Topology-aware Optimization of Communication Cost of Parallel Applications in Heterogeneous HPC Systems

Tania Malik
Student No: 11211133

This thesis is submitted to University College Dublin

in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

School of Computer Science

Head of School: Pádraig Cunningham
Research Supervisor: Alexey Lastovetsky

September 2016

Acknowledgements

I consider myself very lucky for having Dr Alexey Lastovetsky as my thesis supervisor. The immense guidance and encouragement I received from him over the years is truly commendable. I would also like to extend my special gratitude to Dr Vladimir Rychkov for his invaluable help during the initial stages of my research. My thanks are due to the Irish Research Council and IBM for considering me for their prestigious Industry Partnership Scholarship Award. This research award enabled me to focus on my research without worrying about the fees, living expenses etc. In addition, the programme also offered various research networking opportunities for which I remain grateful to IRC, IBM, and Alexey. Most of the experiments reported in my thesis were carried out using the Grid'5000 that was developed under the INRIA ALADDIN funded by CNRS, RENATER, and other agencies. I would like to acknowledge this facility and would like to extend my thanks to the providers.

Looking back, I have no doubt in saying that the last four years of my life have been truly rewarding for me from various perspectives. Becoming a part of the highly prestigious University College Dublin provided me an excellent environment to hone, sharpen, and develop my personality and also advance in my area of research. The Heterogeneous Computing Laboratory at UCD is one of the leading research groups in the world in the domain of parallel and heterogeneous computing and attracts talented researchers from all parts of the world. Being a part of this group enabled me to learn and grow with some of the smartest people of the world. My thanks are extended to all HCL members: Ravi, Jean Noel, Amani, Ashley, David, Jun, Ken, Khalid, Kiril, and Ziming. Here I would also like to thank some of the other amazing people I met during my PhD journey with whom I shared several lunches and enjoyed various brainstorming sessions: Saima, Musfira, Mona, Malika, and Nabila. The collaborations developed and the friendships forged with them will be cherished for years to come. I would also like to extend my gratitude to Ms Hilda Fitzpatrick for her amazing support and a wonderful home (away from home!) she provided me during the first two years of my stay in Ireland.

It was during my PhD that God blessed me with my wonderful life partner Dr. Faisal Zahoor. Having been through the trying and yet rewarding journey of PhD himself, his insight, suggestions, and support during the various ups and downs of my PhD studies were always meaningful and are truly appreciated. Also, it was during my PhD that the sweetest possible thing happened to me; God blessed me with a little angel, Haadia. Although, at times it was difficult to balance between family and research, but the constant support of my husband, encouragement of my supervisor, and the sweet, smiling face of my daughter, gave me the strength to finish the tasks at hand.

We are but an image of our parents. I am eternally grateful to my loving parents Malik Zafat Iqbal and Ghazala Yasmeen for bringing me into this world and providing me with everything possible at their disposal. I am also thankful to my mother in law Khursheed begam for supporting me during my PHD journey. I am also grateful to my loving siblings: Fareesa, Joveria, Wasmiya, Wahaj. Last but not the least, I remain grateful to God who created me, provided me with everything and more, and made everything easy for me. May He, the Almighty accept this humble undertaking of mine.

Dedicated to my Soulmate ... Faisal

Abstract

Modern High Performance Computing (HPC) platforms are comprised of highly heterogeneous computing devices connected by complex hierarchical networks. To execute data-parallel scientific applications efficiently on such platforms, it is necessary to balance the load of processors and to minimize the cost of communications. The former can be achieved by partitioning data between the processors in proportion to their speed. The latter can be achieved by reducing the volume of communications, by optimal mapping of the data to the processors, and by optimal scheduling of communications. In this doctoral research, we aim to optimize the communication performance of parallel applications, assuming that the data have been optimally partitioned between the processors so that the total volume of communicated data has been minimized.

Communications on hierarchical heterogeneous HPC platforms can be optimized based on topology and performance information. For MPI, as a major programming tool for such platforms, a number of topology and performance-aware implementations of collective operations have been proposed for optimal scheduling of messages. These approaches improve the performance of application and do not require modifications to the application source code. However, they are applicable to collective operations only and do not affect the parts of the application that are based on

point-to-point exchanges. This research work addresses the problem of efficient execution of data-parallel applications on interconnected clusters and present optimization approaches that reduce communication cost by taking into account the entire communication flow of the application and underlying network topology.

