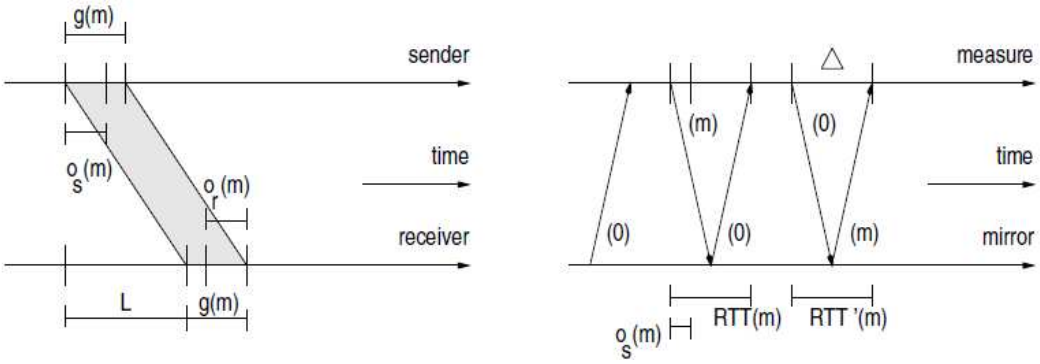


Figure 2.3: Message transmission as modeled by parameterized LogP (left); fast measurement procedure (right) Kielmann *et al.* (2000))

The times for sending and receiving a message of size m, $s(m)$ and $r(m)$, are determined from when both sender and receiver simultaneously start their operations. The $s(m)=g(m)$ is the time at which the sender is ready to send the next message; $r(m)=L+g(m)$ is the time at which the receiver has received the

message, here the latency and the gap added denotes the time a message occupies the network. The latency L is the time it takes for the first part of a message to travel from sender to receiver.

The message gap is the time after this until the last bit of the message has been received, see Figure 2.3 (left). The parameters of the LogP-based models are estimated by more complicated point-to-point experiments. In addition to roundtrips ($i \xleftarrow[0]{M} j$ and $i \xleftarrow[M]{0} j$), one-direction consecutive sendings with a confirmation ($i \xleftarrow[0]{\overbrace{M \ldots M}^{s}} j$) are used to estimate the gap parameter: $g = T_s/s$, where $s = 2^x$. The number of messages is chosen to be sufficiently large in order to ensure that the point-to-point communication time is dominated by the factor of bandwidth rather than latency. This experiment, also known as saturation, reflects the nature of the gap parameter but takes a long time.

The estimation of the functional PLogP parameters, especially $g(M)$, will be time consuming because these experiments are performed for multiple message sizes, which are selected adaptively. For example, if the $g(M_k)$ is not consistent with the linearly extrapolated value based on $g(M_{k-2})$ and $g(M_{k-1})$ then another measurement will have to be performed for the message size $M_k^* = (M_k + M_{k-1})/2$.

The estimation of the PLogP parameters includes the experiments which are similar to those for LogP and have the advantage of the ability to be performed for different message sizes. The disadvantage is that the total number of parameters may become too large, although this model is adaptive in nature, because of the number and location of breaks of piecewise linear functions are determined while the model is being built.

Kielmann *et al.* (2000) measure the gap for each message size with a new method that does not need to saturate the link. The method only has to do this for messages of size zero. It starts by measuring how long it takes to send *100* messages in a row. Then this number is doubled repeatedly until the time per message sent increases by less than a threshold value (e.g. 1%). We take that time as *g(0)*. All other parameters can be determined by the procedure that starts with a synchronization message by which the so-called mirror process (receiver that will send back) indicates being ready. For each size m, two message roundtrips are necessary from measure to mirror and back. They are repeated until the measured times stabilize. In the first roundtrip, measure sends an m-bytes message and in turn receives a zero-bytes message. They measure the time for just sending and for the complete roundtrip. The send time directly yields $o_s(m)$. The $g(m)$ can be determined from the roundtrip times: $RTT(0) = 2(L + g(0))$ and $RTT(m) = L + g(m) + L + g(0)$. This yields $g(m) = RTT(m) - RTT(0) + g(0)$ and $L = (RTT(0) - 2g(0))/2$. The second roundtrip yields $RTT'(m)$. Here, measure sends a zero-bytes message, waits for a $\triangle > RTT(m)$ time and then receives a m-bytes

message. With $RTT'(m) = o_s(0) + \triangle + o_r(m)$ we get $o_r(m) = RTT'(m) - \triangle - o_s(0)$. The measurement procedure assumes that network links are symmetrical, such that sending from measure (sending node) to mirror (receiving node) has the same parameters as for the reverse direction.

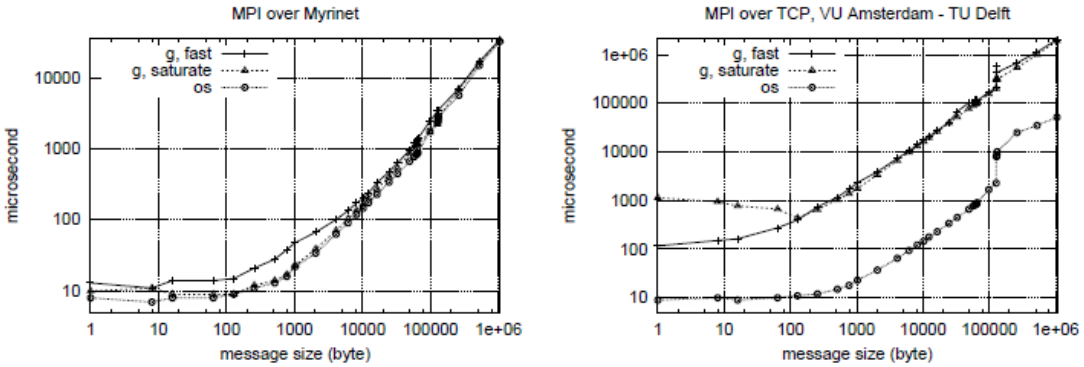


Figure 2.4: Measured send overhead and gap; over Myrinet (left) and over TCP (right) Kielmann *et al.* (2000)

The measurement procedure experimentation was done on four clusters that are made up of Pentium Pros with Myrinet connected by dedicated 6Mbit/s ATM networks, using MPI based on TCP. They measured the LogP parameters for MPI_Send and MPI_Recv as described above. They also compare the two methods of estimation for *g(m)* with the fast fast method, and also by the link saturation method. The graphs in Figure 2.4 show o_s (for comparison) and g, the curves for g are very close to each other, validating the new method.

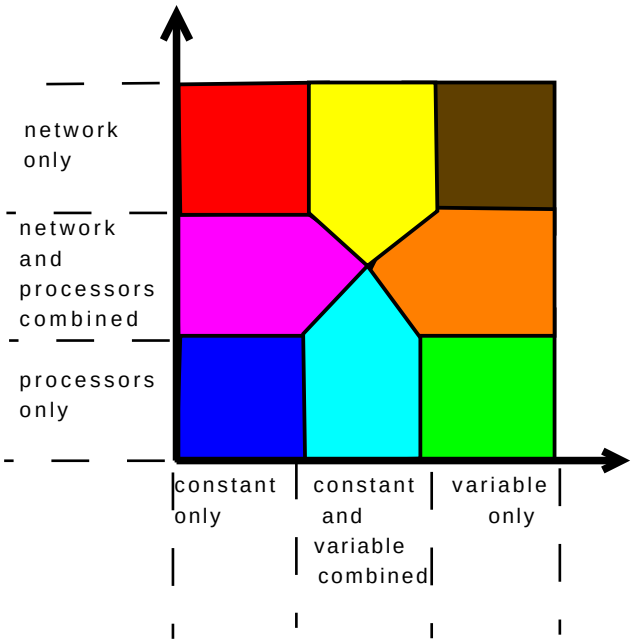
Unlike the LogP/LogGP and Hockney models, the PLogP model is not linear and therefore can more accurately approximate the execution time of point-to-point operations. However, this feature in no way can help toward more accurate analytical predictions of collectives algorithms because its functional parameters still combine the contributions of different origin. Indeed, larger numbers of such non-intuitive parameters (each piecewise linear function can be considered as a set of constant parameters) do not make analytical expression of the execution time of collective operations more intuitive. The LogP-based models can be applied to heterogeneous clusters in the same way as the Hockney model. Namely, the parameters are first found for all pairs of processors, with the above experiments being performed for each link. Then, these parameters (heterogeneous version) or their average values (homogeneous version) are used in modeling.

2.2.5 Summary

The models described above are traditionally designed for homogeneous platforms, with the same values of parameters for each pair of processors. The parameters are found *statistically* from the communication experiments *between any two processors*. The models are summed up in Table 2.1.

Table 2.1: Summary of the Performance Models from Related Works

Model	p2p	Experiments
Hockney	$\alpha + \beta M$	$\left\{ i \xleftarrow[0]{0} j + i \xleftarrow[M]{M} j \right\}_{k=0}^{R}$ or $\left\{ i \xleftarrow[M_k]{M_k} j \right\}_{k=0}^{R}$
LogP	$L + 2o$	$\left\{ i \xleftarrow[M]{M} j + i \xleftarrow[0]{M} j + i \xleftarrow[M]{0} j \right\}_{k=0}^{R} + \{ i \xleftarrow[0]{\overbrace{M \ldots M}^{2^x}} j \}_{x=0}^{S}$
LogGP	$L + 2o +$ $G(M-1)$	LogP experiments + $\{ i \xleftarrow[0]{\overbrace{\bar{M} \ldots \bar{M}}^{2^x}} j \}_{x=0}^{S}$, large $\bar{M}$
PLogP	$L + g(M)$	$\left[\left\{ i \xleftarrow[0]{M_m} j + i \xleftarrow[M_m]{0} j \right\}_{k=0}^{R} + \{ i \xleftarrow[0]{\overbrace{M_m \ldots M_m}^{2^x}} j \}_{x=0}^{S} \right]_{m=0}^{N}$



Contributions	Colour	Hockney	LogP	LogGP	PLogP
Network constant contribution only			L	L	
Network and Processors constant contribution combined		α			L
Processors constant contribution only			o	o	
Processors constant and variable contribution combined					
Processors variable contribution only					o
Processors and Network variable contributions combined		β	g	g	
Network variable contribution only				G	g
Network constant and variable contribution combined					

The colors explain the parameters contribution between network and processors, and if its constant or variable. For example Hockney's α represents the constant combination of network and processor involvement, while o is the constant processor only contribution and differs from o_s, o_r from the PLogP model that represents the variable processor contribution.

The elaboration of communication performance models in order to separate the constant and variable contributions of processors and network can lead to more accurate prediction of the communication execution time. In Lastovetsky *et al.* (2006b), Lastovetsky & Rychkov (2007), an analytical heterogeneous communication performance model which separates the variable contributions of processors and network is proposed. The model, called the *LMO*, is designed for both homogeneous and heterogeneous clusters based on a switched network. While this *LMO* model provides more intuitive and accurate expression of the execution time of MPI collective operations, its parameters cannot be estimated from the traditional point-to-point experiments only. A solution of this problem, proposed in Lastovetsky & Rychkov (2007), Lastovetsky *et al.* (2009), is to introduce additional collective communication experiments involving more than two processors. These experiments are designed to give us sufficient data in order to build and solve simple systems of equations to find the point-to-point parameters. The software tool presented Chapter 6 automates the estimation of the straightforward heterogeneous extensions of the traditional models, such as Hockney, LogP, LogGP, and PLogP. Thus the presented software tool can estimate both traditional and advanced models, and can be used both for heterogeneous clusters

and for homogeneous ones as a particular case.

2.3 Optimization of MPI Collective Communication Operations

Analytical communication performance models play an important role in optimization of parallel applications on computational clusters. Optimization of collective communications may be achieved by a number of different approaches to models and algorithms. Different models may be used together or switched. There is also the use of empirical readings directly:

- Model-based switch between algorithms;
- Multi-model based vs. empirical based switch;
- Mapping of processors and model-based mapping.

Traditional communication models, such as the Hockney model by Hockney (1994), LogP by Culler *et al.* (1993), LogGP by Alexandrov *et al.* (1995), and PLogP by Kielmann *et al.* (2000), are often used for estimation of the execution time of different algorithms of MPI collective communication operations on homogeneous clusters. For example, Chan *et al.* (2004) and Thakur *et al.* (2005) applied the Hockney model to compare the communication cost of different algorithms of the same collective operation in order to choose the fastest one for different message sizes and numbers of processors. Chan *et al.* (2004) found that using different algorithms for different data sizes (vector lengths) for optimization is better than finding one general algorithm for all cases, consistently outperforming nearly all the MPICH implementations. Their preposting method that splits the send and receives in an algorithm was illustrated using *MPI_Allreduce*. It did not give significant performance gains for small message sizes, but with a zero-length message method of preposting, using Reduce-Scatter they achieved gains of a factor of three difference for long messages.

In the case of heterogeneous clusters, there is another application of communication performance models to the optimization of MPI collective operations. Namely the performance of a collective operation can be improved by the optimal mapping of heterogeneous processors to the nodes of the communication tree of the operation. Traditional communication performance models are usually homogeneous, with parameters having the same values for all processors and links. Therefore they give the same prediction for any mapping. Heterogeneous communication models do distinguish the contributions of different links and processors and hence may be used for this purpose. Bhat *et al.* (2003) and Hatta

& Shibusawa (2000) build optimal communication trees for collective operations with help of heterogeneous extension of the Hockney model.

Vadhiyar *et al.* (2000) developed automatically tuned collective communication algorithms. They measured the performance of different algorithms of collective communications for different message sizes and numbers of processes and then used the best algorithm. They also discuss a dynamic topology method that uses the tuned static topology shape, but re-orders the logical addresses to compensate for changing run-time variations. A series of experiments were conducted comparing the tuned collective communication operations to various native vendor MPI implementations. The use of the tuned collective communications resulted in about 30%-650% improvement in performance over the native MPI implementations.

All works on the optimization of collective operations are based on deterministic linear communication models. Implementation of the optimized versions of collective operations in *HeteroMPI*, Lastovetsky & Reddy (2006), uses the *LMO* performance model (of this thesis) that takes into account non-deterministic escalations of the execution time of Many-to-One MPI communications for medium-sized messages, and the leap in the execution time of One-to-Many communications for large messages.

2.3.1 MPICH-2 Collectives

Thakur *et al.* (2005) used a simple linear cost model of a point-to-point single communication in selection of algorithms for a particular collective communication operation. They use the Hockney model to estimate the communication performance of different algorithms of collective operations. For a particular collective operation they suggested switching between algorithms to choose the fastest one for each given message size and number of processors.

They identify the best algorithms and improve on them or develop new algorithms where necessary, and implement them efficiently. For each collective operation, they use multiple algorithms based on message size: The short-message algorithms aim to minimize latency, and the long-message algorithms aim to minimize bandwidth use. They use experimentally determined cutoff points to switch between different algorithms depending on the message size and number of processes. They have implemented new algorithms in MPICH for all the MPI collective operations, namely, scatter, gather, allgather, broadcast, all-to-all, reduce, allreduce, reduce-scatter, scan, barrier, and their variants.

2.3.2 The OCC Library

The Hockney, LogP, LogGP, and PLogP models are directly compared in the paper Pješivac-Grbović *et al.* (2005) to analyze parallel algorithm performance that are used for optimization of collective operations. The models are found to be generally pessimistic and reasonably representative of the operations. They apply different approaches to the switch-optimization: analytical methods based on communication performance models, Pješivac-Grbović *et al.* (2005), and empirical methods based on graphical, Pješivac-Grbović *et al.* (2007a), or statistical analysis of observations, Pješivac-Grbović *et al.* (2007b). They demonstrate that generally, empirical methods better reflect the behavior of different algorithms, and the switch between the algorithms based on empirical methods results in smaller performance penalty. The practical use of empirical methods is limited by non-changeable computational clusters, because these methods require conducting exhaustive communication experiments to collect performance data for different collective algorithms, number of processors and message sizes.

The Hockney model was the simplest and more adaptable for message size than LogP, but was found to be the least accurate. Since none of the models could represent congestion, the authors omit these readings. This is a critical shortcoming mentioned in the cases of all the models investigated.

They compare their predictions from the models to the experimentally gathered data and the findings were used to optimize the implementation of collective operations in the FT-MPI library. This is a new library allowing the best model selection of a particular algorithm implementation for the optimization of the communication. This saves a lot of time by the user in testing out various optimizations of the MPI collective operation. This information is combined with predictions from parallel communication models to make run-time decisions to select near-optimal algorithms and segment sizes for a given operation, communicator, message size, and the rank of the root process.

They propose a decision tree solution to the optimization process, the experimental and analytical analysis of collective algorithm performance is used to determine switching points between available methods. They have a framework for performance testing known as the Optimized Collective Communication (OCC) library, it is an MPI collective library built on top of MPI's point-to-point operations. It provides a simple interface for addition of new collective algorithms and provides basic verification tools for the existing methods. The performance module provides measurement tools for the library. The OCC library currently supports five different virtual topologies: *flat-tree/linear, pipeline (single chain), binomial tree, binary tree*, and *k-chain tree*. For a given a collective operation, message size, and number of processes, the OCC shows which topologies can be beneficial for some combination of parameters.

The FT-MPI uses the OCC library for experimental and analytical analysis of collective algorithm performance was used to determine switching points between available methods. At run time, a particular method is thus selected based on the number of processes in the communicator, message size, and the rank of the root process. They did performance tests for different Barrier, Broadcast, Reduce, Scatter, and Alltoall collective operations implementations using their library FT-MPI compared to MPICH and MPICH-2. They analyzed the algorithm performance and the optimal implementation of different collective operations. They discuss the limitations of current performance models in mapping to collective operations for MPI, finding Hockney, LogP, PLogP and LogGP are too pessimistic in general, with Hockney being too optimistic for large messages. They also have the disadvantage of requiring too many experiments for all mappings and processes with too much empirical data produced. Predictions from the PLogP and LogGP model were sufficiently close that one could use either of the models to reach similar conclusions. They found the PLogP model has more flexible parameters and that the predictions were the closest to the experimental results of collective communications.

2.3.3 MagPIe Software

Kielmann *et al.* (1999) developed the MagPIe library that implements new algorithms for collective communications for Wide Area Networks optimised for wide area systems. MagPIe's algorithms send the minimal amount of data over the slow wide area links and only incurs a single wide area latency. Existing MPI applications may be run unmodified on geographically distributed systems, MagPIe is found to run up to 10 times faster than MPICH. This library is based on MPICH, but the collective communication primitives use new algorithms optimized for wide area systems. The basic assumption is that the wide area system is hierarchically structured and a computational grid to consist of many parallel computers connected by wide area networks. Therefore algorithms need to be adapted to a hierarchical system.

Collective communication algorithms are usually designed for local area networks, which have a low latency. Wide area systems however, have a high latency (and a lower bandwidth) and the performance of collective communication operations is dominated by the traffic over the wide area links. Thus their algorithms are designed to reduce traffic over the slow links, resulting in a markedly different communication structure. Nodes within a local cluster are connected by slower wide area links, by reducing traffic over these the speedup is achieved. They find a flat tree algorithm implementation the best for interconnection between clusters, rather than binomial tree, as it reduces the amount of data required for interconnecting across the wide area.

2.3.4 Summary

The optimization of collective communications operations is achieved in different ways. There are methods to swap between models for different implementations, to rely on direct empirical readings or to map to processors for advantage. The performance of different algorithms for collective operations is estimated in some methods, switching between algorithms to choose the fastest one. The models are used optimize communications operations using different algorithms for different data sizes. There is also the idea of automatic tuning developed from testing performances of different algorithms with different message sizes. A multi-model approach with Hockney and LogP models is compared with a decision tree based on empirical observations that switch between different algorithms, topologies, and message segment sizes. Finally there the approach with the mapping of processors and model-based mapping for an algorithm to optimize its communication performance.

Our own work builds on these ideas, and presents a software tool that supports both traditional and advanced communication performance models. While its primary target platforms are heterogeneous clusters, the software tool can also be used for homogeneous clusters as a particular case.

Chapter 3

MPIBlib: MPI Benchmarking Library

We designed a new MPI benchmarking suite called MPIBlib (MPI Benchmarking Library, Lastovetsky *et al.* (2008a)) for the accurate estimation of the execution time of MPI communication operations. The accuracy of a communications performance model depends on the accuracy with which its parameters are measured. In Chapter 2 we discussed the current benchmarking methods available and took note of their limitations. Our library offers the choice of a variety of timing methods. This suite supports both the fast measurement of collective operations and point-to-point benchmarking. It plays an important role in particular when building the new LMO performance model in Chapter 4. The estimated data is also useful for the tuning of collective communication operations and allows for the evaluation of different available implementations.

Most of the MPI benchmarking suites are designed in the form of a standalone executable program that takes the parameters of communication experiments and produces a lot of output data for further analysis. As such it cannot be integrated easily and efficiently into application-level software. Therefore there is a need for a benchmarking library that can be used in parallel applications or programming systems for communication performance modeling, and for the tuning communication operations. MPIBlib is such a library that can be linked to applications and used at runtime. The software provides some of the wide range of choice of configuration that SkaMPI allows (see chapter 2) but also provides the user with a choice of balance between speed of use and fine tuning of accuracy, with the additional convenience of an integrative library format.

The package consists of a library and benchmark executables (Figure 3.1). The library implements the main functionality. The executables perform point-to-point and collective benchmarks and produce the output, including the results of measurements. The results of measurements can be visualized by the gnuplot

utility, Williams & Kelley (2007), and MPIBlib provides the basic gnuplot scripts.

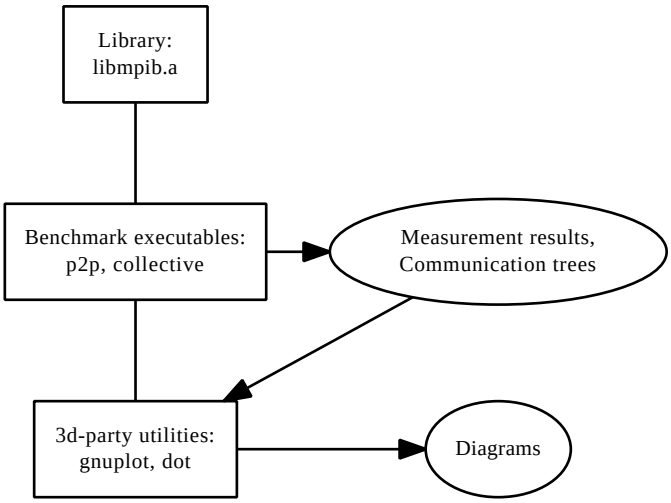


Figure 3.1: MPIBlib design

This chapter describes the main features of MPIBlib that are used in communication performance modeling. We describe in detail the software design of the MPIBlib suite, and the measurement module in particular. The point-to-point and collective modules are described next with details of their software design and functionality. Examples of the usage of the library is outlined and finally there are experiments to demonstrate the benchmarking with graphical results.

3.1 MPIBlib Software Design

The MPIBlib software is made up of a number of modules. The main library modules are shown in Figure 3.2. The modules marked grey can be extended, providing the benchmarking of user-defined point-to-point and collective operations.

A summary of modules of the software includes:

- **Measurement** - Measurement module provides basic data structures and functions for measurement.
- **Point-to-point benchmark** - Measures the point-to-point time with a choice of sequential or parallel methods.

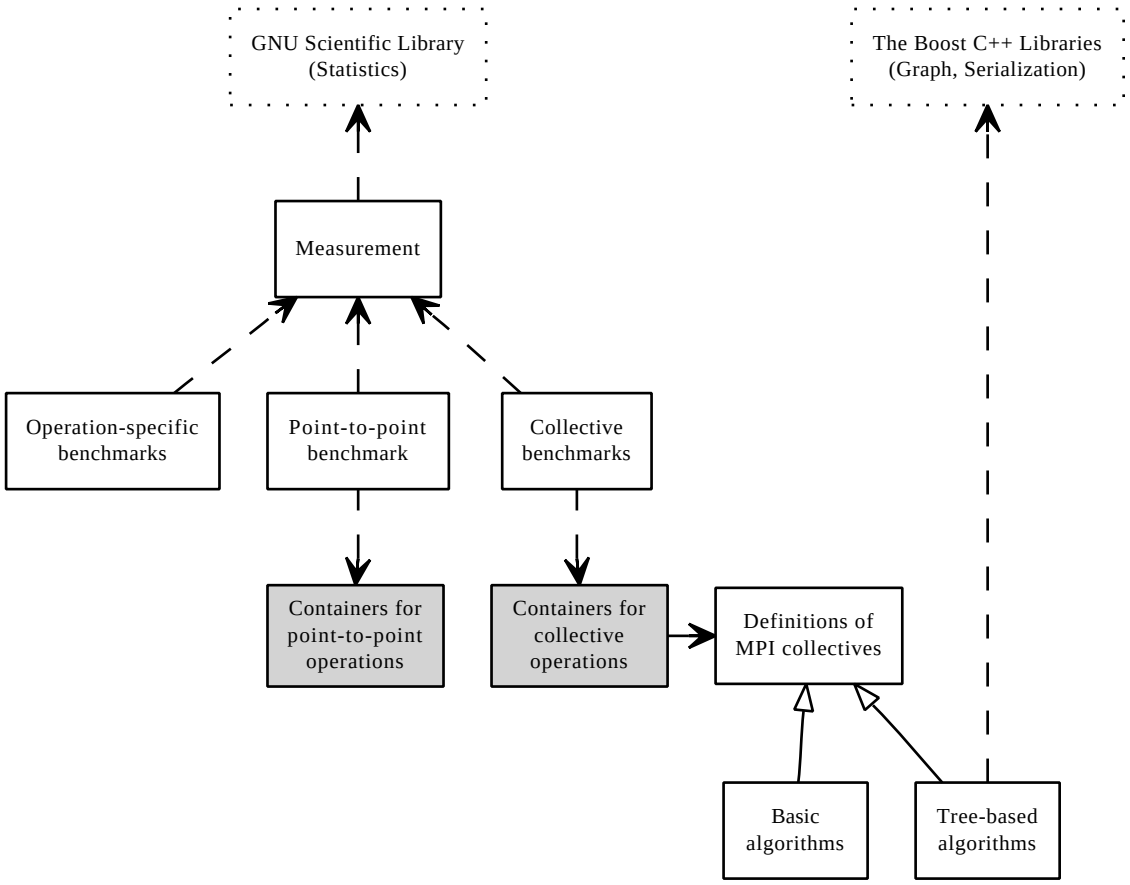


Figure 3.2: MPIBlib modules

- **Containers for point-to-point (p2p) communication operations** - Provides the containers for p2p communication operations to be measured by p2p benchmark.
- **Collective benchmarks** - Measures the collective operation time with a choice of *root, max* or *global* methods.
- **Containers for collective communication operations** - Provides the containers for collective communication operations to be measured by collective benchmark.
- **MPI collective communication operations** - This module contains the definitions of types of MPI collective operations.
- **Basic algorithms** - This module provides basic, mostly linear, algorithms

of MPI collective operations.

- **Tree-based algorithms**- This module provides tree-based algorithms of MPI collective operations. Depends on the Boosh C++ libraries.

We begin with the measurement module as the core of the MPIBlib software. There are two main data structures, *MPIB_precision* and *MPIB_result*. The benchmark functions take *MPIB_precision* as an input argument and return *MPIB_result*, as in Figure 3.3. The *MPIB_precision* contains the parameters of measurement as input parameters of benchmarking functions. The *MPIB_result* data structure contains the results of measurement and also its accuracy, its statistical reliability.

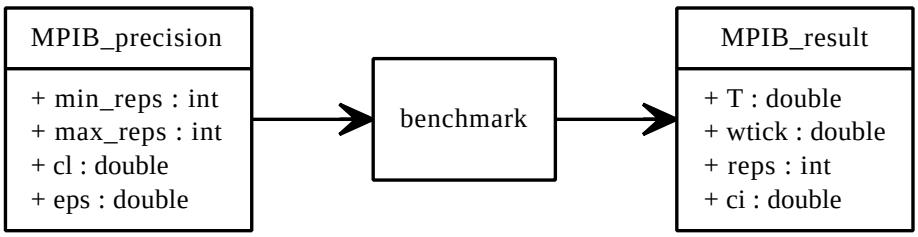


Figure 3.3: Statistical parameters

The *MPIB_precision* contains the following fields:

- *min_reps*; minimum number of repetitions.
- *max_reps*; maximum number of repetitions.
- *cl*; confidence level $\in [0, 1]$ $cl = Pr(|\bar{T} - \mu| < ci) = Pr(\frac{|\bar{T}-\mu|}{\bar{T}} < \epsilon)$ where μ is the mean, $\bar{T}$ is the average time obtained from several observations, Pr is the probability, ci is confidence interval.
- *eps*; relative error $\in [0, 1]$ $\frac{|\bar{T}-\mu|}{\bar{T}} < \frac{ci}{\bar{T}} < \epsilon = eps$.

The precision argument, *precision* in Figure 3.3, specifies how many times to repeat the communication experiment in order to obtain an accurate result. It is used in the following way:

- If parameters *min_reps* and *max_reps* (which represent minimum and maximum number of repetitions) have the same value, the communication experiment will be repeated a fixed number of times, equal to this value. This controls efficiency of benchmarking but does not provide a certain level of accuracy.

- If *min_reps* < *max_reps*, the communication experiment will be repeated until the sample of the measured execution times satisfies the Student's *t*-test, estimations are using the GNU Scientific Library, Galassi *et al.* (2009), with the confidence level, *cl*, and relative error, *eps*, or the number of repetitions reaches its maximum, *max_reps*. In this case, the number of repetitions for different pairs of processors and for different message sizes will vary, but a certain accuracy of benchmarking will be guaranteed.

An auxiliary function *MPIB_ci* returns the confidence interval that contains the average execution time with a certain probability $Pr(|\bar{T} - \mu| < ci) = cl$. For statistical analysis, the GNU Scientific Library, Galassi *et al.* (2009) is used.

```
double MPIB_ci(double cl, int reps, double* T);
```

The *MPIB_result* data structure consists of:

- T; execution time (The main results of measurement).
- *wtick*; resolution of MPI_Wtime (accuracy of the timer).
- *reps*; number of repetitions the benchmark has actually taken.
- *ci*; confidence interval, $|\bar{T} - \mu| < ci$ (statistical reliability).

These two data structures will be used in the benchmarking functions described in the coming sections.

3.2 Point-to-point Benchmarks

The point-to-point benchmark measures the execution time of the point-to-point communications between all pairs of processes in the MPI communicator. The use of the results of the point-to-point benchmarks can be versatile in application. They can be used for the estimation of parameters of the analytical communication performance models, such as Hockney, LMO.

The point-to-point benchmarking is performed by the function *MPIB_measure_p2p* between all pairs of processors.

```
void MPIB_measure_p2p(MPIB_p2p_container *container, MPI_Comm comm,
int M, int parallel, MPIB_precision precision, MPIB_result *results)
```

It performs a point-to-point communication operation multiple times, as it is defined by the precision argument.

The function *MPIB_measure_p2p* returns the results array, which contains C_n^2 values corresponding to each pair: estimations of execution time, timer resolutions, numbers of repetitions and confidence intervals. The point-to-point benchmarks can be run either sequentially or in parallel on the communicator consisting of more than two processors. If performed in parallel, each process is involved in no more than one communication. This allows us to significantly reduce the overall execution time of the point-to-point benchmark code and gives us quite accurate results on the clusters based on switched networks.

3.2.1 Container Paradigm

The benchmarking functions are implemented with the programming paradigm of a *container*, that can contain interchangeable communication operations to be measured. The program creates an instance of the container and then passes the instance to the measurement function, when the container has been 'filled' with an inherited type of special container for a particular operation. The container structures are defined as a cascading set of inherited types. By this mechanism new and more specialized structures can be defined in terms of existing structures. When a child structure (subclass) inherits from a parent structure (superclass), the subclass then includes the definitions of all the attributes and methods that the superclass defines. We can extend the superclass by adding its own attributes and methods, in our case to adapt to the selected communication operation. The base *p2p* container has pointers to functions that prepare buffers, performs the operation, and then free buffers for the point-to-point operation. These are general functions that are then allocated to the particular operation in an extensible way.

```
typedef struct MPIB_p2p_container {
 void (*initialize)(void* this, MPI_Comm comm, int M);
 void (*execute_measure)(void* this, MPI_Comm comm, int M, int mirror);
 void (*execute_mirror)(void* this, MPI_Comm comm, int M, int measure);
 void (*finalize)(void* this, MPI_Comm comm); }
MPIB_p2p_container;
```

The *MPIB_p2p_container* data structure consists of the four functions, *initializing, sending (execute_measure)* and *receiving (execute_mirror)*, and *finalizing*, see Figure 3.4. The MPIBlib library offers the possibility of extension by creating a data structure with the first field of *MPIB_p2p_container*. The container here is a basic data structure that encapsulates point-to-point communication operations. The container is now extended to become the Send/Recv container by inheritance, by creating a data structure with the first field *MPIB_p2p_container*.

```
typedef struct MPIB_Send_Recv_container{
  MPIB_p2p_container base;
  char* buffer;
} MPIB_Send_Recv_container;
```

Next we call the allocation function that constructs this container, this assigns the function pointers for the *Send/Recv* operations in this case, as follows:

```
MPIB_p2p_container* MPIB_Send_Recv_container_alloc() {
  MPIB_p2p_container* container =
    (MPIB_p2p_container*)malloc(sizeof(MPIB_Send_Recv_container));
  container->base.operation = "MPI_Send-MPI_Recv";
  container->base.free = MPIB_p2p_container_free;
  container->initialize = MPIB_Send_Recv_initialize;
  container->execute_measure = MPIB_Send_Recv_execute_measure;
  container->execute_mirror = MPIB_Send_Recv_execute_mirror;
  container->finalize = MPIB_Send_Recv_finalize;
  return container;
}
```

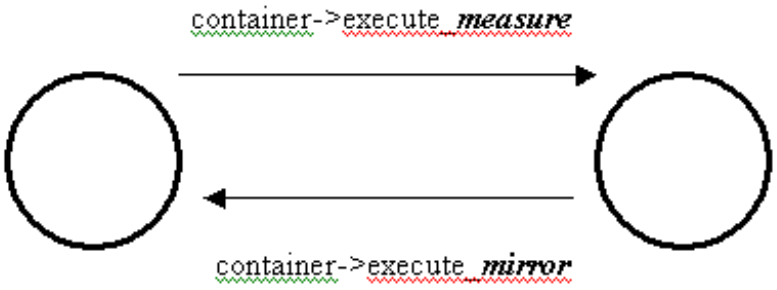


Figure 3.4: Container paradigm for point-to-point benchmarking

The four base functions of the *MPIB_Send_Recv_container* are inherited from the *MPIB_p2p_container*, and are assigned to be the operation *Send/Recv's*. First there is *initialization* of buffers required for the communication operation for a message size. The point-to-point measuring operations now take place. The operation has a *measure* function for the sending node, and a *mirror* for the receiving of the point-to-point communication. The sending communication at the *measure* side has M the message size, with the parameter *mirror* as the

number identifying the mirror processor. The *mirror* function of the point-to-point operation is the receiving processor's function, as in Figure 3.4. Then there is a *finalization* that frees buffers required for the communication operation.

```
void MPIB_Send_Recv_initialize(void* _this, MPI_Comm comm, int M)
{
  MPIB_Send_Recv_container* container
                    = (MPIB_Send_Recv_container*)_this;
  container->buffer = (char*)malloc(sizeof(char) * M);
}

void MPIB_Send_Recv_execute_measure(void* _this, MPI_Comm comm, int
M, int mirror) {
  MPIB_Send_Recv_container* container(MPIB_Send_Recv_container*)_this;
  MPI_Send(container->buffer, M, MPI_CHAR, mirror, 0, comm);
  MPI_Recv(container->buffer, M, MPI_CHAR, mirror, 0, comm,
  MPI_STATUS_IGNORE);
}

void MPIB_Send_Recv_execute_mirror(void* _this, MPI_Comm comm, int
M, int measure) {
  MPIB_Send_Recv_container* container
                            = (MPIB_Send_Recv_container*)_this;
  MPI_Recv(container->buffer, M, MPI_CHAR, measure, 0, comm,
  MPI_STATUS_IGNORE);
  MPI_Send(container->buffer, M, MPI_CHAR, measure, 0, comm);
}

void MPIB_Send_Recv_finalize(void* _this, MPI_Comm comm) {
  MPIB_Send_Recv_container* container =
    (MPIB_Send_Recv_container*)_this; free(container->buffer);
}
```

This approach has the advantage that the code is extendible to have different types of operations to be measured by the same benchmarking function using inheritance.

3.2.2 Using the Point-to-point Benchmark

The MPIBlib is a library that can be included with the parallel application. Consider how this function is used, and in particular its precision argument, in the

estimation of the heterogeneous Hockney model. The first method of estimation (see the code below) measures empty and non-empty roundtrips (line 19) and therefore requires a very high accuracy of measurements. The benchmarking function will be called twice (lines 20, 21), first time with the message size argument equal to zero and second time with a message size $M > 0$. To guarantee the high precision of the measurements, we set the maximum number of repetitions to be sufficiently large, with the confidence level 95% and the relative error 2.5% in the precision data structure. This is passed to the *MPIB_measure_p2p* function as an argument.

```
1 void Hockney_build(MPI_Comm comm, MPIB_precision precision,
      MPIB_msgset msgset, int parallel, Hockney_model** model)
2 {
3     int rank;
4     MPI_Comm_rank(comm, &rank);
5     int size;
6     MPI_Comm_size(comm, &size);
7     if (size < 2) {
8         if (rank == 0)
9             fprintf(stderr, "Cannot compute Hockney parameters for %
                  d processes (must be >= 2)\n", size);
10        return;
11    }
12    *model = rank == 0 ? Hockney_alloc(size) : NULL;
13    MPIB_result* res_0 = (MPIB_result*)malloc(sizeof(MPIB_result) *
          MPIB_C2(size));
14    MPIB_result* res_M = (MPIB_result*)malloc(sizeof(MPIB_result) *
          MPIB_C2(size));
15    MPIB_p2p_container* container = MPIB_Send_Recv_container_alloc()
          ;
16    MPIB_measure_p2p(container, comm, 0, parallel, precision, res_0)
          ;
17    MPIB_measure_p2p(container, comm, msgset.max_size, parallel,
          precision, res_M);
18    MPIB_container_free(container);
19    if (rank == 0) {
20        int i, n;
21        for (i = 0, n = size * (size - 1) / 2; i < n; i++) {
22            (*model)->a[i] = res_0[i].T / 2;
23            (*model)->b[i] = (res_M[i].T - res_0[i].T) / (2 * msgset
                  .max_size);
24        }
25    }
26    free(res_0);
27    free(res_M);
28 }
```

The second method of the estimation of the Hockney model uses multiple roundtrips with different message sizes and therefore requires a high-speed measurement of each individual roundtrip. The accuracy of this method depends on the number of message sizes, for which roundtrips are measured, returning a satisfactory estimate even with low precision of individual measurements. In this latter case, the *MPIB_measure_p2p* function will be called multiple times with different message sizes in the low precision mode: *min_reps = max_reps = 1*.

3.3 Collective Benchmarks

The collective benchmarks measure the execution time of any MPI collective communication operation, using one or another universal timing method. The user has a choice of timing method and collective communication operation (with its particular algorithms and implementations) to be measured. The benchmark functions are defined to be operation-independent by the use of function pointer referencing to an MPI collective operation, in a similar way to point-to-point using the container paradigm. The software is structured so that the set of communication operations and their implementations that can be benchmarked by MPIBlib is open for extensions.

The collective benchmark software offers the user a choice of *root, maximum* or *global* timing, as described in the previous chapter in detail. Root timing selection means the timing is taken from the chosen root processor, a barrier ensures synchronization and short messages are sent to confirm completion of an operation. The maximum choice means taking the maximum of measured execution times for a communication operation and also synchronizes with barriers. The global timing is the most accurate estimate as it takes the differences in local clocks of different processors into account when measuring communication execution time. However, it is more time consuming and less efficient than those methods based on local clocks. The user may find the faster less accurate choices to be accurate enough for their purposes, or they may wish to select the more finely tuned global option.

3.3.1 API for Timing Methods

The measurement choice is made by the user, this is done by selecting the measurement function at run-time - that is *root, max* or *global*, generally a collective benchmark function is defined as follows:

```
typedef void(*MPIB_measure_coll)(MPIB_coll_container *container,
MPI_Comm comm, int root, int M, MPIB_precision precision,
MPIB_result *result)
```

The user assigns a function pointer, *MPIB_measure_coll* to the chosen measurement function. The three measurement choices are described next.

```
void MPIB_measure_max(MPIB_coll_container *container, MPI_Comm comm,
int root, int M, MPIB_precision precision, MPIB_result
*result)
```

The function *MPIB_measure_max* is passed the allocated container, in this way it is separate to the chosen operation and allows for interchangeability. It measures the execution time of collective operation at all processes and finds a maximum. In the loop, this function:

- Synchronizes the processes by double barrier.
- Measures the execution time of collective operation at all processes.
- Finds the maximum execution time by collective operation 'Allreduce' with the values measured at all processors.
- Performs statistical analysis, Student's t-test, at all processors.

Finally, if the sample (the measured execution times) satisfies the Student's t-test, the function returns the result.

```
void MPIB_measure_root(MPIB_coll_container *container, MPI_Comm
comm, int root, int M, MPIB_precision precision, MPIB_result
*result)
```

This is the *root* measurement function to measure the execution time of collective operation at the root process. In the loop, this function:

- Synchronizes the processes by double barrier.
- Measures the execution time of collective operation with barrier confirmation at the root process.
- Subtracts the average execution time of barrier.
- Performs statistical analysis, Student's t-test, at root.

Finally, if the sample (the measured execution times) satisfies the Student's t-test, the function broadcasts and returns the result. This benchmarking function requires the average execution time of barrier to be measured first. The barrier time can be estimated only once for a communicator. At all processes we introduced a global variable to store the average barrier time for the root benchmarking function.

```
void MPIB_measure_global(MPIB_coll_container *container, MPI_Comm
comm, int root, int M, MPIB_precision precision, MPIB_result
*result)
```

The *global* timing function measures the execution time of collective operation between processes using global time. It reuses already obtained offsets between local clocks if the previous initialization was performed on the same MPI communicator. In the loop over repetitions:

- Synchronizes the processes by double barrier.
- Measures the moment of start at the root and the moment of finish at the rest of processes.
- Having subtracted the offset, finds maximum by reducing to the root.
- Performs statistical analysis at root.

This benchmarking function requires the offset of each processor's execution time from the root processor time to be measured first. At all processors, we introduced a global variable to store the offsets between the local clocks and the clocks of other processors in the communicator. These values are used for the global timing.

3.3.2 API for MPI Collective Communication Operations

The MPIBlib library code offers the possibility of extension by creating a base data structure with the first field '*MPIB_coll_container*'. The functions that allocate and free the data structure for example *MPIB_Scatter_container_alloc* has an argument *MPIB_Scatter*, pointer to a scatter implementation. This provides three-level extension of measurement, as in Figure 3.5. The advantages of this approach are that the buffering allocation and operation selection is separate from the measurement functionality, making the code easy to interchange and extend.

The basic container for a collective operation is the *MPIB_coll_container* has pointers to functions that *initialise, execute* and *finish* the collective operation:

```
typedef struct MPIB_coll_container {
        void (*initialize)(void* this, MPI_Comm comm, int root, int M);
        void (*execute)(void* this, MPI_Comm comm, int root, int M);
        void (*finalize)(void* this, MPI_Comm comm, int root);
} MPIB_coll_container;
```

The following sequence demonstrates how to encapsulate the scatter operation in order to benchmark it using different timing methods.