In this thesis, we have proposed and implemented the approximate topology-aware heuristic algorithms aimed at minimization of the communication cost of data parallel applications on heterogeneous hierarchical networks. We tested these algorithms in the context of the parallel matrix multiplication application, which is a very important computation kernel and a building block of many scientific applications. In addition, tests were also performed on real-life CFD application, MPDATA, which is one of the major parts of the EULAG geophysical model. We also demonstrate the correctness and efficiency of the proposed approaches by experimental results on multi-core nodes and interconnected heterogeneous/homogeneous clusters.

Contents

List of Figures

List of Tables

Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

Chapter 1

Introduction

1.1 Motivation

Modern HPC platforms are becoming increasingly complex, heterogeneous and hierarchical. In terms of computing, “heterogeneous” refers to non-uniformity in some aspects of the system. Heterogeneity appears not only in the computing devices but also in networks and can arise from hardware heterogeneity (Central Processing Units (CPUs), Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs) etc.), software heterogeneity (operating system, compilers, libraries, etc.) and complex network topology [1]. These heterogeneous platforms present significant challenges to computer scientists [2], [3]. The HPC applications must adapt to the heterogeneity of these HPC platforms for optimal execution [4].

The widespread deployment of multi-core and many-core architectures has raised the need to exploit parallel computing techniques. The number of cores have recently increased by an order of magnitude. Modern systems are now massively built with multi-core co-processors. In 1995, the number of cores in the top 10 supercomputers were ranged from 42 to 3680 [5]. Nowadays, this number ranges from 115,984 to 3,120,000. Such a substantial increase in scale significantly increases the data movement and subsequently increases the coordination and interaction cost of processes in data-parallel applications.

In heterogeneous systems, data movement is the primary factor that influences power consumption, execution speed, and scalability [6]. Here optimal placement and distribution of data throughout the system are extremely important. While dealing with heterogeneous platform two major challenges arise. First, how to partition data across the heterogeneous processors to efficiently balance the load, so that all processors achieve near-optimal load balance? [7], [8], [9]. And second, how to minimize the communication overhead to improve the overall performance? Minimizing the communication overhead is perhaps the single most important optimization problem that needs to be solved while working on a heterogeneous system [10], [11]. In this doctoral work, we focus on the optimization of communication performance of parallel applications on heterogeneous platforms, assuming that the data have been optimally partitioned between the processors.

1.2 Communication Optimization for Parallel Applications

Communication between the processes of parallel applications executed on heterogeneous platforms involve multiple message hops, non-optimal routes and traffic congestion, which significantly affect performance. Communication optimization is a broad field that comprises a number of different approaches. The goal of all such optimization approaches is to reduce the overall runtime of the communication operations. Communication optimization on heterogeneous HPC platform is comprehensively covered in [10], where all the existing approaches were classified as performance or topology-aware. The increasing complexity of HPC platforms has made topology awareness a critical component of HPC application optimization. A number of topology-aware approaches have been proposed in [12], [13]. The main idea behind the topology-aware optimizations is to reduce communication traffic and contention by considering network topology so that most of the communication occurs between nearby processors. Whereas, in performance-aware optimizations, network properties are reconstructed with performance measurements by using communication benchmarks. This approach is used in the absence of topology information.

Many existing Message Passing Interface (MPI) applications can be executed efficiently on hierarchical heterogeneous HPC platforms by using topology-aware collectives and do not require modifications in the application source code. However, it is applicable to collective operation and does not affect the applications that are based on point-to-point exchanges. In this case, the communication cost can be reduced by placing frequently communicating tasks on physically close processors. This closeness is application-specific and depends on the logical communication flow of the application.

Topology information has been used in developing a number of topology-aware implementations of MPI collectives for optimal scheduling of messages on heterogeneous HPC platforms [14], [15], [16], [17]. However, parallel applications based on point-to-point exchanges need another solution, which could consider the application communication flow. If all point-to-point communications are performed over a virtual topology of processes, they can be optimally mapped onto physical topology of processors, which minimizes the communication cost of the application [18], [12], [19].