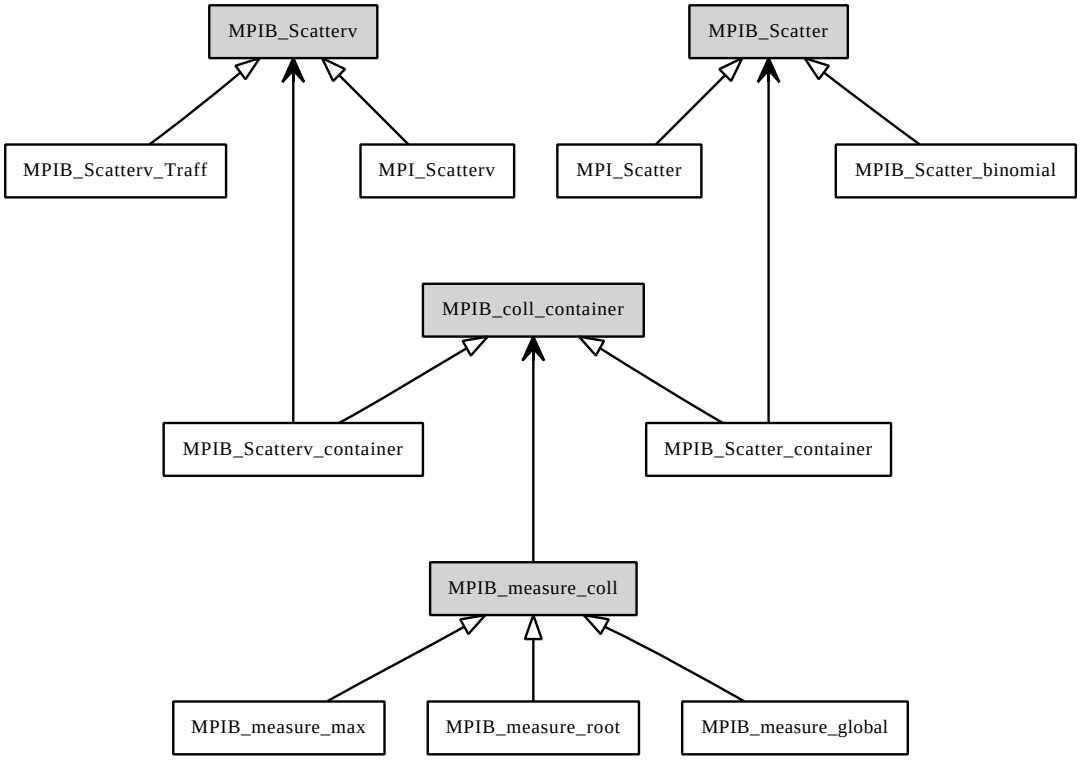


Figure 3.5: Extension of collective benchmarks: timing methods, communication operations, algorithms

- Create a data structure with the first base field *MPIB_coll_container*, thus inheriting the base container's functions *initialize, execute, finalize*:

```
typedef struct MPIB_Scatter_container {
  MPIB_coll_container base;
  char * buffer;
  MPIB_Scatter scatter;
} MPIB_Scatter_container;
```

- The allocation function for scatter returns the scatter container with its scatter operation and *initialize, execute, finalize* functions allocated to that operation, as below, and its function pointers now point to the operation scatter functions:

```
MPIB_Scatter_container* MPIB_Scatter_container_alloc(MPIB_Scatter
```

```
scatter) {
  MPIB_Scatter_container* container
    =(MPIB_Scatter_container*)malloc(sizeof(MPIB_Scatter_container));
  container->base.base.operation = "Scatter";
  container->base.base.free = MPIB_Scatter_container_free;
  container->base.initialize = MPIB_Scatter_initialize;
  container->base.finalize = MPIB_Scatter_finalize;
  container->base.execute = MPIB_Scatter_execute;
  container->scatter = scatter;
  return container;
}
```

- Next we implement the functions to *initialize, execute, finalize* routines, where *_this* argument is a function pointer that can be typecasted to the above data structure. We use these functions of the container in the measurement function for the particular operation eg. scatter.

```
void MPIB_Scatter_initialize(void* _this, MPI_Comm comm, int root,
int M) {
  MPIB_Scatter_container* container = (MPIB_Scatter_container*)_this;
  int rank;
  MPI_Comm_rank(comm, &rank);
  int size;
  MPI_Comm_size(comm,&size);
  container->buffer = rank == root ?
 (char*)malloc(sizeof(char)*M* size):(char*)malloc(sizeof(char)*M);
 }

void MPIB_Scatter_execute(void* _this, MPI_Comm comm, int root, int
M) {
  MPIB_Scatter_container* container=(MPIB_Scatter_container*)_this;
  container->scatter(container->base.buffer, M, MPI_CHAR,
    container->base.buffer, M, MPI_CHAR, root, comm);
}

void MPIB_Scatter_finalize(void* _this, MPI_Comm comm, int root)
{
  MPIB_Scatter_container* container = (MPIB_Scatter_container*)_this;
  free(container->buffer);
}
```

The advantage of the container approach is that it allows the choice of timing methods to be used with different containers. Implementing the scatter container then means one can also measure the execution time of different algorithms of scatter. In this way the code is extendible and reusable for both the timing method and its algorithm implementation. The following code shows how the container data structure is used in the collective benchmarks. We consider only maximum timing method in detail. Other timing methods are implemented in similar way.

```
1  void MPIB_measure_max(MPIB_coll_container* container, MPI_Comm comm,
2  int root, int M,    MPIB_precision precision, MPIB_result* result) {
3      double* T = (double*)malloc(sizeof(double) * precision.max_reps)
           ;
4
5      container->initialize(container, comm, root, M);
6      int stop = 0;
7      double sum = 0;
8      int reps = 0;
9      double ci = 0;
10     while (!stop && reps < precision.max_reps)
11     {
12         MPI_Barrier(comm); MPI_Barrier(comm);
13         double time = MPI_Wtime();
14         container->execute(container, comm, root, M);
15         time = MPI_Wtime() - time;
16         MPI_Allreduce(&time, &T[reps], 1, MPI_DOUBLE, MPI_MAX, comm)
               ;
17         sum += T[reps];
18         reps++;
19         if (reps >= precision.min_reps && reps > 2)
20             stop = (ci = MPIB_ci(precision.cl, reps, T)) * reps /
                   sum < precision.eps;
21     }
22     container->finalize(container, comm, root);
23
24     result->T = sum / reps;
25     MPIB_max_wtick(comm, &result->wtick);
26     result->reps = reps;
27     result->ci = ci;
28     free(T);
29 }
```

In line 6, the container allocates the buffers for collective communication operation at all processors. In line 15, the communication operation is performed. In line 23, the buffers are deallocated. These steps are common for all timing methods. The communication execution time is measured at all processors (lines 14, 16), with maximum value found with help of allreduce (line 17).

3.3.3 Using the Collective Benchmark library

The following is an example of the collective benchmarking library in use. When finding the threshold parameters for the LMO model (Chapter 4 has further description of the model and its estimation), the benchmarks for linear scatter and gather are performed (see the code below, lines 13, 19).

```
1 void LMO_build(MPI_Comm comm, MPIB_precision precision, MPIB_msgset
      msgset, int parallel,   LMO_model** model)
2 {
3     ...
4     int n = max_size / stride;
5     int* M = (int*)malloc(sizeof(int) * n);
6     int i, size;
7     for (i = 0, size = 0; i < n; i++, size += stride)
8         M[i] = size;
9     MPIB_result* results = (MPIB_result*)malloc(sizeof(MPIB_result)
          * n);
10
11    MPIB_coll_container* container = (MPIB_coll_container*)
          MPIB_Scatter_container_alloc(MPIB_Scatter_linear);
12    for (i = 0; i < n; i++)
13        MPIB_measure_max(container, comm, 0, M[i], precision, &
              results[i]);
14    MPIB_container_free(container);
15    Piecewise linear regression analysis on M[] and results[] to
          find the S parameter.
16
17    MPIB_coll_container* container = (MPIB_coll_container*)
          MPIB_Gather_container_alloc(MPIB_Gather_linear);
18    for (i = 0; i < n; i++)
19        MPIB_measure_max(container, comm, 0, M[i], precision, &
              results[i]);
20    MPIB_container_free(container);
21    Piecewise linear regression analysis on M[] and results[] to
          find the M1 nd M2 parameters.
22    ...
23 }
```

In lines 11, 17, the containers for linear scatter and gather are created. Both collective benchmarks are performed for multiple message sizes (lines 12-13, 18-19). The method of estimation of the LMO threshold parameters is based on the piecewise linear regression (lines 15, 21) and requires a large number of single measurements, that is measurements made without any repetitions. In this method we are looking for any escalation in the communication execution time, in order to detect the structural changes in the linear regression models as accurately as possible. Measuring with higher precision, in this case, would take longer and provide the average execution time, which may lead to a wrong regression model.

The MPIBlib suite also provides **standalone** applications for benchmarking point-to-point and collective MPI communications, in addition to the library. These are *p2p* and *collective*, and a set of gnuplot scripts for visualization of the results of measurements. These executables allow the selection of options at run-time to select the following choices below. These choices include message sizes, accuracy selection, collective operation type and for point-to-point a choice of measurements performed in sequential or parallel modes. Typical options for executables are as follows:

- **-i** input file, and **-o** output file
- **-O** collective operation (required): MPI_Scatter, MPI_Gather, MPIB_Scatter_linear, MPIB_Gather_linear MPIB_Scatter_binomial, etc.
- **-t** timing: max, root, global (default: max)
- **-s** message size stride, **-m** minimum message size, **-M** maximum message size (default: 204800)
- **-p** *0/1* parallel p2p benchmarking (default: 1)
- **-r** minimum number, and **-R** maximum number of repetitions (default: 100)
- **-c** confidence level: $0 < D < 1$ (default: 0.95)
- **-e** error: $0 < D < 1$ (default: 0.025)

An example of its use with options to control message sizes and precision variables is as follows, to benchmark the *MPI_Scatter* operation is as follows:

```
$mpirun -np 16 collective -O MPI_Scatter -M 131072 -R 200 -c 0.97
```

3.4 Experiments

We performed experiments with scatter and gather benchmarks on homogeneous and heterogeneous clusters with different MPI implementations. In this experiment we present the results for a heterogeneous 16-node cluster (specified in Appendix A).

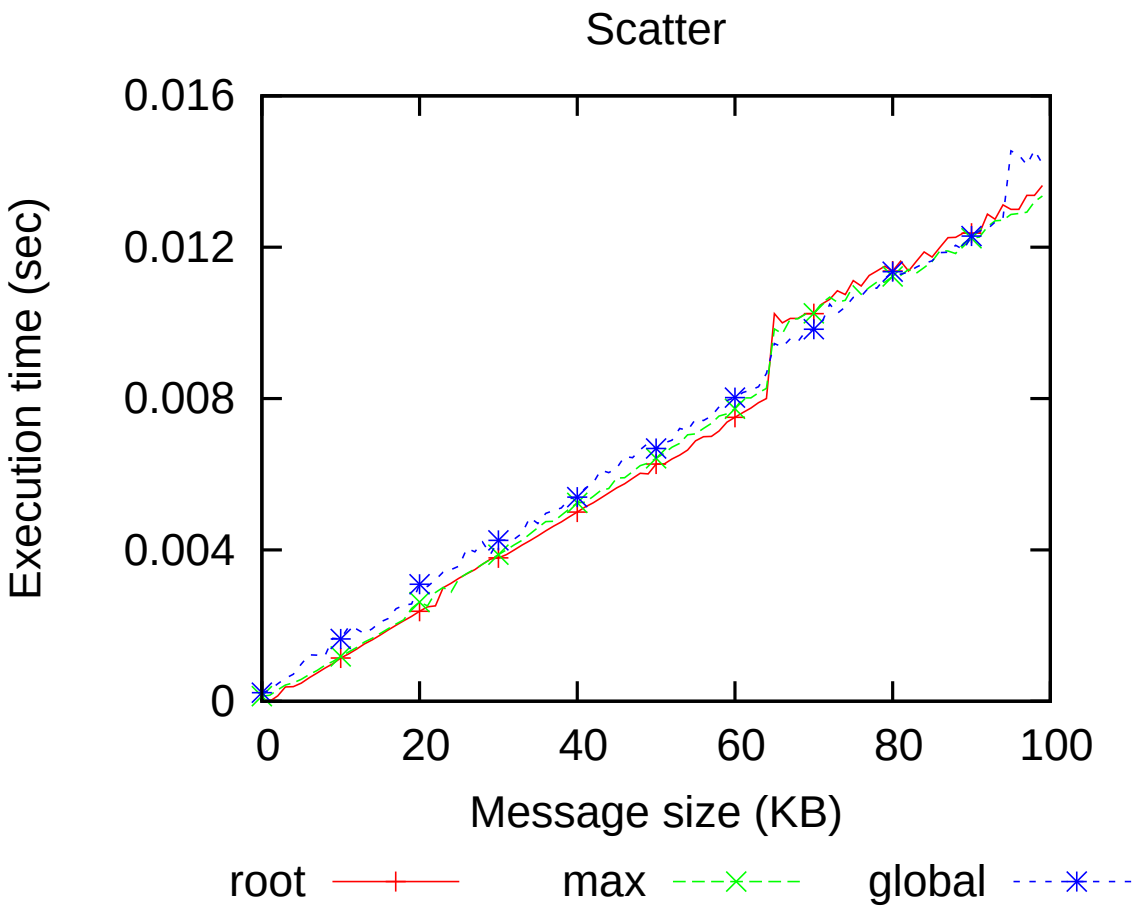


Figure 3.6: Comparison of different timing methods for native (linear) LAM scatter on 16 node heterogeneous cluster

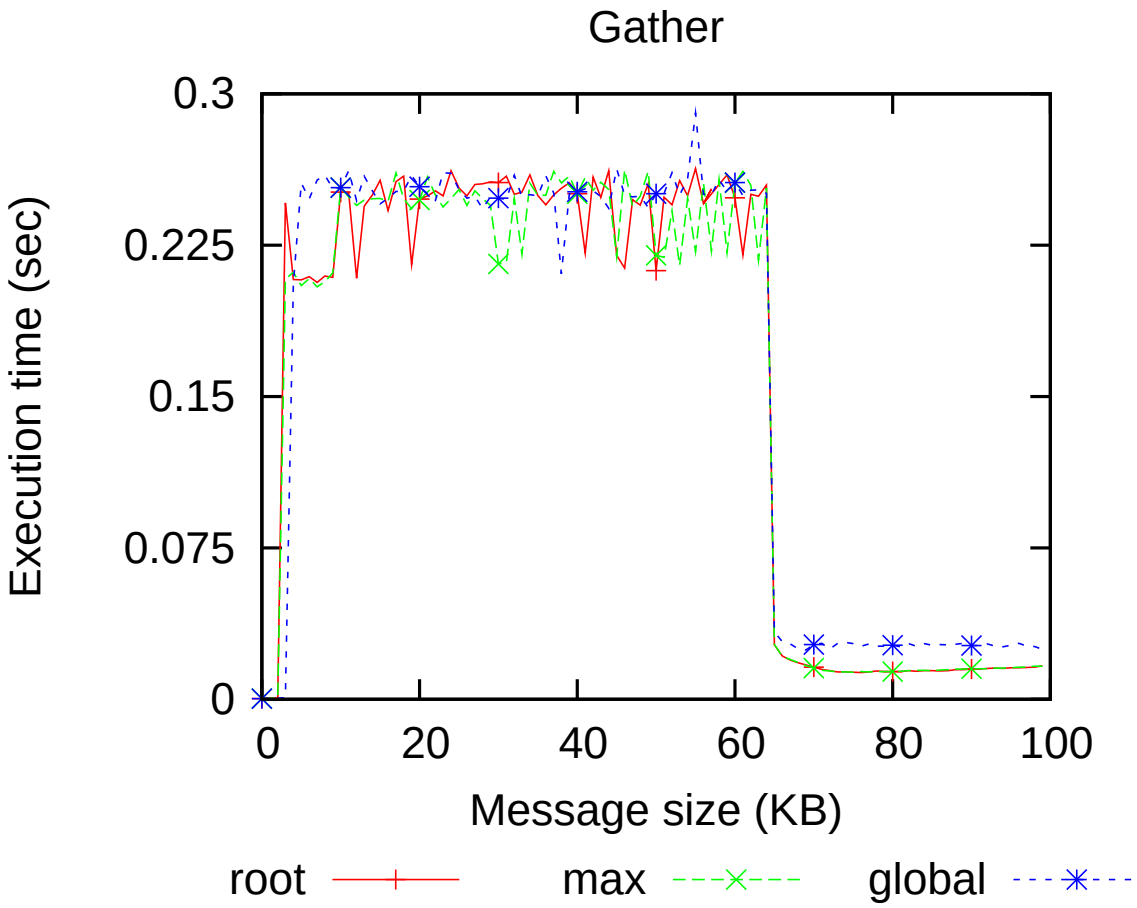


Figure 3.7: Comparison of different timing methods for native (linear) LAM gather on 16 node heterogeneous cluster

Table 3.1: The execution time of scatter and gather benchmarks with different timing methods on 16 node heterogeneous cluster.

Timing method	Scatter, 0..100KB, 1KB stride, 1 rep (sec)	Gather, 0..100KB, 1KB stride, 1 rep (sec)
Global	28.7	44.7
Maximum	0.8	15.6
Root	0.8	15.7

The importance of benchmarking collective operations for different message sizes is demonstrated in Lastovetsky *et al.* (2007), we reported on the observations of escalations of the execution time of gather caused by the use of TCP/IP layer in the communication stack with switched networks, as in Figure 3.7.

The maximum and root methods are generally as accurate as that with global-time (see Figures 3.6, 3.7) but much more efficient. The difference between overall scatter and gather execution times is caused by escalations of the execution time of gather for messages of middle sizes. It is interesting to note that the global timing method tends to smooth the non-deterministic readings in the gather in Figure 3.7 due to its averaging methods, and is thus less accurate that maximum and root timing methods for non-linear escalations.

We use MPIBlib to compare the cost of different methods of the measurement of native MPI scatter and gather operations on the target platform. For the cost comparison we measured the overall benchmarking times for scatter and gather operations. Table 3.1 shows the overall execution time of the benchmarks that use different timing methods and consist of one collective communication for each message size from *0* to *100 KB*, with *1 KB* stride. One can see that the global-time approach is very costly compared to maximum and root timing methods.

3.5 Summary

The key advantage of the MPIBlib software is that it is integrative as a library with the application sources. Previous benchmarking utilities are stand-alone and do not allow such ease of use with applications. It includes *p2p* and *collective* benchmarks that we will need for estimation of parameters for the performance model described in the next chapter. The code has an extensible form to allow for implementation of new operations for both point-to-point and collective, and allows for the designing of new communication experiments.

MBIBlib is efficient and accurate as shown by our experimental results. There is the option of running *p2p* in parallel or sequentially, to allow for extra efficiency.

The accuracy of previous benchmarking is questionable since it is assumed the time for each clock of all processors to be the same, ignoring any disparity that may arise from differences. Our software implementation MPIBlib addresses this problem to improve the measurement of parameters. The choice of timing method for collective benchmarks allows for the selection of ratio between accuracy and efficiency, with global timing being the most accurate but takes more time. We can chose this ratio to be most suited to our model selection, in this way the basis for the estimation is more accurate and hence a better performance model is possible.

Chapter 4

LMO: An Advanced Heterogeneous Communication Performance Model

This chapter presents a new model, the LMO model (Lastovetsky *et al.* (2006a), Lastovetsky & OFlynn (2007), Lastovetsky *et al.* (2009)), to predict the performance time of MPI collective communications. The performance model assesses execution times of MPI collective operations to assist applications designers in tuning programs for greater effectiveness. The traditional models use a small number of parameters to describe communication between any two processors and as a result the traditional point-to-point communication model can not be used intuitively to map collective operations. There are difficulties as they cannot be easily mapped to the communications in a heterogeneous switched network. This thesis seeks to address this issue and presents a new model that is designed to be more accurate as the solution. The essence of the problem is the nature of the parameters themselves. Traditional models such as Hockney use a very small number of parameters that do not allow the parts of communication to be separated out. This means building the collective operations is non-intuitive and difficult for the designers when modeling collective communications. The LMO model expresses the linear response of execution times as a function increasing message sizes, with a definition of regions of nonlinear escalation regions. The model is a simple and understandable combination of parameters that represent the heterogeneity of the system. It allows for the easy assessment of collective operations for MPI applications without an need for knowledge or changes to the underlying platform.

The model has key advantages over previous approaches:

- The new LMO model is a novel *intuitive performance model* with a new

mapping of parameters for MPI collective communications in an easy and clear way.

- The model is suitable for *heterogeneous* networks, and is more accurate and efficient than other previous models.
- The model is defined for a full range of message sizes, and includes previously undocumented ranges of sizes where *significant escalations* in execution times are found. No other model to date charts these critical areas of greatly elevated execution times.
- It is fully automated with parameter estimation at runtime with a software tool, the CPM, described in Chapter 6.

The second part of the chapter describes how the parameters of the model are found. The problem of heterogeneity is a key issue that is addressed for the first time by our model. Finding the individual parameter values for different nodes requires new innovative techniques for models designed for a heterogeneous network. The model requires that the fixed and variable parameters are found for each heterogeneous node. The model also has new threshold parameters that divide the model into that for small, medium and large message sizes. The greater number of parameters allows for the building of a intuitive model that reflects the switched network topology in an understandable way. The method for finding these parameters needs to be reasonably quick to allow for an efficient design tool for application developers. First we describe the parameters of the new model, and then how the point-to-point model is extended to map collective operations for MPI communications.

4.1 The Point-to-Point Parameters

The model we build is made of a simple collation of the parameters that represent the three stages of the point-to-point operation. The total execution time is the time for the processor to send the message, added to the transfer time while on the link to the other processor, and the time for the other processor to receive the message.

4.1.1 The Point-to-Point Model

The LMO model is based on six parameters characterizing the point-to-point communication. Like the most of traditional models, it represents the point-to-point communication time by a linear combination of its parameters and the

message size. The execution time of sending a message of M bytes from processor i to processor j in a heterogeneous cluster is estimated by a simple combination of the following parameters. The model is a simple addition of these constant and variable contributions from both the network and the two processors.

- C_i and C_j, are the fixed or *constant* processor delays;
- t_i, t_j, are the *variable* delays of the processors for each byte;
- L_{ij} is the latency, the *constant* contribution from the network;
- β_{ij} is the transmission rate, the *variable* network contribution.

The delay parameters, which are attributed to each processor, reflect the heterogeneity of the processors. The latencies and transmission rates, which correspond to each link, reflect the heterogeneity of communications. Notice how we distinguish between the processors and network contributions with separate parameters for the *constant* and *variable* parts of each for the point-to-point execution time.

Hence the execution time of sending a message of M bytes from processor i to processor j in a heterogeneous cluster ($i \xrightarrow{M} j$) is a straightforward summation, and is estimated by the execution time with parameters as below. The point-to-point execution time is the fixed processors sending and receiving times and link latency added to the variable delays due to variations in message size.

$$
\begin{aligned}
i \xrightarrow{M} j: \quad & (C_i, t_i) \xrightarrow{(L_{ij}, \beta_{ij})} (C_j, t_j) \\
\text{point-to-point execution time:} \quad & C_i + L_{ij} + C_j + M(t_i + \tfrac{1}{\beta_{ij}} + t_i) \\
\text{processor parameters:} \quad & \text{fixed } (C_i, C_j) \text{ and variable } (t_i, t_j) \text{ delays} \\
\text{network link parameters:} \quad & \text{latency } (L_{ij}) \text{ and transmission rate } (\beta_{ij}) \\
& \text{we suppose } L_{ij} = L_{ji} \text{ and } \beta_{ij} = \beta_{ji} \\
& 2(n + C_n^2) \text{ parameters}
\end{aligned}
$$

$$
\begin{aligned}
\text{Hockney:} \quad & \alpha_{ij} = C_i + L_{ij} + C_j \\
& \beta_{ij} = t_i + \tfrac{1}{\beta_{ij}} + t_j
\end{aligned}
$$

For networks with a single switch, it is realistic to assume $\text{Ł}_{ij} = \text{Ł}_{ji}$ and $\beta_{ij} = \beta_{ji}$. In terms of the Hockney model, the parameters can be expressed as a combination of constant and variable contributions that our model is able to separate fully.