1.3 Thesis Scope

Existing approaches for point-to-point communication optimization do not incorporate the heterogeneity of processors and networks. To optimize communications in scientific data-parallel MPI applications, we have to consider both topology information and application communication flow. Performance of data-parallel applications, especially those designed for heterogeneous platforms, highly depends on balanced workload, which is achieved by partitioning the data in proportion to the speed of processors. In turn, heterogeneous data partitioning affects the application communication flow and need to be considered in topology-aware optimization. Assuming balanced workload among the processors, the main contribution of this work is to propose and implement the topology-aware approximate heuristic algorithms aimed at the minimization of the communication cost of data parallel applications on heterogeneous hierarchical platforms.

In this thesis, we target data intensive parallel scientific applications, such as dense linear algebra and Computational Fluid Dynamics (CFD). We test the proposed algorithms in the context of parallel matrix multiplication, which is an important computational kernel and a building block of many scientific applications, and also for a real-life CFD application, MPDATA, which is one of the major parts of the EULAG geophysical model.

In our research, we target dedicated heterogeneous HPC platforms with two-level network hierarchy, such as interconnected computer nodes and clusters. The processors of these platforms are connected by a network that can be represented as a two-level rooted tree with faster communications within sub-trees and slower communications between. These networks are quite common in today's computing world. Popular examples include grid and cloud infrastructures. Even supercomputers with thousands of nodes are also examples of heterogeneous networks where the communication cost is different on different hierarchical levels - e.g. intra-node vs inter-node communication. Topology information can be taken into account, when application data is mapped to the processors, in order to minimize message hops and maximize data throughput.

1.4 Contributions

The major contributions of this thesis are:

1. Demonstrating that the problem of communication optimization has not been comprehensively addressed in literature for huge proportion of parallel applications that are designed using point-to-point communication as compared to applications based on collectives. Hence, there is a need of designing communication optimization mechanisms for this class of applications.

2. Comprehensive study and analysis of the role of application communication pattern in communication optimization. We employ parallel matrix multiplication as a case study for analysing the communication pattern and based on it we clearly motivate the need for topology-aware process mapping to reduce the communication overheads.

3. Design and implementation of two cost measurement functions to measure the communication cost of any 2D arrangement.

   - Hop-count cost function
   - Bandwidth cost function

4. Design and implementation of two topology-aware heuristic algorithms based on evaluation of the matrix multiplication application communication pattern:

   - Hop-count based algorithm
   - Bandwidth based algorithm

5. On-line simulation of the matrix multiplication application with the Simulated Message Passing Interface (SMPI) simulator of the SimGrid framework for prediction of the performance of MPI applications for complex heterogeneous platforms.

6. Study and analysis of the communication pattern of a real life CFD application, MPDATA.

7. Design and implementation of a topology-aware heuristic algorithm based on the evaluation of the MPDATA application communication pattern.

8. Demonstration of the accuracy and efficiency of the proposed solutions using experiments on two-level hierarchical networks, namely, interconnected nodes (intra- and inter-node communication levels) and interconnected clusters (intra- and inter-cluster communication levels).

In the following sub-sections, we elaborate the aforementioned contributions.

1.4.1 Need for Topology-Aware Communication Mechanism for Applications Based on Point-to-point Communications

Many high performance computing applications are designed using MPI point-to-point communications to transfer large amount of data between various processes. On a heterogeneous platform, applications written using collective operations result in poor performance and become expensive, which drives software developers to modify the applications that are based on collectives originally to point-to-point communications for efficient adaptability on the heterogeneous platform. Nearest neighbour applications and multi-dimensional stencil-based applications are the most popular types which are based on point-to-point communication. Matrix multiplication, as an example, performs well with point-to-point communication as compared to broadcast on heterogeneous platforms with 100‘s of processors.

For these applications, it is an open challenge whether it is possible to achieve performance improvement by providing topology-aware mapping. Because placement of logical MPI ranks on heterogeneous HPC system also has a significant impact on performance, thus, certain mappings will be more advantageous than others. On current HPC systems, when a scheduler supplies a list of available nodes, MPI processes are started on these nodes and assigned by logical ranks. This rank assignment is the crucial part that affects the overall performance of the application. All communication steps of the application are based on these ranks. A poor ranking may result in poor locality of communication. Generally, the default mapping scheme is the allocation of processes in blocks. This implies that all sequential ranks are allocated to the same node. This block mapping improves the performance of application over random task mapping. However, in certain situations, for example, if the application is running on a platform that has hierarchical heterogeneous clusters of heterogeneous hybrid nodes and are composed of dozens of sites, further improvement can be achieved if we consider the application communication pattern and network topology. Thus, profiling the application to identify its communication pattern and performing a rank re-ordering based on the application communication pattern can help improve the overall application performance and scalability.