The model is extended to map the performance of different collective operations. We notice that this is more complex than the simple collation of point-to-point communications. By examining the collective operations in detail we can see that they are in fact a combination of parallel and sequential communications for a single-switched network, see Figure 4.1. The collective communications operations such as Scatter, Gather, Broadcast and Reduce are standard MPI operations that applications developers wish to estimate to design applications. Our model maps these collective operations as two distinct mapping patterns:

- ***One-to-Many*** for the linear scatter
- ***Many-to-One*** for the linear gather

The model is built with a combination of heterogeneous parameters to give a linear representation of execution times for a message size.

The extended models also play a critical role in finding the estimations of the parameters themselves. We have an additional number of parameters for the new model that help make it more flexible and accurate in the representation of collective operations. There is a key problem with this approach however, as the traditional way of finding parameters using point-to-point communication experiments do not provide sufficient data for finding the new extended amount of parameters. We present an innovative solution to this problem, we use the extended LMO model for three processors to obtain extra data to find the additional parameter values. In this way the model for point-to-point that is extended to the One-to-Many and Many-to-One provides the solution to finding the extra LMO model parameters. This is a significant as previous traditional models were limited in their possible number of parameters as they were found by point-to-point only. This method is explained in more detail in the coming section for estimation in this chapter. First we describe the LMO model for collective communications in detail, divided into One-to-Many operations such as scatter, and Many-to-One operations such as gather.

4.2 The LMO Model for One-to-Many

The model is based on empirical observations of performance of collective operations on a single-switched network. By observing the collective operations directly, the behavior for that particular network may be modeled. Assessing an example of the modeling process for the linear scatter operation on a switched cluster in an intuitive way, we separate with our model the contribution of the root processor, the communication layer and each of the receiving processors. We can see that the scatter communication is mapped with two regions, one in serial

going toward the switch and then the messages leave the switch in parallel to each receiving node, see Figure 4.1.

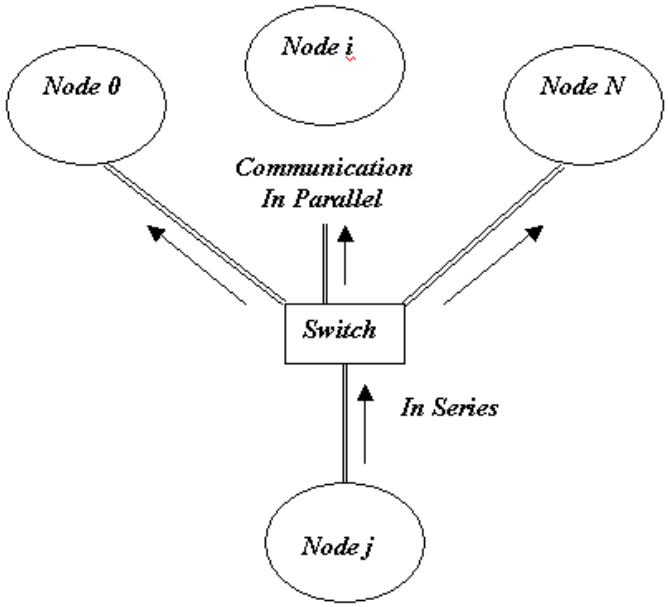


Figure 4.1: Mapping a One-to-Many Communication Operation, serial and parallel on a single switched network

Our model expresses the serialization of outgoing messages on the root processor followed by their parallel transmission over the communication layer and parallel processing on the receiving processors. This is a logical and intuitive approach that is described with difficulty in the corresponding traditional models, see the example of Hockney in Chapter 5. The key idea is a greater separation of the parts of the communication with a greater number of parameters. This allows for a simple combination of parameters that mapped to the collective operations in a clear and understandable way, as the extra parameters can represent both serial and parallel aspects of the collective communication. There is a noticeable difference in response with message size as messages increase for One-to-Many type collective operations. We therefore divide the model into two regions for small and large messages to represent this behavior, as in Figure 4.2.

The estimated time of operations such as *MPI_Scatter* for messages of size M from node 0 to nodes $1, ..., n$ is given by an intuitive combination of delay times from the sending processor to the switch, that is $C_0 + t_0 M$ added to the parallel communication from the switch to the receiving nodes, this means a

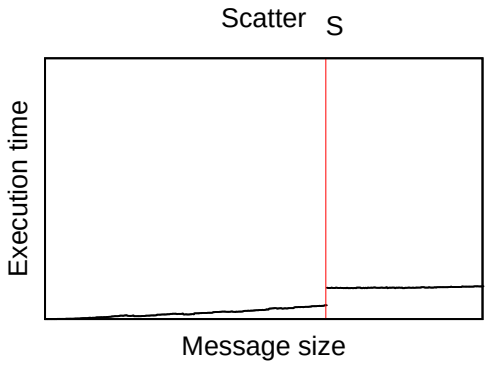


Figure 4.2: Threshold Parameters for One-to-Many Operations

simple addition of the processor's *serial* sending of n messages to the switch + *maximum* value of the *parallel* sending from the switch to the receiver nodes.

Hence taking the maximum of these parallel transfers and adding them together, the execution time for **small message** sizes is as follows:

$$(n-1)(C_r + Mt_r) + \max_{i=0,i\neq r}^{n-1}\{L_{ri} + C_i + Mt_i + \frac{M}{\beta_{ri}}\} \tag{4.1}$$

where smaller messages are less than a threshold value of $M < S$.

The sequential part of this formula,$(n-1)(C_r + Mt_r)$, is related to the root processor, which serially processes the messages to be sent to the rest *n-1* processors. The maximum reflects the parallel transmissions followed by the parallel processing on the receivers. Thus, this formula conforms to the features of network switches, which parallelize the messages addressed to different processors. On computational clusters with a TCP/IP layer, we did observe a leap in the execution time of the linear scatter, reflected in earlier versions of the model. However, the scale of the leap is not that significant and therefore in the improved version of the LMO model presented in this thesis, we use a linear approximation of the scatter execution time, which appears sufficiently accurate in practice. Despite this, we still estimate the message size, S, for which the leap is observed, as an important parameter used in the design of communication experiments for estimation of the LMO point-to-point parameters.

4.3 The LMO Model for Many-to-One

The model for standard MPI operations of type Many-to-One such as the linear gather is again based on the empirical observations that are particular to a net-

work platform. Figure 4.3 for a switched network shows that the linear response with increasing message size is different for small, medium and large messages.

For medium-sized messages with the Many-to-One operation we noted a very significant change in behavior from the linear response found for small and large messages. Based on the empirical readings from experiments it is indicated that the Many-to-One collective communication often experienced very significant delays as the message size was increased. This *nonlinear* behavior was manifested by very large fluctuations in execution times (Figure 4.3). These extreme values were observed to occur for different networks of various size, operating systems and MPI implementation. Previous models in related works (Chapter 2) all ignore this region of instability, our model provides a new and unique solution to this critical issue.

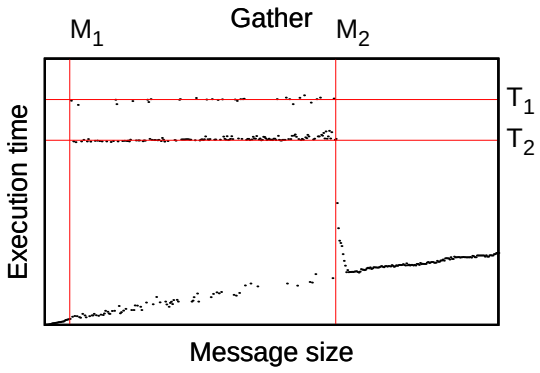


Figure 4.3: Many-to-One Non-linearity for Medium Messages Sizes

For large messages, the experimental observations show a return to a linear response of the execution time to message size. This change in behavior occurs when the size of the message exceeds some particular threshold value. Standard MPI implementation changes the sending to synchronous rather than asynchronous for large messages to allow reservation of resources for the communication operation. This provides an explanation for the sudden resumption of linearity for Many-to-One communications, as congestion is now avoided by resource reservation by underlying communication protocols prior to sending.

The model is therefore based on three part approach, to allow for the accommodation of the full range of small, medium and large message sizes. The thresholds for each size are determined by observations for the particular platform at installation, and the three message sizes give distinctive *linear* and *non-linear* responses, as follows:

1. ***Small*** message sizes, where the model is ***linear***.

2. ***Medium*** message sizes where the model gives thresholds for non-deterministic ***non-linear*** behavior, with significant increases in execution times.

3. ***Large*** Message sizes where ***linearity*** of response to increasing message size is resumed.

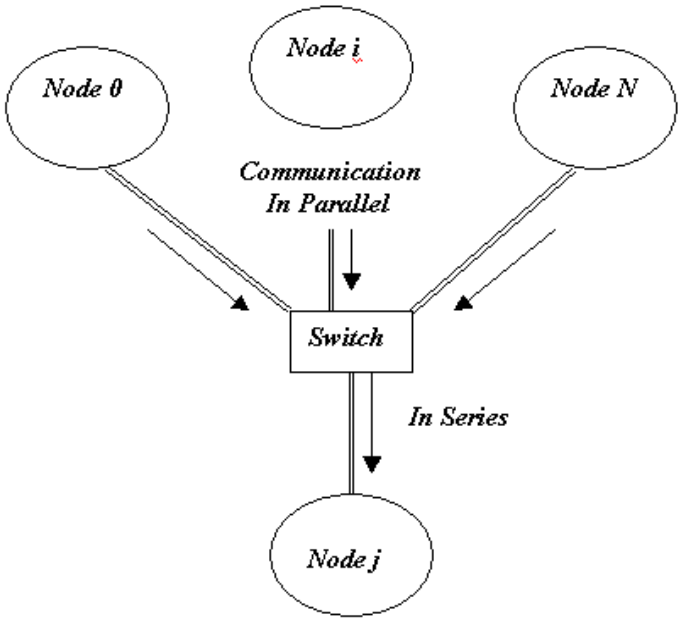


Figure 4.4: Communications both parallel and serial for Many-to-One Operation on a single switched network

For the operations such as *MPI_Gather* operation, we separate small, medium and large messages by introducing parameters M_1 and M_2. For small messages, $M < M_1$, the execution time has a *linear* response to the increase of message size. The model is a simple collation of parameters with mapping to the operation, on a switched network this means a simple addition, (see Figure 4.4) of the *maximum* value of the parallel sending from the switch from the sending nodes + receiver processor's serial receiving of n messages.

The execution time for **small messages** for the Many-to-One communication involving n processors ($n \leq N$, where N is the cluster size) is therefore estimated by:

$$(n-1)(C_r + Mt_r) + \max_{i=0,i\neq r}^{n-1}\{L_{ri} + C_i + Mt_i + \frac{M}{\beta_{ri}}\} \tag{4.2}$$

For **medium messages**, $M_1 \leq M \leq M_2$, we observed a number of extreme levels of escalation, and the response to increasing message size is *non-linear*, until the message size reaches the large message size threshold, M_2.

For **large messages**, $M > M_2$, the execution time resumes a *linear* predictability with increasing message size. Hence, this part is similar in design to the model for small messages, but has a different slope of linearity that indicates greater values due to extra communication overheads found for large messages. This is reflected in the *summation* of parameters as follows:

$$(n-1)(C_r + Mt_r) + \sum_{i=0, i\neq r}^{n-1} \{L_{ri} + C_i + Mt_i + \frac{M}{\beta_{ri}}\} \tag{4.3}$$

4.4 Estimation of Parameters

As the point-to-point communication experiments do not provide sufficient data for the estimation of the parameters, some particular collective experiments between small numbers of processors (in our experiments, between three processors) are introduced. To make use of the results of these additional experiments, the heterogeneous point-to-point performance model is extended by using the LMO model of these particular collective operations, with their execution time expressed via the point-to-point parameters.

We propose an elaborated approach to the estimation of the parameters of the advanced communication performance models such as LMO. Then we apply this approach to the extended LMO model and present a modified set of experiments required to estimate its parameters. This approach can be summarized as follows:

- A system of equations with the point-to-point parameters as unknowns and the execution times of the communication experiments as a right hand side is built and solved.

- Since more than two processors are participating in these additional experiments, the execution time should be measured by an appropriate timing method (Lastovetsky *et al.* (2008a)), which provides a reasonable balance between the accuracy and efficiency. We propose to measure the execution time of the collective experiments on the sender side. This method is proved fast and quite accurate for collective operations on a small number of processors.

- The additional collective communication experiments should be designed very carefully in order to avoid the irregularities in the execution time of the used collective operations. We suggest performing a preliminary test of

the collective operations for different message sizes to identify the regions of irregularities and avoid the use of message sizes from these regions.

- For reliable estimation of the parameters, we perform multiple repetitions of the experiments and statistical analysis of their results.

The cost of the accurate estimation of a communication model of the heterogeneous cluster can be quite significant as it typically involves multiple repetitions of the same communication experiments between different subsets of the processors, and then the statistical processing of their results for a reliable approximation of the parameters. As the efficiency of the estimation is an important issue, especially if the model is supposed to be estimated at runtime, we employ the following optimization techniques in the design of the experiments. The cost of the estimation can be significantly reduced if we simultaneously execute several independent communications involving non-overlapped sets of processors without degradation of their performance. On clusters based on a single switch, the parallel execution of the non-overlapping communication experiments does not affect the experimental results and can be used for acceleration of the estimation procedure. This optimization technique can be very efficient. For example, in our experiments on the 16-node heterogeneous cluster, the parallel estimation of the heterogeneous Hockney model with the confidence level 95% and relative error 2.5% took only 5 sec, while its serial estimation with the same accuracy took 16 sec. Both experiments give the same values of the parameters.

We applied this approach to estimation of the parameters of the extended LMO model. The modified set of communication experiments is similar to one that was proposed in Lastovetsky & Rychkov (2007). In addition to roundtrips, it includes the parallel communications between three processors $i \xleftrightarrow[M]{N} j,k$, which consist of the sending of M bytes from the processor i to the processors j,k and the receiving of the N byte replies. The execution time of this communication experiment can be represented as a sum of the execution times of linear scatter and gather, $T_{scatter}(M) + T_{gather}(M)$.

4.4.1 Estimation of the Threshold Parameters

The threshold parameters determine the range for small, medium and large message sizes for the Many-to-One model, and small and large ranges for the One-to-Many model. These parameters are very important as they distinguish different responses in execution time for the full range of message sizes. These are found from empirical measurements so that the model may be adapted specifically for a particular platform. To estimate the threshold parameters, we use experiments for *MPI_Scatter* and *MPI_Gather* to observe them for different message sizes.

The data rows for scatter and gather consist of the message sizes incremented stride and the measured execution time $\{M^i, T^i\}$, $M^{i+1} = M^i + stride$.

Typical data rows for heterogeneous clusters based on a switched network are shown in Figures 4.2 and 4.3. One can see that:

- the execution time of scatter can be approximated by the piecewise linear function with one break that corresponds to the threshold parameter S to be found;
- the execution time of gather has the regions of linearity for small, $M < M_1$, and large, $M > M_2$, messages and can also be approximated by the two linear functions.

The threshold parameter S for One-to-Many corresponds to the leap in the execution time, separating small and large messages. It may vary for different combinations of clusters and MPI implementations. We conducted the observation experiments for a full range of message sizes for the collective operations. We observed the leap in the execution time of scatter for large messages and the non-deterministic escalations of the execution time of gather for medium-sized messages (see Fig. 4.3).

These are the particular parameters for a range of message sizes, used to categorize the threshold the message size ranges where distinctly different behavior of the collective MPI operations is observed. The nature of these regions is *non-linear*, compared to the *linear* behavior of the smaller and larger message sizes of the thresholds, as found in the previous sections.

To find the threshold parameters, we use the algorithm proposed in Bai & Perron (1998). It considers the statistical linear models with multiple structural changes such as with our measurements for collective operations. These measurements may be found to change from linear to non-linear and then back to linear with increasing message size. The algorithm uses dynamic programming to identify optimal partitions with different numbers of segments. This is a complex method, but the essential idea is that the linearity of the data is examined segment by segment, and when the data is determined to change from linear to non-linear, the breakpoint is found. The algorithm allows us to locate the breakpoint in the execution time of scatter, S, and the range of large messages for gather, M_2. The R statistical package is used for analysis, Galassi *et al.* (2009). The function for finding the One-to-Many S parameter finds the execution time for the message sizes $0 < M < max\_size$ and performs the Bai & Perron algorithm, using the F statistic to find the goodness of fit of its regression process to the data. This algorithm finds the S breakpoint in the piecewise linear regression of execution times for message size.

The model computes the parameters of Many-to-One thresholds M_1 and M_2, it measures Many-to-One execution time for the message sizes $0 < M < max\_size$ and performs the Bai & Perron algorithm with the F statistic for the data. This is to find the M_2 breakpoint in the piecewise linear regression for execution times using the R statistical package. Then in a loop it measures Many-to-One execution time for the message sizes $0 < M < m$ with the stride reduced twice each time, where m is a message size for which a tenfold escalation of the execution time has been observed on the previous step. As stride reaches 1-byte value m is truncated to the nearest kb value.

4.4.2 Estimation of the Point-to-Point Parameters

To estimate the *constant* and *variable* parameters of the model for each heterogeneous node and link, we have an innovative approach to accurately evaluate the heterogeneity of each processor. For a network consisting of n processors, there will be $2(n + C_n^2)$ unknowns: that is n for fixed processing delays for each processor, n variable processing delays, and C_n^2 transmission rates and latencies. The execution time of the roundtrip, namely sending A_1 bytes and receiving B_1 bytes between nodes. The key problem is that the roundtrip experiments will give us only C_n^2 equations. The traditional approach to solving this problem to find the $2n + 2C_n^2$ parameters uses roundtrips with non-empty message, but this will give only C_n^2 linearly independent equations. This is a critical issue that has limited the design of analytical performance models to a small number of obtainable parameters. Therefore, the first challenge is to find a set of experiments that gives a sufficient number of linearly independent linear equations to find the extra parameters we want for heterogeneous communications on a switched network.

We begin with the constant parameters. Because we assume that the execution time of the copying of the root's block on the root processor is negligibly small, the constant parameters are estimated from the roundtrips and One-to-Two communications with empty message as in (4.4). The innovative approach we use here is to use the LMO model for One-to-Two communications as well as point-to-point measurements. This gives the extra simultaneous equations we need to solve for the greater number of parameters.

The expressions for the roundtrips (4.5) can be used to simplify the formula for the One-to-Two communication. The solution of the system of equations is

shown in (4.6).

$$\begin{cases} T_{ij}(0) = 2(C_i + L_{ij} + C_j) & i \xleftrightarrow[0]{0} j \\ T_{jk}(0) = 2(C_j + L_{jk} + C_k) & j \xleftrightarrow[0]{0} k \\ T_{ik}(0) = 2(C_i + L_{ik} + C_k) & i \xleftrightarrow[0]{0} k \\ T_{ijk}(0) = 2(2C_i + \max\limits_{x=j,k}(L_{ix} + C_x)) & i \xleftrightarrow[0]{0} jk \\ T_{jik}(0) = 2(2C_j + \max\limits_{x=i,k}(L_{jx} + C_x)) & j \xleftrightarrow[0]{0} ik \\ T_{kij}(0) = 2(2C_k + \max\limits_{x=i,j}(L_{kx} + C_x)) & k \xleftrightarrow[0]{0} ij \end{cases} \tag{4.4}$$

$$T_{ijk}(0) = 2(2C_i + \max_{x=j,k}(L_{ix} + C_x)) = 2C_i + \max_{x=j,k} T_{ix}(0) \tag{4.5}$$

$$\begin{cases} C_i = (T_{ijk}(0) - \max\limits_{x=j,k} T_{ix}(0))/2 \\ C_j = (T_{jik}(0) - \max\limits_{x=i,k} T_{jx}(0))/2 \\ C_j = (T_{kij}(0) - \max\limits_{x=i,j} T_{kx}(0))/2 \\ L_{ij} = T_{ij}(0)/2 - C_i - C_j \\ L_{jk} = T_{jk}(0)/2 - C_j - C_k \\ L_{ik} = T_{ik}(0)/2 - C_i - C_k \end{cases} \tag{4.6}$$

The variable parameters are found with help of the same communication experiments but with non-empty messages. Due to the irregularities of linear scatter and gather observed on switched clusters, the size of messages should be carefully selected in the One-to-Two experiments (Lastovetsky & Rychkov (2007)). We send the messages of medium size to avoid a possible leap in the execution time of scatter observed for LAM and Open MPI, and receive empty replies to eliminate the escalations in the execution time of gather. We build the system of equations (4.7). Having replaced some items by the point-to-point execution time, we obtain the expression (4.8) of the execution time of One-to-Two communication. The variable processor delays and transmission rates are found as in

(4.9).

$$\begin{cases} T_{ij}(M) = 2(C_i + L_{ij} + C_j + M(t_i + \frac{1}{\beta_{ij}} + t_j)) & i \xleftrightarrow[M]{M} j \\ T_{jk}(M) = 2(C_j + L_{jk} + C_k + M(t_j + \frac{1}{\beta_{jk}} + t_k)) & j \xleftrightarrow[M]{M} k \\ T_{ik}(M) = 2(C_i + L_{ik} + C_k + M(t_i + \frac{1}{\beta_{ik}} + t_k)) & i \xleftrightarrow[M]{M} k \\ T_{ijk}(M) = 2(2C_i + Mt_i) + \max\limits_{x=j,k}(2(L_{ix} + C_x) + M(\frac{1}{\beta_{ix}} + t_x)) & i \xleftrightarrow[0]{M} jk \\ T_{jik}(M) = 2(2C_j + Mt_j) + \max\limits_{x=i,k}(2(L_{jx} + C_x) + M(\frac{1}{\beta_{jx}} + t_x)) & j \xleftrightarrow[0]{M} ik \\ T_{kij}(M) = 2(2C_k + Mt_k) + \max\limits_{x=i,j}(2(L_{kx} + C_x) + M(\frac{1}{\beta_{kx}} + t_x)) & k \xleftrightarrow[0]{M} ij \end{cases} \tag{4.7}$$

$$\begin{aligned} T_{ijk}(M) &= 2(2C_i + Mt_i) + \max_{x=j,k}(2(L_{ix} + C_x) + M(\frac{1}{\beta_{ix}} + t_x)) \\ &= 2C_i + Mt_i + \max_{x=j,k}(T_{ix}(0) + T_{ix}(M))/2 \end{aligned} \tag{4.8}$$

$$\begin{cases} t_i = (T_{ijk}(M) - \max\limits_{x=j,k}(T_{ix}(0) + T_{ix}(M))/2 - 2C_i)/M \\ t_j = (T_{jik}(M) - \max\limits_{x=i,k}(T_{jx}(0) + T_{jx}(M))/2 - 2C_j)/M \\ t_j = (T_{kij}(M) - \max\limits_{x=i,j}(T_{kx}(0) + T_{kx}(M))/2 - 2C_k)/M \\ \frac{1}{\beta_{ij}} = (T_{ij}(M)/2 - C_i - L_{ij} - C_j)/M - t_i - t_j \\ \frac{1}{\beta_{jk}} = (T_{jk}(M)/2 - C_j - L_{jk} - C_k)/M - t_j - t_k \\ \frac{1}{\beta_{ik}} = (T_{ik}(M)/2 - C_i - L_{ik} - C_k)/M - t_i - t_k \end{cases} \tag{4.9}$$

4.4.3 Estimation of Point-to-Point Parameter Values

The values of each parameter are now found by executing these roundtrip and triplet experiments across the entire cluster. This gives rise to a full set of experiments that comprises of C_n^2 roundtrips and $3C_n^3$ One-to-Two communications.