1.4.2 Study and Analysis of the Communication Pattern of Matrix Multiplication

Matrix multiplication is the core component of many scientific computations. Being computationally intensive, the last decade has seen a great interest of the scientific community to develop parallel formulation of matrix multiplication on various parallel architectures [20]–[23]. Several parallel formulation of matrix multiplication are already available for homogeneous platforms [22], [24], [25] as well as for heterogeneous platforms [7], [9], [26]. At each step of matrix multiplication, communication takes place between the processors, which make it a most suitable candidate application for finding the communication optimal solution. In this thesis, we use the matrix multiplication application as a case study to propose a topology-aware communication optimized solution for the matrix multiplication kernel for heterogeneous platforms. Furthermore, if a solution can be applied successfully to the parallel matrix multiplication, it can be scaled to other tightly coupled parallel applications. Therefore we have wider applicability of the solution in mind.

We have considered parallel matrix multiplication application for heterogeneous platforms which is based on the Scalable Universal Matrix Multiplication Algorithm (SUMMA) [27]. SUMMA is designed for homogeneous multiprocessors and implemented using MPI. It distributes the workload evenly between the processors, mapping dense matrices onto a 2D grid of processors. Each processor receives one rectangle of matrices and participates in two MPI communicators that combine all processors in the same row and column. The communication flow consists of multiple broadcasts of matrix elements over these communicators. If SUMMA is executed on a hierarchical network of processors, its performance may be lower than expected due to higher communication cost. When network topology is known, this problem can be solved by using topology-aware broadcasts.

SUMMA was adapted for heterogeneous platforms, with matrices being partitioned into irregular 2D rectangles in proportion to the speed of processor [7], [9]. The rectangles, and hence the processors, are arranged in columns. In columns, the processors communicate the same way as in the original algorithm. In the horizontal direction, the partition, and hence the communication flow, is irregular. Usually, irregular communications between processors are implemented via point-to-point operations. Non-blocking point-to-point operations additionally allow for overlapping communications and computations, which can significantly improve the performance of heterogeneous algorithms. For parallel applications based on point-to-point exchanges, like heterogeneous SUMMA, no solution has been proposed yet which could use topology information to minimize communication cost.

1.4.3 Cost Measurement Function

Using the observations from the communication pattern, we propose two cost functions for matrix multiplication with the ring communication flow and two-level network hierarchy. One function estimates the number and volume of inter-cluster communications incurred by a partition arrangement of matrix rectangles. Another function estimates the communication time by using the bandwidth properties of the individual links. These cost functions are described in Chapter 3.

1.4.4 Heuristic Algorithms for Matrix Multiplication

After analysing the communication pattern of matrix multiplication on heterogeneous platforms, we propose to rearrange the rectangles of the matrix partition in order to minimize communications between different levels of the network hierarchy. Finding the optimal arrangement is an NP-complete combinatorial optimization problem; therefore, it can be solved by using heuristics. We propose two heuristic algorithms based on evaluation of the application communication flow on the given network topology. To evaluate the communication cost we use the number of message hops between clusters and the bandwidth information. These algorithms are presented in Chapter 3.

1.4.5 Simulation of the Matrix Multiplication Application

Simulation is a popular approach for predicting the performance of MPI applications for simulated complex platforms. We use the latest SMPI module of the SimGrid. SimGrid is a simulator that is developed to study the behaviour of large-scale distributed systems, such as Grids, Clouds, HPC or P2P systems. It provides ready to use models and API to simulate different distributed systems [28], [29]. We perform on-line simulation of a matrix multiplication application by creating a complex Grid-like heterogeneous platform with SMPI. Our experiments show that due to complexity and various design constraints, SimGrid cannot measure the realistic communication cost on highly heterogeneous complex platforms with application having asynchronous point-to-point communication operations. SimGrid-SMPI simulation experiment details are given in Appendix A where we present our efforts for running the experiments on a simulated platform and discuss the challenges that we have faced and most important the factors that influenced the realistic measurement of the application execution time on SMPI.