As the parameters of our point-to-point model are found from a small number of experiments, they can be sensitive to inaccuracies of measurement. Therefore it makes sense to perform the series of the measurements for One-to-One and One-to-Two experiments for the entire cluster, and to use the averaged execution times in the corresponding linear equations.

The processing delays, C_i and t_i, can be obtained from C_{n-1}^2 different triplets, the processor i takes part in and can be averaged; the latencies, L_{ij}, and the transmission rates, β_{ij}, can be averaged from $n-2$ values:

$$\bar{C}_i = \frac{\sum\limits_{j,k\neq i} C_i}{C_n^2} \quad \bar{t}_i = \frac{\sum\limits_{j,k\neq i} t_i}{C_n^2} \quad \bar{L}_{ij} = \frac{\sum\limits_{j,k\neq i} L_{ij}}{n-2} \quad \bar{\beta}_{ij} = \frac{\sum\limits_{j,k\neq i} \beta_{ij}}{n-2} \tag{4.10}$$

The total execution time of the estimation of the parameters depends on:

- The number of measurements ($2C_n^2$ One-to-One and $3C_n^3$ One-to-Two measurements)
- The execution time of every single measurement (fast roundtrips between 2 and 3 processors), and
- The complexity of calculations ($3C_n^3$ comparisons, $12C_n^3$ simple formulas for calculation of the values of the parameters of the model, and $2(n + C_n^2)$ averagings).

Minimization of the total execution time of the experiments is another issue that we address. The advantage of the proposed design is that these series do not have to be lengthy (typically, up to ten in a series) because all the parameters have already been averaged during the process of their finding. Another optimization is related to the target platform, that is a switched cluster. All communication experiments are performed in parallel on non-overlapped pairs or triplets of processors. As network switches provide forwarding packets between sources and destinations without contentions, the parallel execution does not affect the accuracy of the estimation.

4.5 Summary

The LMO model is a new performance model for MPI communications operations that expresses the linear response of execution times as a function increasing message sizes. The model also includes the definition of regions of nonlinear

escalation regions, that are unrecognized by previous models. Previous models are restricted to a few parameters that do not map intuitively or easily to collective operations. Our model is a simple and understandable combination of parameters that can represent the heterogeneity of a system of different processors. It allows for the mapping of collective operations for MPI applications in a more intuitive way without an need for knowledge or changes to the underlying platform.

The method for determining the message threshold parameters allows for a new comprehensive approach to assessing the full range of message sizes for a given network. The parameters for threshold values demonstrate how the model can be used to determine the full range of message sizes, and is not as limited compared to more traditional models. These threshold parameters chart a distinct region of non-linear escalation that is determined by our model that was previously undocumented by methods to date.

The heterogeneity of the model means we have more parameters to represent it. We describe a new approach that performs a series of the measurements for One-to-One and One-to-Two experiments using the Many-to-One paradigm to estimate these parameters. This allows for sufficient linear equations to find the parameter values. The innovative method for node parameter determination exploits the new model itself, and we use parallelism to provide a quick and accurate assessment method.

Chapter 5

Comparison with Extended Heterogeneous Traditional Models

The chapter presents our heterogeneous versions of traditional models, Lastovetsky *et al.* (2009). We determine the new heterogeneous versions of the traditional homogeneous models, Hockney and LogP variants, and then we compare them to our new LMO model for accuracy and effectiveness. The chapter concludes with experimental results demonstrating that the LMO model can more accurately predict the execution time of collective operations than the traditional models.

In this chapter, we analyze the limitations of traditional communication performance models that are preventing them from the accurate estimation of the execution time of collective communication operations on computational clusters with a single switch. Usually, communication performance models for high performance computing are analytical and built for homogeneous clusters. The basis of these models is a point-to-point communication model characterized by a set of integral parameters, having the same value for each pair of processors. The execution time of collective operations is expressed as a combination of the point-to-point parameters and predicted for different message sizes and numbers of processors. For homogeneous clusters, the point-to-point parameters are found statistically from the measurements of the execution time of communications between any two processors. Typical experiments include sending and receiving messages of different sizes, with the communication execution time being measured on one side.

The design of traditional models based on point-to-point measurements can be extended for networks of heterogeneous processors in several ways. The performance models can be applied to heterogeneous clusters by averaging values obtained for every pair of processors. In this case, the heterogeneous cluster

will be treated as homogeneous in terms of the performance of communication operations. Another way is the heterogeneous extension of traditional models, where different pairs of heterogeneous processors are characterized by different parameters. The small number of parameters is an obvious advantage of the first approach. It allows the expression of the execution time of any communication operation by a simple compact formula, which is independent of the processors involved in the operation. While simpler to use, the homogeneous models are less accurate than the heterogeneous ones. When some processors or links in the heterogeneous cluster significantly differ in performance, predictions based on the homogeneous models may become quite inaccurate.

5.1 Extended Hockney Model

We start with a traditional model proposed by Hockney (1994). We can extend the Hockney model for heterogeneous clusters by introducing the different parameters α_{ij} and β_{ij} for different pairs of processors i, j. These heterogeneous parameters also combine the processor and network contributions in a similar way to the homogeneous model, $\alpha + \beta M$ (see Chapter 2 for the full model definition). The communication experiments are performed for each pair of processors in order to estimate the parameters of both the original and extended models for a heterogeneous cluster.

Let us consider how these models can be used for estimation of the execution time of MPI collective communication operations, for example for different algorithms of scatter. We begin with a simple linear algorithm, when messages are sent in a flat tree. There are only two ways to model this operation with this model. The first option is to assume that all point-to-point communications between the root and destination processors are performed sequentially. In this case, the total execution time will be expressed as a sum of *n-1* point-to-point execution times: $(n-1)(\alpha+\beta M)$ (homogeneous Hockney) or $\sum_{i=0,i\neq r}^{n-1}(\alpha_{ri}+\beta_{ri}M)$ (heterogeneous Hockney). The second option is to assume that the point-to-point communications are fully parallel. In that case, the predictions will be $\alpha+\beta M$ with the homogeneous Hockney models and $\max_{i=0,i\neq r}^{n-1}(\alpha_{ri}+\beta_{ri}M)$ with the heterogeneous version.

Unfortunately, both these assumptions do not accurately reflect the way the operation is executed on a switched cluster. On this platform, the linear scatter combines serial execution at the sending processor and parallel execution in the network and at the receiving nodes. Figure 4.1 in the previous chapter demonstrates this combination of serial and parallel communications within the

operation on a single switched network. The lack of parameters separating the contributions of the processors and the network in the Hockney model does not allow for expressing such effects, unlike the LMO model. For this reason we say the model lacks intuitiveness. As a result, both homogeneous and heterogeneous sequential Hockney predictions of the linear scatter are pessimistic, while their parallel counterparts are too optimistic. Because of the design of the Hockney model, the same formulas can be applied to the estimation of linear gather and presents the same issues of accuracy.

In general, heterogeneous extensions of traditional models can provide more accurate predictions of collective operations on heterogeneous platforms, at least for algorithms with some inherent parallelism. Examples of such algorithms are the algorithms of scatter and gather based on binomial communication trees. The communication tree for scatter/gather and *16* participating processors is shown in Figure 5.1. The nodes of the tree represent the processors. The arcs represent the logical communication links between the processors. Given *16* data blocks of the same size are to be scattered/gathered, each arc is marked by the number of blocks communicated over the corresponding link during the execution of the algorithm. With the use of the homogeneous Hockney model, the execution time of the binomial algorithm of scatter/gather can be approximated by the following formula by Chan *et al.* (2004): $(log_2 n)\alpha + (n-1)\beta M$, where M is a size of the receive (scatter) and send (gather) buffers. In each sub-tree, the largest messages $2^k M$ are sent first (scatter) or received last (gather), with k starting with $log_2(n-1)$. The formula includes parallel (constant contributions in sub-trees of the same order, $(C_k, k = 1, ..., 4)$ and sequential (accumulated variable contributions) parts.

In this formula communications in sub-trees of the same order are assumed simultaneous, which is unrealistic in the case of a heterogeneous cluster. The communication execution times in two sub-trees of the same order may be different. Moreover, the communication execution time associated with each sub-tree will also depend on mapping of the processor of the cluster to the nodes of the binomial communication tree. The homogeneous Hockney model is not detailed enough to express these nuances. At the same time, the use of the heterogeneous Hockney model allows us to propose the following more accurate formula for binomial scatter/gather:

$$T(k) = \alpha_{rs} + \beta_{rs} 2^{k-1} + \max_{c \in C_{k-1}} T_c(k-1) \tag{5.1}$$

where k is an order of the sub-tree (starts with $log_2 n$ - the whole tree), r is a root processor of the sub-tree (0, for the whole tree), and s is a root of a sub-sub-tree with the highest order (8, for the whole tree in Figure 5.1). $T_c(k-1)$ is the execution time of the sub-tree c of order $k-1$ from the set C_{k-1}. For the tree in

Figure 5.1, C_3 consists of two sub-trees, with roots 0 and 8. The execution times of sending/receiving of the largest block in each sub-tree are summed (sequential part). Maximums and recursion correspond to parallel communications in the sub-trees of the same height. For 8 participating processors with the root 0 the formula will look as follows:

$$\alpha_{04} + 4\beta_{04} + \max \left\{ \begin{array}{l} \alpha_{02} + 2\beta_{02}M + \max \left\{ \begin{array}{l} \alpha_{01} + \beta_{01}M \\ \alpha_{23} + \beta_{23}M \end{array} \right\} \\ \alpha_{46} + 2\beta_{46}M + \max \left\{ \begin{array}{l} \alpha_{45} + \beta_{45}M \\ \alpha_{67} + \beta_{67}M \end{array} \right\} \end{array} \right\} \tag{5.2}$$

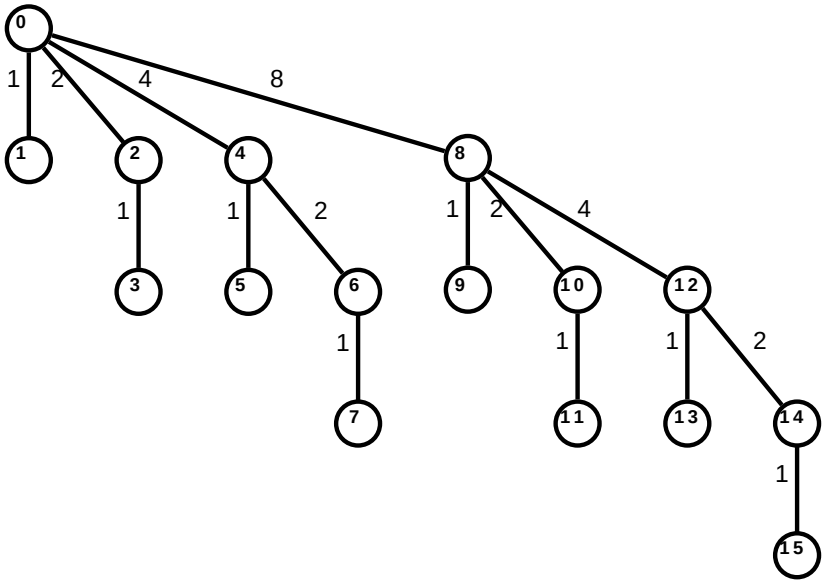


Figure 5.1: The binomial communication tree for scatter (gather) involving 16 processors. The nodes represent the processors. Each arc represents a logical communication link and is marked by the number of data block communication over this link

One can see that the formula for the homogeneous Hockney model is a special case of this formula. If all the point-to-point parameters are the same in the case of 8 processors, it will be rewritten as:

$$\alpha + 4\beta M + 2\beta M + \alpha + \beta m \approx log_2 8\alpha + (8-1)\beta M \tag{5.3}$$

An example of the advantages of the Hockney model with algorithms such as binomial with inherent parallelism is demonstrated by experiments with a 16-node cluster in Chapter 7.

5.2 Extended LogP-based Models

All the traditional models designs (as reviewed in Chapter 2) combine the contributions of different nature in a communications operation, as we will show next. The more elaborated traditional models such as LogP, LogGP and PLogP are shown to have this problem as well as Hockney.

The LogP-based models can be applied to heterogeneous clusters in the same ways as before the previous section, for Hockney. The parameters are found for all pairs of processors, with the above experiments being performed for each link. Then these parameters (heterogeneous version) or their average values (homogeneous version) will be used in modeling. However, there are options with how to build heterogeneous extensions of these models for heterogeneous clusters with single switch. For example, since the PLogP overheads, $o_s(M)$ and $o_r(M)$, correspond to the processor variable contributions, it is sensible to assume that they should be the same for all point-to-point communications in which the processor can be involved. This means that in the heterogeneous extension of the PLogP model, the average processor overheads should be used (averaged from the values found in the experiments between all pairs included the given processor). On the other hand, the latency, L, and the gap, $g(M)$, parameters (which are connected with the overheads in the design of above communication experiment) represent a combination of both processor and network contributions, and cannot therefore be averaged in this way. For this reason, it is not trivial and straightforward to extend the LogP-based models.

Let us consider how these models can be used for estimation of the execution time of linear scatter/gather. Similarly to the Hockney model, the execution time of both operations can be approximated by the same formulas: $L + 2o + (n-1)(M-1)G + (n-2)g$ (LogGP), $L + (n-1)g(M)$ (PLogP, Kielmann *et al.* (2000)), where M is a block size. Pješivac-Grbović *et al.* (2005) demonstrated that the analytical prediction of the execution time of collectives provided by these models was not accurate. There are major difficulties in expressing the execution time of these operations with the combination of heterogeneous parameters of the LogP-based models. The new LMO model allows for the separation of the constant and variable contributions of processors and network, that allows for an intuitive mapping of parameters to the communication operation. This elaboration on traditional models lead to more accurate prediction of the communication execution time. In Lastovetsky *et al.* (2006a), Lastovetsky &

OFlynn (2007), Lastovetsky *et al.* (2009) we proposed an analytical heterogeneous communication performance model, LMO, designed for both homogeneous and heterogeneous clusters based on a switched network. The model was described in detail in the last chapter, it includes the parameters that reflect the contributions of both links and processors to the communication execution time. This allows us to represent the aspects of heterogeneity for both links and processors and solves the problem of representing the serial and parallel aspects of the communications operations.

5.3 Experimental Results

In this section we present the experimental results demonstrating that our LMO model is more accurate than the other traditional models. We also show that our model is more efficient, that it is faster to build and it outperforms the others in accurate predictions.

We measured the execution time of point-to-point communications and compared it with the predictions provided by the heterogeneous LogGP, PLogP and LMO models. We present the experimental results obtained on the 16-node heterogeneous cluster described in Appendix A. For the measurements, we used the *MPIBlib*, the MPI benchmarking library, which provides accurate and efficient benchmarking of MPI communication operations Lastovetsky *et al.* (2008a). The LogGP and PLogP parameters were found for all pairs of processors with help of the *logp_mpi* library from Kielmann *et al.* (2000). In Figure 5.2, the results for one pair of processors are shown. The linear predictions provided by the LogGP and LMO models are almost the same and acceptably accurate for small and medium sized messages. The PLogP point-to-point model is piecewise linear. It includes a lot of empirical data stored in the functional parameters, and reflects the deviations of the execution time from the linear predictions.

All point-to-point models considered in this section use a lot of measurements and very simple computations. Therefore, the measurements are the most time consuming part in the finding of the parameters of these models. In Table 5.1, the number of measurements is estimated for each model and the time they take on 16-node heterogeneous cluster is shown. In Table 5.1, we compare the measurement costs of the point-to-point models of a heterogeneous cluster. For a cluster of n processors there will be C_n^2 single point-to-point communications. The parameters of the Hockney model for a single point-to-point communication are found by linear regression of k execution times of the roundtrips with different message sizes. Larger k provides a more accurate prediction. The execution time of each measurement depends on the message size. In our experiments, we used 10 message sizes ranging from 0 to 100 kb.

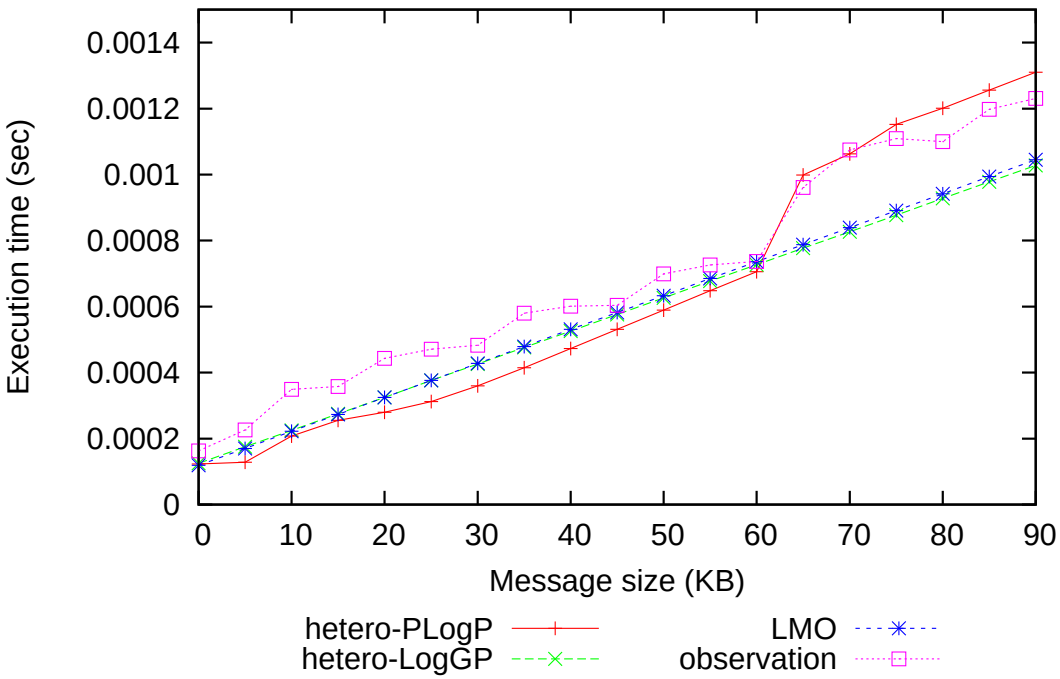


Figure 5.2: Comparison of the predictions of the point-to-point models for a single point-to-point communication.

Estimation of the PLogP parameters for each pair of processors includes:

- s experiments on saturating the link by empty messages, the i-th experiment of which consists of 2^i sendings, and
- $2mr$ experiments on $i \xleftrightarrow[0]{M} j$ and $i \xleftrightarrow[M]{0} j$ roundtrips, where: r is the number of roundtrips required to obtain more accurate send and receive overheads (the averaged execution time of the roundtrips) $i \xleftrightarrow[0]{M} j$ is also used for estimation of $g(M)$, and m is the number of message sizes, necessary for accurate piecewise linear approximation of the execution time of point-to-point communication.

The numbers s, r and m are found experimentally and can be different for different pairs of processors. In formulae in Table 5.1 that estimate the total number of measurements, we use the averaged values of s, r and m. The saturation experiments take much more time than single roundtrips as they include up to

2^s sendings. The direct measurements of the gap for each message size require $(m-1)s$ more experiments.

The LogGP model requires three saturation processes with message of 0, 1, and M bytes to estimate the gap values and two roundtrips with the message of 1 byte to estimate the values of the send/receive overheads. The execution time of the estimation of the parameters is omitted because they were found via the PLogP parameters as described in Kielmann *et al.* (2000).

The accuracy in our LMO heterogeneous communication point-to-point model is achieved by averaging the execution times in:

- a series of the k_0 measurements for each of C_n^2 empty roundtrips,
- a series of the k_1 measurements for each of C_n^2 One-to-One communications, and
- a series of the k_2 measurements for each of $3C_n^3$ One-to-Two communications.

In our experiments, no more than ten measurements in a series were needed to achieve the acceptable accuracy.

Table 5.1: Total Number of Measurements and Model Execution Times

Communication Model	Number of Measurements	Execution times
Hetero-Hockney	kC_n^2	0.17
Hetero-LogGP	$3sC_n^2 + 2rC_n^2$	-
Hetero-PLogP	$sC_n^2 + 2mrC_n^2 + msC_n^2 + 2mrC_n^2$	63.11
LMO	$k_0C_n^2 + k_1C_n^2 + k_2 3C_n^3$	0.33

The LMO model provides the more accuracy (as shown in Figure 5.2) and can be efficiently estimated on heterogeneous clusters. The next section further demonstrates how our LMO model outperforms the traditional extended models.

5.3.1 A Comparison of Model Predictions

Next we present the experimental results comparing the models predictions for execution time of linear scatter and gather for the 16-node heterogeneous cluster, with a single Ethernet switch and LAM 7.1.3 specified (as in Table, see Appendix A). We developed the software tool that provides the estimation of parameters of the LMO model and the heterogeneous extensions of the Hockney, PLogP and

Table 5.2: The Prediction of the Execution Time of Linear Scatter and Gather.

Model	**Linear scatter prediction**	**Linear gather prediction**
Hetero-Hockney	$\sum_{i=0,i}^{n-1}(\alpha_{ri}+\beta_{ri}M)$	
LogGP	$L+2o+(n-1)(M-1)G+(n-2)g$	
PLogP	$L+(n-1)g(M)g$	
LMO	$(n-1)(C_r+Mt_r)+$ $\max_{i=0,i\neq r}^{n-1}(L_{ri}+C_i+t_iM+M(\frac{1}{\beta_{ri}}+t_i))$	$(n-1)(C_r+Mt_r)+$ $\begin{cases}\max_{i=0,i\neq r}^{n-1}(L_{ri}+C_i+t_iM+M(\frac{1}{\beta_{ri}}+t_i)) & M<M_1\\ \sum_{i=0,i\neq r}^{n-1}(L_{ri}+C_i+t_iM+M(\frac{1}{\beta_{ri}}+t_i)) & M>M_2\end{cases}$

LogGP models (Lastovetsky *et al.* (2008b), Lastovetsky *et al.* (2009)). Details of the software tool are described in Chapter 6. Using the models parameters, this tool predicts the execution time of different algorithms of collective communication operations. In this section we present the experimental results of the modeling of scatter and gather. The communication execution time was measured with help of the MPIBlib benchmarking library (Lastovetsky *et al.* (2008a)) with the confidence level 95% and the relative error 2.5%.

The expressions of the execution time of linear scatter and gather are summarized in Table 5.2. Only the LMO model provides two different formulas for scatter and gather, reflecting steeper slope in the execution time of gather observed for large messages. Only the LMO model reflects the irregular behavior of linear gather. On computational clusters with a TCP/IP layer (including the cluster specified above), we observed a leap in the execution time of linear scatter (see Figure 5.3, observation, 64KB). In the previous version of the LMO model, we included the extra parameter that reflects this leap. However, for larger messages, these leaps regularly repeated, converging to the line with the same slope. We could have included multiple empirical parameters for the LMO model and have presented the execution time of scatter as a piecewise linear function, but due to the not very significant values of the leaps and for simplicity, we considered only the linear model, which satisfactorily approximates the observed execution time of the native (linear) LAM scatter. The PLogP prediction provides the same accuracy for medium size messages and also reflects the leap in the execution time, after which it diverges from the observations. The estimations of other

traditional models are inaccurate.

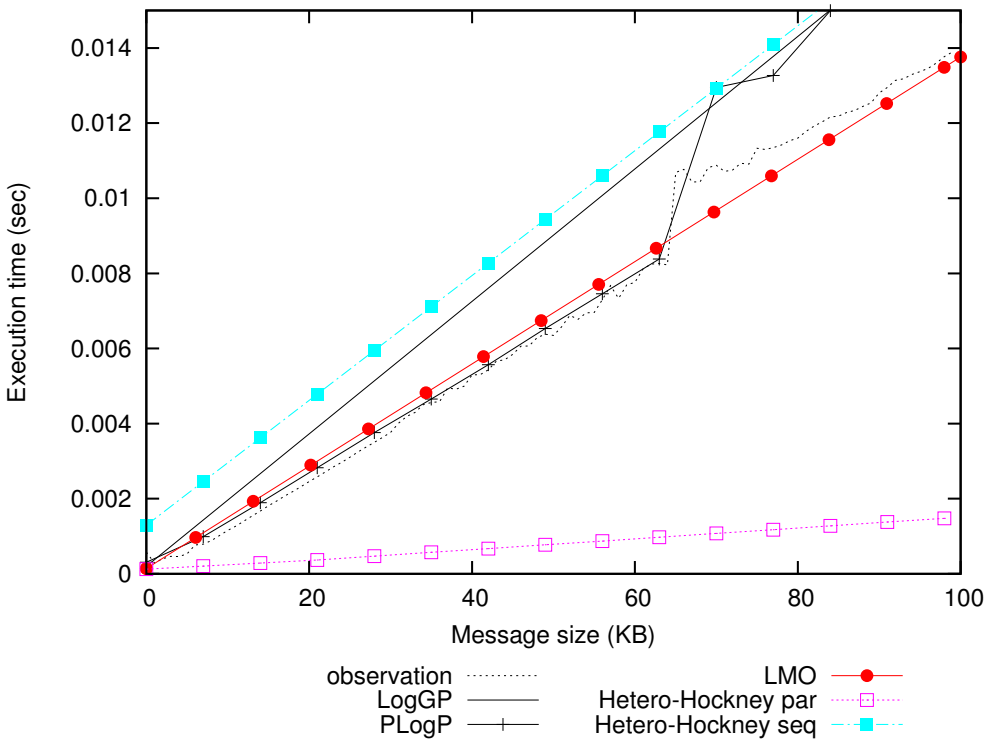


Figure 5.3: The prediction of the execution time of linear scatter on the 16-node heterogeneous cluster

The LMO includes not only analytical but also empirical parameters. For small (less than 4KB) and large (more than 65KB) messages, the execution time of linear gather is represented as two lines with different slopes (Figure 5.4). For medium size messages, the LMO model defines the message range for non-linear responses, $M_1 < M < M_2$. Therefore only the LMO prediction reflects the irregularity in the execution time of the native (linear) LAM gather. With regard to the intervals for small and large messages, traditional models make better predictions of the execution time of gather rather than of scatter.

Figure 5.3 demonstrates that the intuitive prediction provided by the LMO model is more accurate than the predictions of the traditional models. The accurate prediction of the execution time allows for a correct decision by the LMO model to be made regarding the switching between the algorithms of a collective communication operation. In Figure 5.5, the predictions provided by the heterogeneous Hockney and LMO models are presented for the linear and binomial algorithms of scatter for messages $100KB < M < 200KB$. Similarly

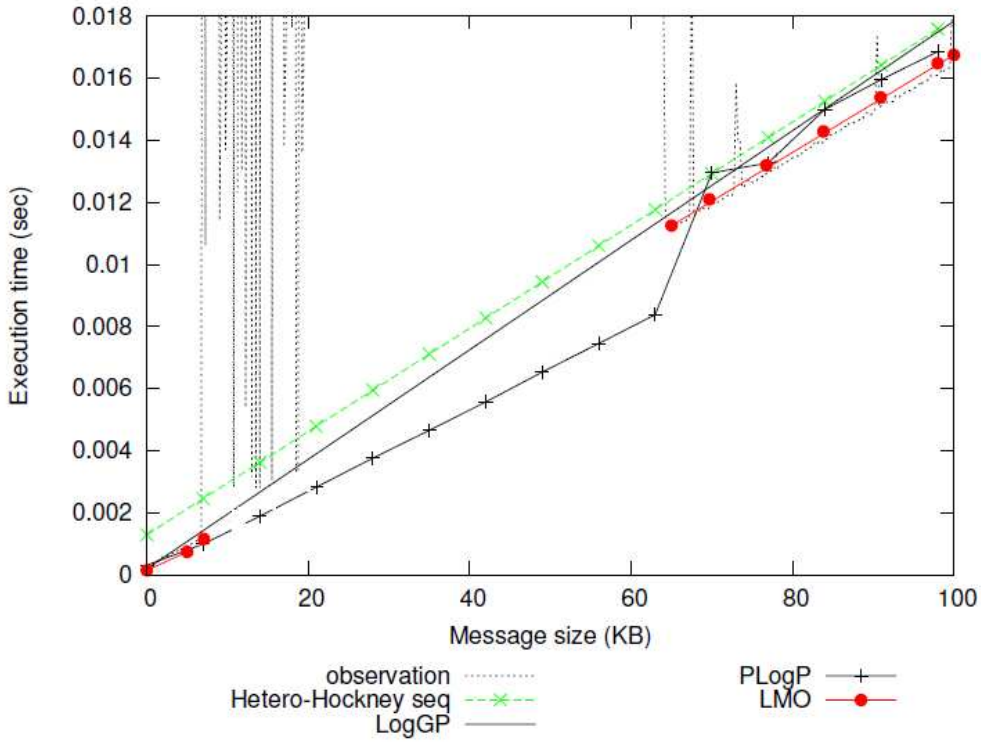


Figure 5.4: The prediction of the execution time of linear gather on the 16-node heterogeneous cluster.

to Pjesivac-Grbovic *et al.* (2005), the Hockney model miscalculates that the binomial algorithm outperforms the linear one, switching in favor of the first, whereas the decision based on the LMO approximation would be correct. The accurate prediction can be a basis for the model-based optimization of collective operations.

5.4 Summary

We can apply the traditional models to heterogeneous clusters in two ways. Firstly, by averaging values obtained for every pair of processors, the heterogeneous cluster will be treated as homogeneous in terms of the performance of communication operations. Second, there are the heterogeneous extensions of traditional models, found from taking pairs of heterogeneous processors for measurement. This approach gives rise to different parameters. The first approach has a small number of parameters in comparison and the execution time is rep-

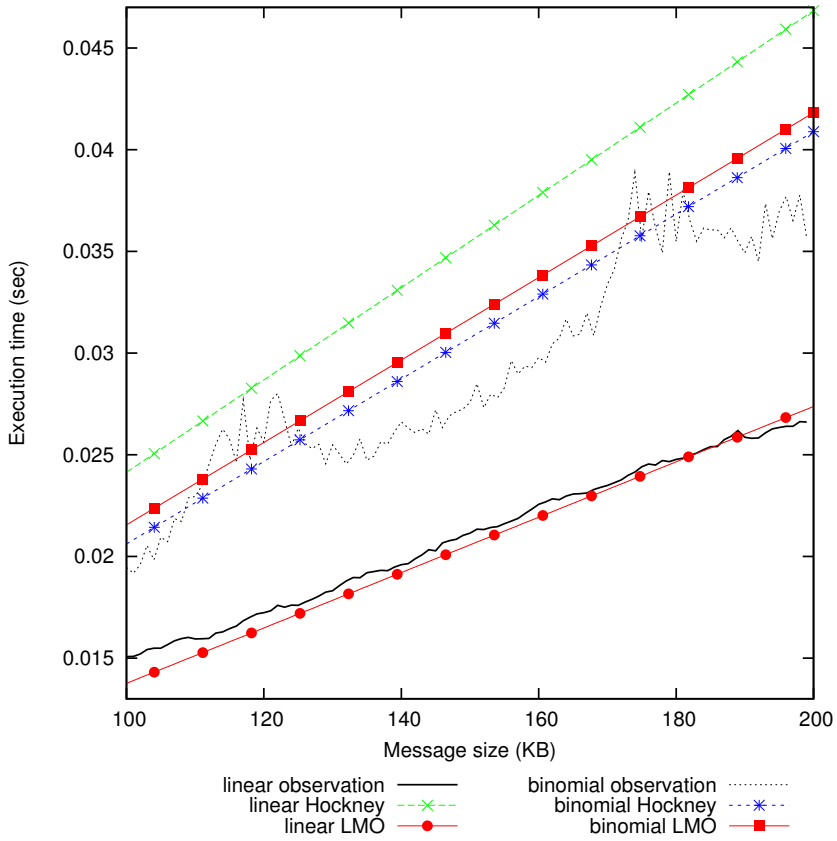


Figure 5.5: The performance of the linear and binomial algorithms of scatter vs the heterogeneous Hockney and LMO predictions.

resented by a simple compact formula. The homogeneous models are shown to be less accurate than the heterogeneous approach however from our experiments on a *16* node single switched network.

Our LMO model is shown to be more accurate than the heterogeneous extension models. It is an intuitive approach that allows for representation of both serial and parallel parts of the communication operations on a switched network. It is also more accurate because it is based on an empirical approach that is adaptive and specific for a particular heterogeneous network platform. The traditional models are purely analytical and cannot chart the nonlinear areas of performance degradation found empirically for a range of message sizes.

Chapter 6

A Software Tool for the Estimation of Heterogeneous Communications Models

We present the software tool, the *CPM* (Communication Performance Model), that automates the estimation of the heterogeneous communication performance models of clusters based on a switched network, (Lastovetsky *et al.* (2008b), Lastovetsky *et al.* (2009)). The software tool is implemented in *C/C++* on top of MPI. The package consists of a library and a model builder (Figure 6.1). The library implements the main functionality of the software. The model builder estimates the parameters of the heterogeneous communication performance models with given accuracy and stores the parameters in files. The results of measurements can be visualized by the gnuplot utility (Williams & Kelley (2007)). There is an API for the estimation of heterogeneous communication performance models with a universal interface that is designed to be extensible. The *LMO* model and heterogeneous extensions of two traditional models, *Hockney* and *PLogP* are currently supported.

The library consists of three main modules: Measurement, Models and Model-based collectives, which are presented in Figure 6.2 and described in the following subsections.

6.1 The Measurement module: benchmarking specific communication experiments

The Measurement module is responsible for the measurement of the execution time of the communication experiments required for estimation of the parameters of heterogeneous models. It is used along with the *MPIBlib* benchmarking

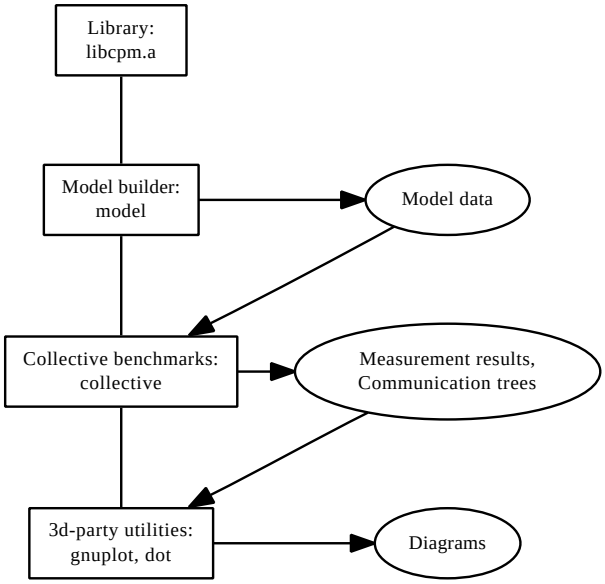


Figure 6.1: Library Structure

library (Lastovetsky *et al.* (2008a)). The point-to-point and collective benchmarks provided by *MPIBlib* are directly used for the estimation of parameters of the heterogeneous *Hockney* (point-to-point benchmark) and the *LMO* models (point-to-point, scatter, gather benchmarks). Using the *MPIBlib* interface, the Measurement module implements the function for measuring the execution time of the point-to-two communication experiments, $i \xleftrightarrow[0]{0} jk$ and $i \xleftrightarrow[0]{M} jk$. To provide reliable results, the communication experiments in each benchmark are repeated for either a fixed or variable number of times.

The Measurement module implements a function for benchmarking of the point-to-two communications on the top of the *MPIBlib* library, required for the estimation of the LMO point-to-point parameters (as described in Chapter 4):

```
void CPM_measure_p2pp(MPI_Comm, int M, int parallel,
     MPIB_precision precision, MPIB_result** results);
```

One point-to-two communication experiment consists of a combination of the blocking send and receive operations. Similar to the point-to-point benchmark, measurements can be performed either serially or in parallel, which is specified by the parallel argument. In the parallel mode, as many as possible point-to-two

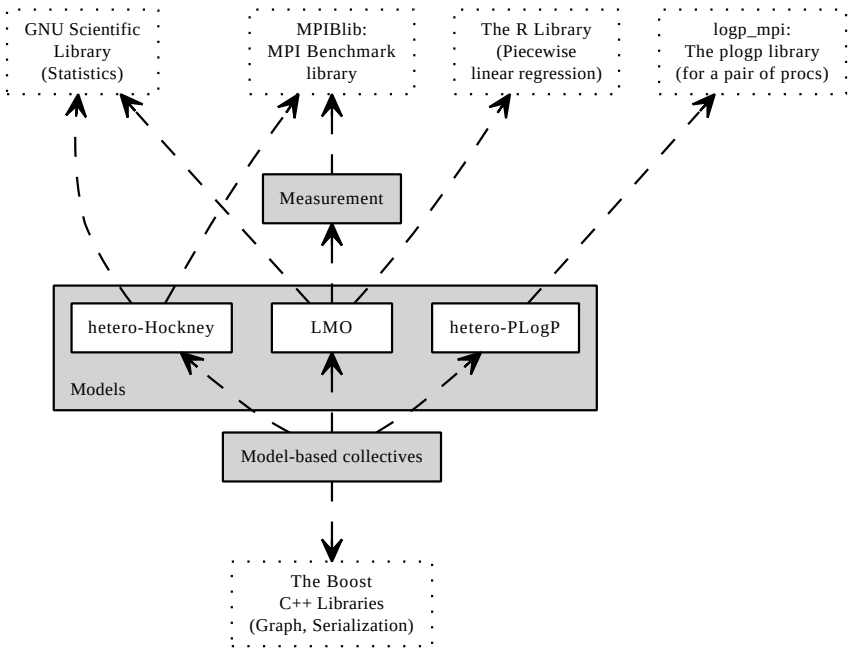


Figure 6.2: Model-based collective operations

communications on independent triplets of processors are performed simultaneously. During the estimation of the LMO point-to-point parameters, this function is called twice, with an empty and non-empty messages, in the high-precision mode. This decreases the execution time the benchmark takes and returns quite accurate results for clusters with a single switch. For other platforms, these benchmarks can be performed in sequential mode also supported by these benchmarks. The Measurement module along with the *MPIBlib* library provides the basis for estimation and modeling of the communication execution time.

6.2 The Models Module: API for heterogeneous communication performance models

The Models module provides an API for estimation of the heterogeneous communication performance models and prediction of the execution time of collective algorithms with these models. Apart from the *LMO* model, this module supports the heterogeneous extensions of the traditional model *PLogP*. For each model, it provides a function for estimation of the model and a set of functions for prediction of the execution time of different MPI communication operations.

The heterogeneous extension of the *Hockney* model can be estimated by one of the following functions implementing both methods discussed in Chapter 5:

```
void Hockney_estimate(MPI_Comm comm, int M, MPIB_precision
                precision, int parallel, Hockney_model** model);

void Hockney_estimate_regression(MPI_Comm comm, int min_size, int
            max_size, int max_num, int parallel, Hockney_model**
            model);
```

In the first function, the point-to-point parameters are found directly from two series of roundtrips with *0* and M bytes, and the accuracy of the returned estimate depends solely on the precision argument, which specifies the number of repetitions for each roundtrip. In the second function, based on linear regression analysis, the parameters are found from a series of single roundtrips, each with a different message size. In this case, the accuracy of the returned estimate is determined by the set of message sizes. The message sizes are found adaptively, during the execution of the function. The corresponding adaptive procedure is controlled by the following arguments:

- The minimum and maximum message sizes, *min_size* and *max_size*, and
- The maximum number of message sizes, *max_num*.

This procedure can be summarized as follows:

- The execution time of the roundtrip with the message size M_k is compared with the linear model based on the times for smaller messages $M_{k-1}...M_0$.
- The number of message sizes is limited by the *max_num* argument. Linear regression analysis is implemented with help of the gsl library, Galassi *et al.* (2009). In both estimation functions, the measurements can be preformed in parallel.

The output of the estimation functions, *model*, is a data structure containing the parameters of the estimated model which can be used, in particular, as the input argument of the predicting functions. A function predicting the execution time of a communication operation Y has the following interface:

```
double Hockney_predict_Y(Hockney_model* model, args);
```

For example, the point-to-point execution time will be predicted by the following function:

```
double Hockney_predict_p2p(Hockney_model* model, int i, int j,
   int M) {
      int index = IJ2INDEX(model->n, i, j);
      return model->a[index] + model->b[index] * M;
 }
```

For the **heterogeneous extension of the PLogP model** we can use the *logp_mpi* library, Kielmann *et al.* (2000), that has functions estimating the PLogP parameters for a pair of processors. These functions are used in the implementation of the following estimator of the heterogeneous PLogP model:

```
void PLogP_estimate(MPI_Comm comm,
  int min_size, int max_size, int max_num,
  MPIB_precision precision, int parallel, PLogP_model** model);
```

The above function calls the *logp_mpi* estimation function for all pairs of processors either sequentially or in parallel. In the parallel mode, the estimations for as many as possible independent pairs of processors will be performed simultaneously. The precision argument of the *PLogP_estimate* function defines the precision of measurements. The message set for which the parameters (the piecewise linear functions) are estimated, is defined by the four integer arguments, similar to the *Hockney_estimate_regression* function. The output of the estimation function, model, is used in the set of the functions that predict the execution time of different communication operations, using the heterogeneous extension of the PLogP model.

The following function automates the estimation of the LMO model, which has been described in Chapter 4:

```
void LMO_estimate(MPI_Comm comm,
  int min_size, int max_size, int max_num,
  MPIB_precision precision, int parallel, LMO_model** model);
```

The procedure implemented in this function consists of the following steps:

1. First, the linear scatter and gather benchmarks are performed once for each adaptively selected message size.

2. Then, the obtained results of measurement are used to find the threshold parameters, S, M_1, M_2 with help of the *R strucchange* library (Zeileis *et al.* (2002)). The library automates the detection of the structural changes in piecewise linear regression models. This is finding the areas of nonlinear escalations in execution times for message sizes.

3. After that, the point-to-point and point-to-two benchmarks are performed in the high-accuracy mode for an empty message and a message of the size slightly less than the value of the threshold parameter S.

4. Finally, the point-to-point parameters are found by building and solving the systems of linear equations based on the results of the benchmarks performed at step 3.

The output of the estimation function, *model*, contains the LMO parameters and may be used in functions predicting the execution time of different communication operations. Unlike the predicting functions for other models, the LMO uses not only analytical but also empirical parameters. This allows it to reflect these significant irregularities in the communication execution time for some message sizes.

6.3 Use of the Software Tool

Next we show how the software tool can be used by application programmers using the "*cpm.h*" library with their parallel applications. Using the *CPM* software tool the programmer can include the communication performance of an application. In the following code, the *LMO* model may be used as the contents of the '*model*' data structure:

```
1:  #include "cpm.h"
2:  int main(int argc, char** argv) {
3:    MPI_Init(&argc, &argv);
4:    LMO_model* model;
5:    if (to_build) {
6:        LMO_estimate(MPI_COMM_WORLD, msgset, precision, parallel,
7:            &model);
8:    }
9:    else {
10:     if (rank == 0)
11:       LMO_read(istream, &model);
12:     LMO_bcast(model, MPI_COMM_WORLD);
13:   }
....
14: MPI_Finalize();
15: }
```

In line 4, a variable for the *LMO* model is defined. Then the model is estimated at runtime (lines 5-8) or read from the file and broadcast to all processors (lines 9-13). The *LMO* model can be used to estimate the communication time, (line 6), some additional cost will be incurred if the model is estimated at runtime.

The software tool also provides a **stand-alone** utility that is run from the command line that consists of the following:

- The model builder (tools/model.c) that performs measurements, estimates parameters of the model and stores the model data to an output file.

Usage:

```
$ mpirun [mpi options] model -M LMO -o lmo.mod > lmo.out
```

where the options are the same as those for the *MPIBlib* software use in Chapter 3, with the addition of the specification of the model type is the *-M* option and the *-o* option for the name of the model data file that is created, *lmo.mod* above. The results can then be displayed with *gnuplot*.

6.4 Summary

The software tool described in this chapter provides an estimation of parameters of heterogeneous communication models. The user can control the precision of the model parameters and the total building time. The software tool supports both traditional and advanced communication performance models. The design of the tool is very flexible. It can be extended in different directions, such as the support of other communication performance models, conduction of additional communication experiments required to estimate the model parameters. It can also be used in the implementation of new collective communication algorithms with optimization, as described in the next chapter.

Chapter 7

Model Based Optimization of Collective Operations

In this chapter we show how the advantages of heterogeneous communication models and how they can optimize collective operations. We demonstrate their usefulness by showing how they can significantly improve the design of parallel applications for better performance. We present three examples to show how they may be applied. We start with a traditional model proposed by Hockney (1994) that we have extended with our *CPM* software tool (Chapter 5 and 6) for heterogeneous clusters. The example demonstrates the improvements found with the heterogeneous version of the Hockney model when compared to the homogeneous version.

The second example is one of optimization that is using the *LMO* model with collective communications operations scatter and gather. The model allows the application designer to select message sizes that will avoid the significant escalations in execution time observed on certain platforms. The experiments demonstrate the improvements by comparing the performance of optimized and ordinary collective operations. As a final example there is an application with a matrix multiplication. We show how the *LMO* model is used to optimize collective operations by the selection of communication algorithm to avoid regions of performance degradation due to escalations in execution time for some message sizes.

7.1 Extended Hockney Models

To demonstrate the advantage of the heterogeneous version of the Hockney model Hockney (1994), we compared the prediction of execution time for the scatter operation to the homogeneous Hockney prediction. We show how the linear scat-

ter algorithm, and then binomial scatter with homogeneous and heterogeneous Hockney model predictions compared to each other and to actual results. In this example we consider how a traditional model can be used for estimation of the execution time of MPI collective communication operation scatter. We have extended the Hockney model for heterogeneous clusters and introduced different parameters α_{ij} and β_{ij} for different pairs of processors, which also combine the processor and network contributions, (see Chapter 5 for details of applying Hockney to linear and binomial algorithms).

We conducted experiments on a 16-node heterogeneous cluster (see Appendix A for specification of each of the seven node types of the cluster). First we found the parameters of the homogeneous Hockney model of this cluster: $\alpha = 63.9\mu s$, $\beta = 0.00967\mu s$. The execution time of the scatter predicted by this model will be the same regardless of which processor will be the root of the operation and how the remaining processors will be mapped to the nodes of the communication tree. Therefore given the root is fixed, according to the homogeneous Hockney model, any mapping of the remaining processors will be equally optimal. Then we found the parameters of the heterogeneous Hockney model of this cluster. Table 7.1 presents these parameters for the links between different types of the nodes (the first value is α measured in μs, the second one is β, $\mu s/B$). The "X" sign indicates that there are no communication links between nodes of the corresponding types (three nodes have a unique type).

Node	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	63.9	124	61.2	121	60.7	61.4	61.3
	0.00967	0.00941	0.00932	0.0172	0.00948	0.00950	0.00939
2		122	58.3	122	60.8	57.9	60.4
		0.00963	0.00932	0.017	0.00934	0.00941	0.00934
3			39.4	61.8	43.1	46.3	38.1
			0.0151	0.00914	0.00918	0.00923	0.00908
4				X	61.2	60.9	61.2
				X	0.0146	0.0143	0.0142
5					X	59	48.8
					X	0.0488	0.00923
6						X	53.3
						X	0.00923
7							36.1
							0.00921

Table 7.1: Specification of the parameters of the heterogeneous Hockney model of the 16-node heterogeneous cluster.

We compared the observed execution time of the linear and binomial scatter with the homogeneous and heterogeneous Hockney predictions (Figure 7.1, 7.2). With the fixed root processor (Dell Poweredge SC1425, 3.6 Xeon, 800MHz, 2MB), we used the heterogeneous Hockney model to predict the execution time of the linear and binomial scatter.

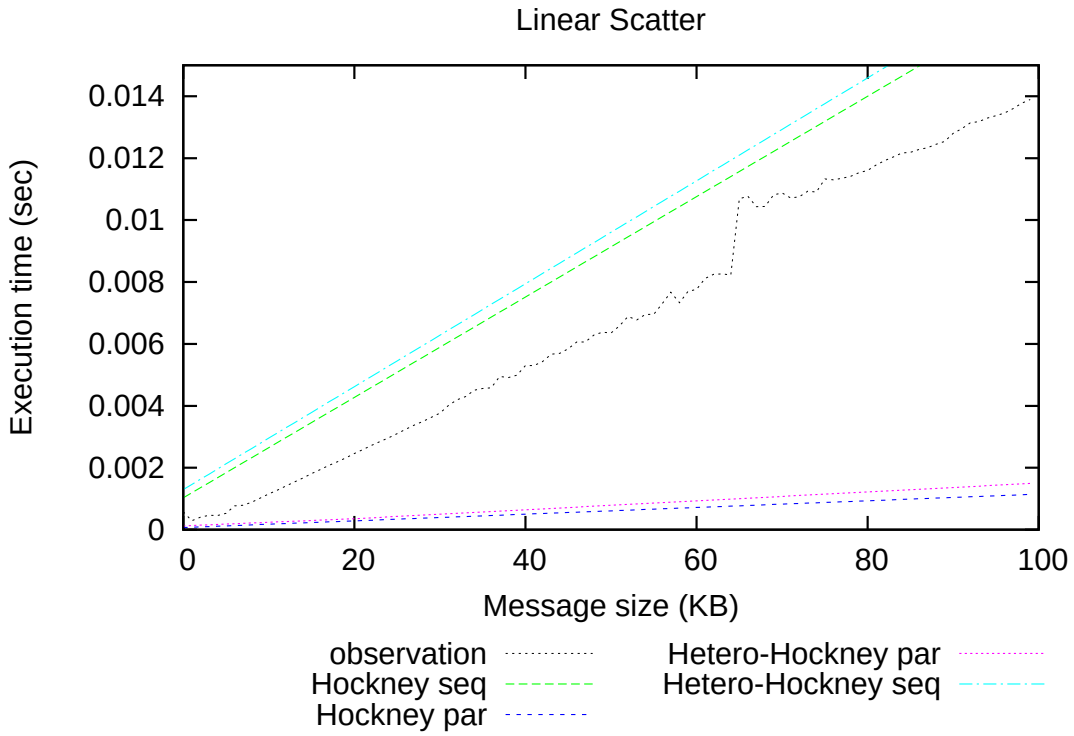


Figure 7.1: The prediction of the execution time of linear scatter on the 16-node heterogeneous cluster

In Figure 7.2, both homogeneous and heterogeneous Hockney predictions are compared with the observed execution time of the binomial scatter on the 16-node heterogeneous cluster (Appendix A). One can see that the heterogeneous Hockney model much better approximates the performance of the binomial scatter. At the same time, the example of linear scatter/gather reminds us that both heterogeneous and homogeneous Hockney models are quite restricted in their ability to accurately predict the execution time of arbitrary algorithms of collective communication operations. The main reason is that the Hockney model does not separate contributions of different nature in the execution time of a point-to-point operation, non-intuitively combining them in a small number of point-to-point parameters.

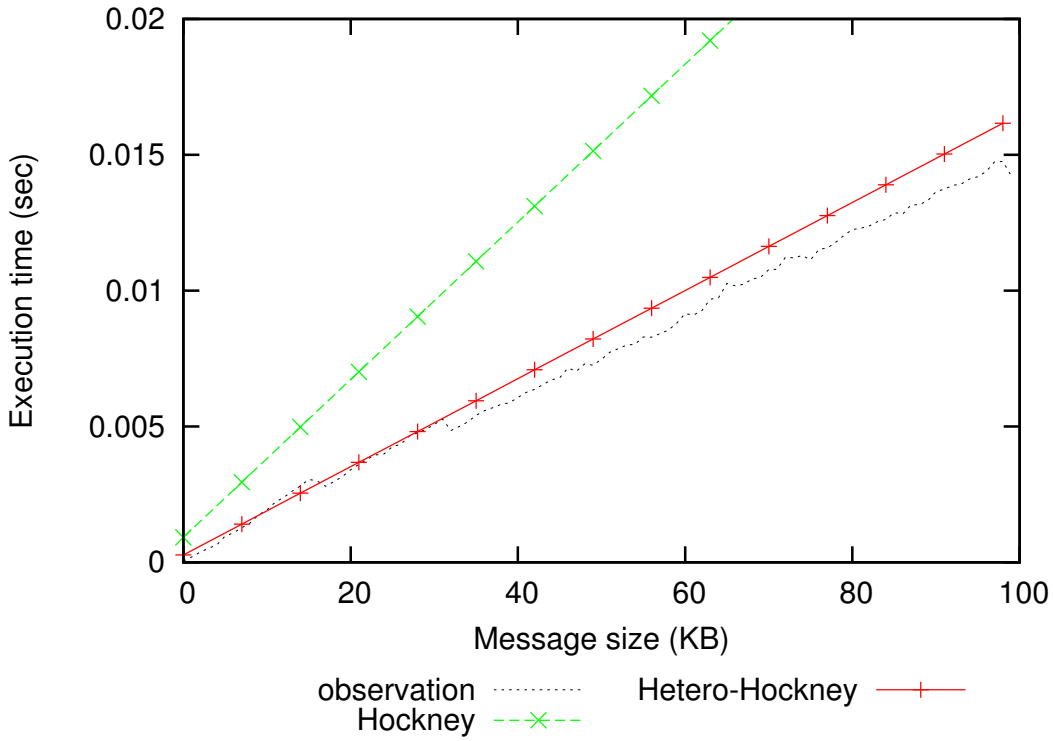


Figure 7.2: The prediction of the homogeneous and heterogeneous Hockney models vs the observation of the binomial scatter.

7.2 Optimization of Collective Operations

This section describes the implementation of two optimized versions of native *MPI_Scatter* and *MPI_Gather*, *LMO_Scatter* and *LMO_Gather* respectively. The implementation uses the *LMO* communication performance model presented in Chapter 4 in order to avoid the *MPI_Gather* time escalations and the *MPI_Scatter* leap in the execution time. More precisely, only the message size thresholds S, M_1 and M_2 are used in the implementation. These parameters are computed by the MPI programming system upon its installation on the parallel platform. In the implementation, point-to-point and low-level communications are not used, but only the native MPI counterparts.

The implementation of *LMO_Gather* re-invokes the native *MPI_Gather* for small and large messages. The gathering of medium-sized messages, $M_1 \leq M \leq M_2$, is implemented by an equivalent sequence of m x *MPI_Gather* operations with messages of the size that fits into the range of small messages: $\frac{M}{m} < M_1$ and $\frac{M}{m-1} \geq M_1$. Small-sized gatherings are synchronized by barriers, which removes

communication congestions on the receiving node. The following pseudo code implements the gather:

```
if (M1<=M<=M2) {
  find m such that M/m<M1 and M/(m-1)>=M1;
  for (i=0; i<m; i++) {
    MPI_Barrier(comm);
    MPI_Gather(sendbuf + i*M/m, M/m);
  }
} else MPI_Gather(sendbuf, M);
```

Note: If *MPI_Barrier* is removed from the code, the resulting implementation will behave exactly as the native *MPI_Gather*. It means that this synchronization is essential for preventing communication congestions on the receiving side.

The implementation of *LMO_Scatter* uses parameter S of One-to-Many communication model. For small messages, $M < S$, the native *MPI_Scatter* is reinvoked. The scattering of large messages is implemented by an equivalent sequence of *MPI_Scatter* operations with messages of the size less then S: $\frac{M}{m} < S$ and $\frac{M}{m-1} \geq S$. The pseudo code of the optimized scatter is as follows:

```
if (M>=S) {
  find m such that M/m<S and M/(m-1)>=S;
  for (i=0; i<m; i++)
    MPI_Scatter(recvbuf + i*M/m, M/m);
} else MPI_Scatter(recvbuf, M);
```

As the presented approach does not use the communication parameters reflecting the heterogeneity of the processors, it can be applied to both homogeneous and heterogeneous switch-based clusters.

To compare the performance of the optimized MPI collective operations with their native MPI counterparts, we experimented with various MPI implementations and different clusters. Here we present the results for the following two platforms:

- **LAM-Ethernet:** 11 x Xeon 2.8/3.4/3.6, 2 x P4 3.2/3.4, 1 x Celeron 2.9, 2 x AMD Opteron 1.8, Gigabit Ethernet, LAM 7.1.3,
- **OpenMPI-Myrinet:** 64 x Intel EM64T, Myrinet, Open MPI 1.2.2 over TCP.

Figure 7.3 shows the results for the heterogeneous LAM-Ethernet cluster, with all nodes in use. The message size thresholds for this platform are $M_1 =$

$5KB$, $M_2 = 64KB$, $S = 64KB$. Similar results are obtained on the 64-node homogeneous OpenMPI-Myrinet cluster (Figure 7.4). For this platform, $M_1 = 5KB$, $M_2 = 64KB$, $S = 64KB$. The results show that the optimized *LMO* versions outperform their native MPI counterparts, avoiding the escalations and restoring the linear dependency of the communication execution time on message size. On all platforms we observed $S = M_2$.

The communication execution time was measured on the root node. The barrier was used to ensure that all processes have finished the scatter-like operations. The communication experiments for each message size in a series were carried out only once. The repeated measurements gave similar results. To avoid the pipeline effect in a series of the experiments for different message sizes, the barriers were included between collective operations.

7.3 Matrix Multiplication

As a sample application we use parallel matrix multiplication, a simple but important linear algebra kernel representative for many real-life scientific applications. We use the *LMO* heterogeneous communication performance model for optimization of the MPI broadcast, scatterv and gatherv collective operations used in this application. We compare the performance of the parallel application using the optimized versions of these collective operations against the one using their native MPI implementations.

Our sample application is based on the master-slave paradigm. The master process distributes the data between the slave processes, coordinates the computations and collects the result. This type of parallel applications is often used in practice, for example, in processing of a large amount of image data collected from the hyperspectral sensors on airborne/satellite platforms (Plaza *et al.* (2006), Valencia *et al.* (2008)). Our application multiplies two dense square matrices, $C = A \times B$, and employs a simple heterogeneous parallel algorithm based on one-dimensional matrix partitioning (see, for example, Lastovetsky & Reddy (2007)). As it is shown in Figure 7.5, the matrices A and C are horizontally sliced such that the number of elements in a slice is proportional to the speed of the processor owning the slice. All the processors contain all the elements of matrix B. We assume one process per processor configuration. With this one-dimensional partitioning, our application performs: (1) irregular scatter of matrix A, (2) broadcast of matrix B, (3) parallel multiplications, and (4) irregular gather of matrix C.

We ran this application on the heterogeneous cluster (see Appendix A for specifications) in two modes. The first one uses native MPI collective operations. The second one uses our implementation of these operations optimized for the cluster. The optimization is based on the *LMO* model of the cluster, whose

parameters were estimated using the presented techniques. The results of these experiments are presented in Table 7.2. The performance of the application was improved by up to 20% as a result of the optimization of communications.

Table 7.2: Results for native collectives compared to optimized collectives

Matrix size, NxN	With native collectives, sec	With optimized collectives, sec
1000	0.80	0.65
2000	5.5	5.02
3000	16.61	11.05
4000	74.29	64.79
5000	221.00	188.72
6000	494.24	394.65

The main idea is to use non-flat communication trees and to perform their optimal mapping based on the analytical predictions of the *LMO* model. The native scatterv and gatherv are traditionally implemented using a flat tree (in all known MPI implementations). They are exposed to the escalations of the execution time similar to their regular counterparts scatter and gather. The *LMO* model is used to determine the message sizes that give the escalations and performance degradation. Then the use of the non-flat trees allowed us to eliminate these escalations. The optimal mapping of these trees allowed us to significantly reduce the execution time of these operations (up to 40% compared to their native counterparts). The native broadcast is implemented by the binomial algorithm.

7.4 Summary

In this chapter we presented the experimental results demonstrating how heterogeneous communication models can assist applications designers more accurately than traditional models. Our first example shows the heterogeneous version of the traditional Hockney model providing a means of assessing execution times of linear and binomial scatter. The Hockney model is limited in that it cannot separate different contributions of communication like the *LMO* model. It is seen to have better accuracy as a predictor of binomial scatter than linear as it can estimate the parallelism aspects of communication more effectively than the sequential communications.

The *LMO* model is unique in that it maps significant regions of non-linear behavior that result in performance degradation for a range of message sizes. The model adapts to each platform and allows the applications designer the opportunity to avoid these problems that are specific to each platform. We demonstrated a key advantage of our *LMO* model that is based on empirical parameters found for the particular platform over a purely analytical traditional model. We show the improvements achieved with this model by comparing the performance of optimized and ordinary collective operations.

Finally we show that the use of the heterogeneous communication models can significantly improve the performance of parallel matrix multiplication based on the master-slave paradigm. We use the *LMO* heterogeneous communication performance model for the MPI broadcast, scatterv and gatherv collective operations. The algorithms used non-flat trees to avoid escalations in communication time predicted by the *LMO* model. Improvements in application performance of 20% are shown using the optimized versions of collective communications, compared to their native MPI implementations.

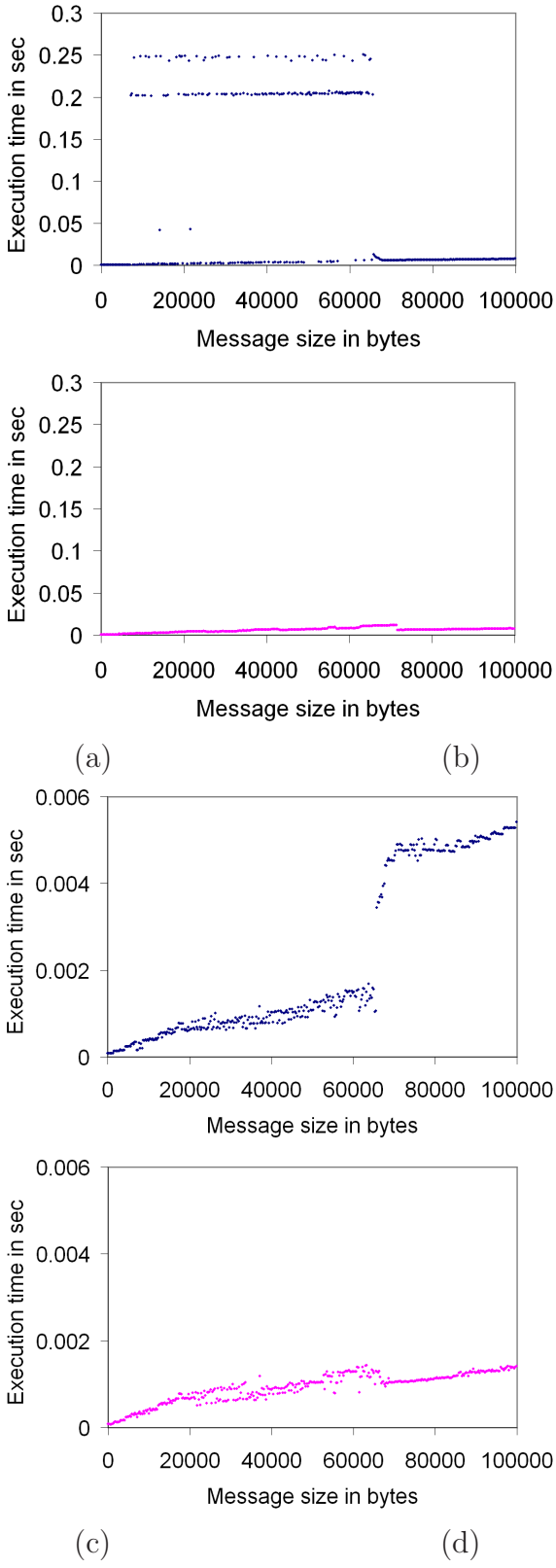


Figure 7.3: Performance of (a) MPI_Gather, (b) LMO_Gather, (c) MPI_Scatter, (d) LMO_Scatter on 16-nodes heterogeneous cluster LAM-Ethernet.

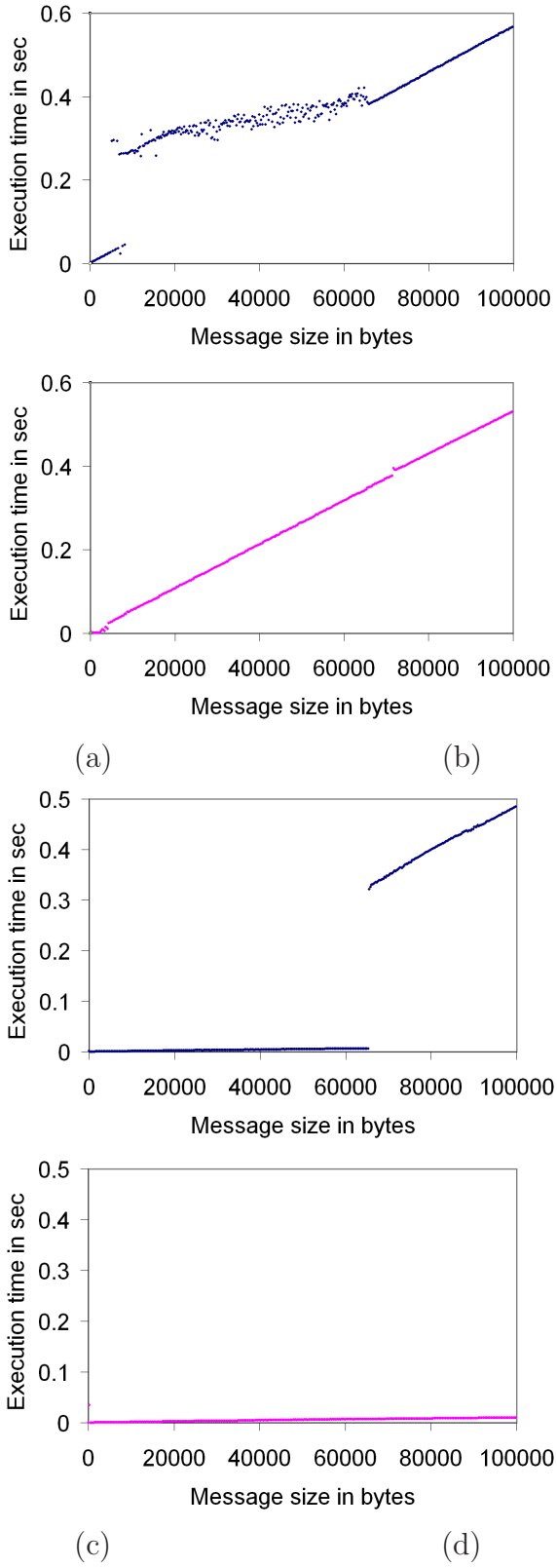


Figure 7.4: Performance of (a) MPI_Gather, (b) LMO_Gather, (c) MPI_Scatter, (d) LMO_Scatter on 64-nodes OpenMPI-Myrinet cluster.

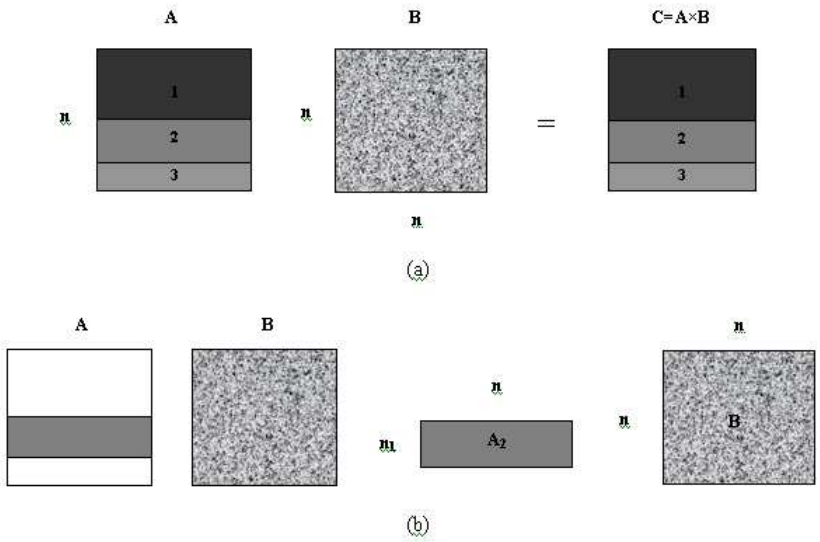


Figure 7.5: Matrix operation C=AB with matrices A, B, and C. Matrices A and C are horizontally sliced such that the number of elements in the slice is proportional to the speed of the processor. (b) Serial matrix multiplication $A_2 \times B$ of dense matrix A_2 of size $n-1 \times n$ and dense matrix B of size $n \times n$ to estimate the absolute speed of processor 2.

Chapter 8

Conclusions

Traditionally parallel computing was on a specialist platform of homogeneous processors dedicated to high-performance computing. The situation has evolved however and with new systems software the parallel applications can now take advantage of ordinary clusters of personal computers. These networks of today are heterogeneous with a variety of computers with different power and communication speeds. The simple analytical models used to predict performance and estimate parallel applications prediction are designed for homogeneous systems with a uniformity that no longer exists with heterogeneous systems. In this thesis we addressed the new issues raised for communication performance models for parallel computing on heterogeneous platforms, by extending the current traditional models, and we proposed our own *LMO* performance model as a better solution. The models were designed to further assist applications designers to tune and optimize parallel communications with the new challenges of the heterogeneous environment on switched networks.

At each step we addressed the shortcomings of current methods with our own solutions in innovative ways. We analyzed the design restrictions of traditional communication performance models that limit their ability to accurately reflect the nature of collective communications on the switched network. As a result the prediction of the execution time of collective communication operations on homogeneous and heterogeneous clusters was adversely affected. In particular, we showed that the constant and variable contributions of processors and network were not fully separated in these models. Full separation of the contributions that have different nature and arise from different sources had lead to a more intuitive and accurate models, but the additional parameters of such models could not be estimated from only the point-to-point experiments, which were usually used for traditional models. The *LMO* is a new model that overcame this problem and that was an adaptive and intuitive reflection of collective communications for MPI. We described a set of communication experiments that is sufficient for

accurate estimation of its parameters. We also presented an implementation of the new model in the form of a software tool, *CPM*, that automated the estimation of both this model and heterogeneous extensions of traditional communication performance models. It was shown by our experimental results demonstrating that the elaborated *LMO* model that it was more accurate in predicting the execution time of collective operations than traditional models.

8.1 MPIBLib: A New Benchmarking Library

Our first task was to examine current benchmarking methods, taking note of their limitations, we then designed a new MPI benchmarking suite called *MPIBlib*. This performed the accurate estimation of the execution time of MPI communication operations with the choice of a variety of timing methods. Since most MPI benchmarking suites were in the form of a standalone executable program and produce a lot of output data for further analysis, this means they could not be integrated into application-level software. We addressed this need with a benchmarking library that can be integrated easily in parallel programming systems, the *MPIBlib*, and that can be linked to other applications and used at runtime. We described the software design of the *MPIBlib* suite, with its point-to-point and collective modules and the usage of the library and the executables and experiments to demonstrate the benchmarking with graphical results. The second short-coming of current benchmarking suites was that they are limited to one timing method. We implemented a choice of timing methods that allows the user to select the most suitable to balance the needs for speed and accuracy. The benchmarking suite played the important role as the basis for building the new *LMO* performance model. It was also useful for the tuning of collective communication operations and allows for an evaluation of different available implementations.

8.2 The LMO and Heterogeneous Extended Versions of Traditional Models

The core work of the thesis is a new performance model, the *LMO* model, and the extending of existing performance models to become heterogeneous. The models are used by applications developers to estimate parallel communications operations for the tuning and optimization processes. These are greatly facilitated by a model when redesigning the application to allow for better computation and

communication cost. The current models used to optimize the applications design are designed for homogenous systems, and are therefore not fully representative of the diversity of the heterogeneous switched network and are liable to be inaccurate in their estimation. We examined this challenge in a two-fold way. We explored and extended the current models for Hockney and LogP variants, and developed our own model, the *LMO* model, as a better solution for heterogeneous switched clusters. The essential problem with current models is the they cannot map collective communications formations completely due to the limited nature and few number of their parameters. This means that the efficiency of the application becomes less determined. Our new model addressed this shortcoming, and allowed a natural and intuitive mapping to the collective operation. We did this with a greater number of parameters that separated the constant and variable aspects of communication. The greater number of parameters of our model were then calculated in an innovative and efficient way.

Any analytical communication performance model only makes sense if it can be implemented, that is, supported by a method of accurate estimation of its parameters. Thus a communication model was seen as consisting of two parts, the design of the model and the method of its estimation. The estimation part of the model is very important and can exert significant influence on the design part. For example, the estimation methods for all the traditional communication models are based on point-to-point communication experiments. The use of the traditional estimation methods obviously restricted the design of the communication models. These models could not include point-to-point parameters that could not be found from point-to-point communication experiments, even if the additional parameters would make the expression of collective communication algorithms much more intuitive and hence more accurate. In this thesis we showed that it was the traditional estimation methods that were the main factor limiting the expressive power of the traditional communication models. They did not allow them to fully separate the contributions of different nature and originating from different sources in the communication cost. Heterogeneous communication performance models had not been as intensively studied as homogeneous ones to date. We proposed an original heterogeneous model, the *LMO* model, the design of which could not be supported by the traditional estimation methods. This model separated the *constant* and *variable* contributions of the processors and network allowing for more intuitive expressions of collective algorithms than the traditional models or their straightforward extensions. A new estimation method supporting the design of this model was also proposed. The *CPM* software tool presented in this thesis implemented this new estimation method and therefore could be used for automatic estimation of non-traditional heterogeneous communication models.

Our approach to the model design was that it was adaptive to each switched

network platform, with an installation that found the characteristics particular to that network with empirical data. The *LMO* model expressed the linear response of execution times as a function increasing message sizes, with a definition of regions of nonlinear escalation regions. It allowed for the easy assessment of collective operations for MPI applications without an need for knowledge or changes to the underlying platform.

The LMO model has the following innovative features over previous approaches:

- It is possible to map our parameters in a much more direct and flexible manner for MPI collective communications in an easy and clear way. It allows for the distinctive representation of both serial and parallel parts of the communication operations on a switched network. This is an important improvement on traditional models that do not allow direct mapping of collective communications on a switched network.

- We designed the model for *heterogeneous* networks, and is more accurate and efficient than other previous models that are all for homogeneous systems. The model design required that we have more parameters than could be able to be found by point-to-point methods as traditionally used. Instead we included parameter estimation techniques based on the model for three nodes, in this innovative way we could find the extra parameters we need.

- Our empirical studies revealed a particular range of mid-size messages show a marked non-linear response with significant performance degradation. This is a critical finding from the point of view of applications designers, and has been largely undocumented by other works. The new model is defined for a full range of message sizes, and includes previously ignored ranges of sizes where *significant escalations* are found. No other model to date charts these critical areas of greatly elevated execution times, they assume linearity throughout.

- The model is more accurate because it is based on an empirical approach that is adaptive and specific for a particular heterogeneous network platform. The characteristics of the particular network are established fully automatically at installation, the application designer does not need to know or adjust any underlying platform specifications.

- The model is fully automated with parameter estimation at runtime with the *CPM* software tool that links to the use *MPIBLib* library software.

The model was implemented with the software tool that also implemented a choice of heterogeneously extended traditional models. The software linked

to the new *MPIBlib* library, and allowed us to compare the effectiveness of the heterogeneous models and the advantages of the new *LMO* model that was shown to be more accurate and easier to use.

We explored ways to extend a heterogeneous version of traditional models. Their assumption of homogeneity and their limited nature of their parameters prevented them from accurate estimation of the execution time of collective communication operations on computational clusters with a single switch. These communication performance models for high performance computing were analytical, with the basis of these models was a point-to-point communication model characterized by a set of integral parameters, having the same value for each pair of processors. Their parameters were based on point-to-point that are found statistically from the measurements of the execution time of communications between any two processors. The execution time of collective operations was expressed as a combination of the point-to-point parameters and predicted for different message sizes and numbers of processors. The weakness of this approach was that processors could differ in a heterogeneous system. They were also purely analytical and therefore could not reflect any escalations or nonlinear behavior for some message sizes that were found empirically.

The traditional models were applied to heterogeneous clusters in two ways. The first method averaged values obtained for every pair of processors, the heterogeneous cluster would be treated as homogeneous. This approach had a small number of parameters and the execution time was represented by a simple compact formula. Instead we took a second approach by deriving heterogeneous extensions of traditional models. By taking pairs of heterogeneous processors for measurement we got different parameter values. The first approach was shown to be less accurate than the heterogeneous approach however from our experiments on a 16 node single switched network. Our LMO model was then shown to be more accurate than the heterogeneous extension models. This is because it allowed for an intuitive representation of both serial and parallel parts of the communication operations on a switched network. It was also more accurate because it was based on an empirical approach that was adaptive and specific for a particular heterogeneous network platform.

8.3 Models and Communication Optimization

Our final work is with the use of the models for optimization, and we presented the software tool, the *CPM*. This allows for rapid implementation of traditional and advanced heterogeneous communication performance models. Our *CPM* software tool allows for the selection of the particular model. In the case of the LMO

model, we demonstrated its use with a model-based modification of the linear algorithm. Our research had found that significant non-deterministic escalations of the execution time for medium-sized messages within some range were observed for different platforms and MPI implementations. The LMO communication performance model was the only model that could address this issue and had two parameters which bounded this range. The algorithm used these parameters to split the medium-sized messages into a series of linear gathers with smaller messages. In this way the performance degradation from the non-deterministic escalations of the execution time was avoided. The software allowed a choice to switch between algorithms, or the selection of binomial tree algorithms. The binomial algorithm could be used for nonlinear response regions of message sizes found with the linear algorithm. In these ways we have demonstrated how the new LMO model and the software framework could be used to optimize algorithms for collective communications.

8.4 Future Work

The design of our LMO model and software is highly extensible and suitable for further applications and designs. Parameters that allow the separation of contributions from network and processors and the constant and variable parts of the communication distinguishes our model. This is a flexible design that may be adapted and extended in a variety of ways with its possibilities for intuitive mapping of collective operations.

Some areas of further work are as follows:

- The LMO model may be continued to map other collective communication operations such as All-to-All, etc.
- The model could be redesigned to fit multi-core and multiprocessor architectures due to its flexible design. The separation of the constant and variable costs of communication and the processor costs from network costs the model facilitates its adaptability to these more complex systems. A group of processors would act in a similar way to a single cluster.
- Adapting the model for different cluster topologies, (multiple switches etc). Further extensions of design for mapping WANs, global and grid networks would require an adaptive approach to the threshold parameters, to allow for the different regions of nonlinear behavior.
- The software for the *CPM* library is easily extended to include new operations, models and algorithms due to its intrinsic design. The software for

optimization may be adapted to allow for a selection of different models within the same algorithm implementation. The work of Pješivac-Grbović *et al.* (2005) showed how models may be used together, switching between them for message sizes. Our software could facilitate this design process with some additions to the algorithm optimization process.

- Other areas of further work are to explore the models applicability to other languages such as PVM. The high level nature of the model design indicates some promise in this area, as it is independent of platform specifications and underlying operating systems software.

The issue of new network topologies also suggests the issue of ***scalability***. The model has yet to be tested on very large networks. The model design is highly adaptable and lends itself to new reflections of further previously uncharted regions of non-linear behavior with its additional empirical parameters. The networks of the future will benefit from ever increasing sizes, the model may have potential to be adapted to the new challenges of much larger scale computing.

All types of parallel systems to be estimated have similar performance requirements for a model that is simple, adaptable and an accurate reflection of communications performance. The LMO has a greater number of parameters for heterogeneity and may offer potential for mapping for other systems than MPI, as the parameter estimation techniques are based on parallelism at a high level. It is hoped that the use of these models will facilitate the astounding new abilities of heterogeneous parallel computing systems in the future.

Appendix A

The 16 node heterogeneous cluster.

A.1 Switches

The network hardware consists of two Cisco $24 + 4$ port Gigabit switches. Each node has two Gigabit ethernet ports - each eth0 is connected to the first switch, and each eth1 is connected to the second switch. The switches are also connected to each other. The bandwidth of each port can be configured to meet any value between 8Kb/s and 1Gb/s. This allows testing on a very large number of network topologies. As the bandwidth on the link connecting the two switches can also be configured, the cluster can actually act as two separate clusters connected via one link.

A.2 Processors

The 16 node heterogeneous cluster.

Node Type	Model	Linux	Processor	Bus (MHz)	L2 cache(MB)	#
1	Dell Poweredge SC1425	2.6	3.6 Xeon	800	2	2
2	Dell Poweredge 750	2.6	3.4 Xeon	800	1	6
3	IBM E-server 326	2.4	1.8 AMD Opteron	1000	1	2
4	IBM X-Series 306	2.4	3.2 P4	800	1	1
5	HP Proliant DL 320 G3	2.6	3.4 P4	800	1	1
6	HP Proliant DL 320 G3	2.6	2.9 Celeron	533	0.256	1
7	HP Proliant DL 140 G2	2.4	3.4 Xeon	800	1	3

References

(2007). *Intel MPI Benchmarks. User Guide and Methodology Description*. Version 3.0 edition, Intel, http://software.intel.com/en-us/articles/intel-mpi-benchmarks/. 12, 14, 16

Alexandrov, A., Ionescu, M.F., Schauser, K.E. & Scheiman, C. (1995). LogGP: Incorporating long messages into the LogP model — one step closer towards a realistic model for parallel computation. Tech. rep., Santa Barbara, CA, USA. 23, 30

Bai, J. & Perron, P. (1998). Computation and analysis of multiple structural-change models, http://ideas.repec.org/p/mtl/montde/9807.html. Cahiers de recherche 9807, Universite de Montreal, Departement de sciences economiques. 66

Bernaschi, M. & Iannello, G. (1998). Collective communication operations: experimental results vs. theory. In: *Concurrency - Practice and Experience*. vol. 10, pp. 359–386. 14

Bhat, P.B., Raghavendra, C.S. & Prasanna, V.K. (2003). Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, **Vol. 63**, pp. 251–263. 30

Chan, E.W., Heimlich, M.F., Purkayastha, A. & van de Geijn, R.A. (2004). On optimizing collective communication. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pp. 145–155, IEEE Computer Society, Washington, DC, USA. 30, 74

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R. & von Eicken, T. (1993). LogP: towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, **Vol. 28**, pp. 1–12. 22, 30

Culler, D., Liu, L.T., Martin, R.P. & Yoshikawa, C. (1996). LogP Performance Assessment of Fast Network Interfaces. *IEEE MICRO, February 1996*, **Vol. 16**, pp. 35–43. 23

Forum, M.P.I. (2008). MPI : A Message-Passing Interface Standard, http://www.mcs.anl.gov/research/projects/mpi/. Tech. rep., MPI Forum. 2

Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L. & Woodall, T.S. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pp. 97–104, Budapest, Hungary. 2

Galassi, M., Davies, J., Theiler, J., Gough, G., B.and Jungman, Alken, P., Booth, M. & Rossi, F. (2009). *GNU Scientific Library (2009), http://www.gnu.org/software/gsl/*. Bristol: Network Theory Limited. 39, 66, 87

Gropp, W. & Lusk, E.L. (1999). Reproducible measurements of MPI performance characteristics. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 11–18, Springer-Verlag, London, UK. 14, 15

Gropp, W., Lusk, E., Doss, N. & Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, **Vol. 22**, pp. 789–828. 2

Grove, D. & Coddington, P. (2001). Precise MPI performance measurement using MPIBench. In *In Proceedings of HPC Asia, 2001 : Gold Coast, Australia*. 12, 13, 14, 19

Hatta, J.i. & Shibusawa, S. (2000). Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In *ICPP '00: Proceedings of the 2000 International Workshop on Parallel Processing*, pp. 173, IEEE Computer Society, Washington, DC, USA. 30

Hockney, R.W. (1994). The communication challenge for MPP: Intel paragon and meiko CS-2. *Parallel Computing*, **Vol. 20**, pp. 389–398. 22, 30, 73, 91

Kielmann, T., Hofman, R.F., Bal, H.E., Plaat, A. & Bhoedjang, R. (1999). Magpie: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '99)*, pp. 131–140, Atlanta, Georgia. 33

Kielmann, T., Bal, H. & Verstoep, K. (2000). Fast measurement of LogP parameters for message passing platforms. *IPDPS '00: Proceedings of the 15th IPDPS 2000 Workshops on Parallel and Distributed Processing*. In *Lecture Notes in Computer Science, Springer-Verlag, London, UK*, pp. 1176–1183. viii, 22, 24, 25, 26, 30, 76, 77, 79, 88

Lastovetsky, A. & OFlynn, M. (2007). A Performance Model of Many-to-One Collective Communications for Parallel Computing. *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007)-Workshop on Parallel and Distributed Scientific and Engineering Computing, PDSEC*, IEEE Computer Society, Long Beach, California, USA. 2, 8, 56, 76

Lastovetsky, A. & Reddy, R. (2006). HeteroMPI: Towards a message-passing library for heterogeneous networks of computers. *Journal of Parallel and Distributed Computing*, **Vol. 66**, pp. 197–220. 31

Lastovetsky, A. & Reddy, R. (2007). Data distribution for dense factorization on computers with memory heterogeneity. *Parallel Computing*, **Vol. 33, December 2007**, pp. 757–779. 96

Lastovetsky, A. & Rychkov, V. (2007). Building the communication performance model of heterogeneous clusters based on a switched network, *Proceedings of the 2007 IEEE International Conference on Cluster Computing (Cluster 2007)*, September 17-20. pp. 568–575, IEEE Computer Society, Austin, Texas, USA. 65, 68

Lastovetsky, A., Mkwawa, I. & OFlynn, M. (2006a). An Accurate Communication Model of a Heterogeneous Cluster Based on a Switch-Enabled Ethernet Network, *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS 2006), 12-15 July 2006*. vol. 2, pp. 15–20, IEEE Computer Society Press, Minneapolis, Minnesota, USA. 2, 8, 56, 76

Lastovetsky, A., Mkwawa, I. & OFlynn, M. (2006b). Modeling Performance of Many-to-One Collective Communication Operations in Heterogeneous Clusters, *HCL Laboratory, School of Computer Science and Informatics, UCD, Dublin, Ireland*. Tech. rep. 2

Lastovetsky, A., O'Flynn, M. & Rychkov, V. (2007). Optimization of Collective Communications in HeteroMPI, *14th European PVM/MPI User's Group Meeting, Sept 30 - Oct 3 2007, Paris, France*, Lecture Notes in Computer Science, Ed.s Franck Capello, Thomas Herault, Jack Dongarra. vol. 4757, pp. 135–143, Springer-Verlag Berlin Heidelberg. 2, 9, 54

Lastovetsky, A., Rychkov, V. & OFlynn, M. (2008a). MPIBlib: Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters. In A. Lastovetsky, T. Kechadi & J. Dongarra, eds., *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 15th European PVM/MPI User's Group Meeting, September 7-10th, Dublin, Ireland, Lecture Notes on Computer Science*, vol. 5205, pp. 227–238, Springer-Verlag Berlin Heidelberg. 2, 6, 8, 35, 64, 77, 80, 85

Lastovetsky, A., Rychkov, V. & O'Flynn, M. (2008b). A Software Tool for Accurate Estimation of Parameters of Heterogeneous Communication Models. In T.D.J. Lastovetsky Alexey; Kechadi, ed., *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, September 7-10th, Dublin, Ireland, Lecture Notes in Computer Science*, vol. 5205, pp. 43–54, Springer-Verlag Berlin Heidelberg. 2, 7, 8, 80, 84

Lastovetsky, A., Rychkov, V. & O'Flynn, M. (2009). Revisiting communication performance models for computational clusters. In *IPDPS '09: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09), May 25-29, 2009, Rome, Italy*. 2, 5, 7, 8, 9, 56, 72, 77, 80, 84

Nagle, D. (2005). MPI – The Complete Reference, Vol. 1, the MPI core, 2nd ed., Scientific and Engineering Computation Series, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra. *Scientific Programming, January 2005*, **Vol. 13**, pp. 57–63. 2

Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G.E., Gabriel, E. & Dongarra, J.J. (2005). Performance analysis of MPI collective operations. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15*, 272.1, IEEE Computer Society, Washington, DC, USA. 32, 76, 108

Pješivac-Grbović, J., Bosilca, G., Fagg, G.E., Angskun, T. & Dongarra, J.J. (2007a). MPI Collective Algorithm Selection and Quadtree Encoding. *Parallel Computing*, **Vol. 33**, pp. 613–623. 32

Pješivac-Grbović, J., Bosilca, G., Fagg, G.E., Angskun, T. & Dongarra, J. (2007b). Decision Trees and MPI Collective Algorithm Selection Problem. In *Euro-Par 2007 Parallel Processing, Lecture Notes in Computer Science*, vol. Volume 4641/2007, pp. 107–117, Springer Berlin / Heidelberg, Germany. 32

Plaza, A., Valencia, D., Plaza, J. & Martinez, P. (2006). Commodity cluster-based parallel processing of hyperspectral imagery. *Journal of Parallel and Distributed Computing*, **Vol. 66**, 345–358. 96

Supinski, B.d. & Karonis, N. (1999). Accurately measuring MPI broadcasts in a computational grid. In *HPDC'99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, pp. 29–37, IEEE Computer Society, Washington, DC, USA. 13

Thakur, R., Rabenseifner, R. & Gropp, W. (2005). Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, **Vol. 19**, pp. 49–66. 30, 31

Turner, D., Oline, C.X., A. & Benjegerdes, T. (2003). Integrating new capabilities into NetPIPE. In *EuroPVM/MPI 2003. LNCS, Dongarra, J., Laforenza, D., Orlando, S. (eds.)*, pp. 37–44, Springer, London, UK. 14, 15

Vadhiyar, S.S., Fagg, G.E. & Dongarra, J. (2000). Automatically tuned collective communications. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 3, IEEE Computer Society, Washington, DC, USA. 31

Valencia, D., Lastovetsky, A., O'Flynn, M., Plaza, A. & Plaza, J. (2008). Parallel Processing of Remotely Sensed Hyperspectral Images On Heterogeneous Networks of Workstations Using HeteroMPI. vol. 22, pp. 386–407. 96

Williams, T. & Kelley, C. (2007). Gnuplot: an interactive plotting program, http://www.gnu-plot.info/docs/gnuplot.pdf. 36, 84

Worsch, T., Reussner, R. & Augustin, W. (2002). On benchmarking collective MPI operations. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 271–279, Springer-Verlag, London, UK. 12, 13, 14, 19

Zeileis, A., Leisch, F., Hornik, K. & Kleiber, C. (2002). strucchange: An R Package for Testing for Structural Change in Linear Regression Models. *Journal of Statistical Software*, **Vol. 7**, pp. 1–38. 88