1.4.6 Study and Analysis of the Communication Pattern of MPDATA

The real life CFD application, which we considered to study and analyse the communication pattern of, is the Multidimensional Positive Definite Advection Transport Algorithm (MPDATA), that is one of the major parts of the dynamic core of the EULAG geophysical model [30], [31]. This geophysical model can be used for simulation of thermo-fluid flows across a wide range of scales and physical scenarios, including the numerical weather prediction. The

MPDATA belongs to the group of non-oscillatory forward-in-time algorithms and performs a sequence of stencil computations. The very original version of MPDATA was implemented in FORTRAN 77 and parallelized using MPI library only. In [32] it was proposed to rewrite the MPDATA code and replace conventional HPC systems with modern homogeneous and heterogeneous multi- and many-core based platforms. A new version of MPDATA much better exploited the available computational features of novel processors or Intel Xeon Phi coprocessors. However, the communication cost of MPDATA on modern HPC clusters has not been properly optimized. The current approach to mapping of the partitions of the MPDATA computational domain onto computing resources does not take into account neither the actual properties of the MPDATA communication flow nor the heterogeneity, hierarchy and performance of the communication network. Study and analysis of the communication pattern of the MPDATA application reveals that MPDATA is very sensitive to the choice of the logical topology of processes as the cost per byte of horizontal communications is higher than that of vertical communications even for homogeneous communication networks.

1.4.7 Heuristic Algorithm for MPDATA

The asymmetric communication pattern of MPDATA further complicates the task of partitioning of the MPDATA computational domain and mapping of the sub-domains to the processors in a way that minimizes the cost of communications between different levels of the network hierarchy. In general, finding the optimal arrangement of processors in a 2-D grid is an NP-complete combinatorial optimization problem [33],[34] but it can be approximately solved by using heuristics. For MPDATA, we propose a new algorithm that is built on the top of the cost functions and heuristic of one of our previously proposed algorithms and reduces overall message hops and increases data throughput for a wider range of applications, and apply it to optimization of the communication cost of MPDATA. This algorithm is non-intrusive to the source code of the application and, compared to our previously described algorithms, is not application specific. Our previous algorithms deal with a two-dimensional symmetric communication pattern that is why we tested these algorithms in the context of the parallel matrix multiplication application. With this new algorithm, any data-parallel application with two-dimensional homogeneous computational domain and asymmetric heterogeneous communication pattern can benefit. This algorithm is presented in Chapter 4.

1.5 Thesis structure

The contents of this thesis are organised as follows: In Chapter 2 we present the background and related work, where we discuss the existing heterogeneous platforms and programming challenges these platforms have introduced. Existing work on performance optimization area is also comprehensively reviewed in this chapter. In Chapter 3 we address the problem of efficient execution of data-parallel applications on interconnected clusters and propose two topology-aware optimization algorithms to improve data partitioning by taking into account the entire communication flow of the application. We have used matrix multiplication as a driving example to develop these algorithms. We also demonstrate the correctness and efficiency of the proposed approaches by experimental results. Chapter 4 presents new topology-aware algorithm that is based on cost functions of one of our general heuristic algorithms and applies it to optimization of the communication cost of real-life application, MPDATA. We also present experimental results demonstrating performance gains due to this optimization. Finally, in Chapter 5 we conclude the thesis by drawing conclusions and presenting an insight into the future work.

Chapter 2

Background and Related Work

This chapter aims to discuss high performance computing platforms, particularly the concept of heterogeneity in HPC. It also summarises the programming challenges introduced by heterogeneity. Additionally, it presents a comprehensive literature review of the work to date, mainly focusing on performance optimization area, which is a major challenge encountered by scientific programmers while writing applications for these platforms. Finally, it includes a discussion from the field of simulation explored for HPC systems.

2.1 Heterogeneous HPC Platforms

High-Performance Computing introduced between 1940s-1960s with the development of the first supercomputers. In the past, mainstream supercomputers were homogeneous by design. These supercomputers were used for running scientific applications efficiently, reliably and quickly for more than two decades. Many of these machines contained Symmetric Multiprocessor (SMP) with identical tightly-coupled processors. The demand for heterogeneity in computing systems have increased partially due to the need for high performance computing in recent years with more advanced and complex scientific applications. The shift from homogeneous to heterogeneous has tremendous impact on high performance computing. Many new architectures, network topologies and technologies, algorithms, programming models and tools have been developed under the umbrella of HPC. The earlier heterogeneous HPC systems were composed of single-switched heterogeneous clusters of uniprocessor workstations. Recent heterogeneous HPC systems are hierarchical clusters of heterogeneous hybrid nodes and are composed of dozens of sites, each site is composed of several heterogeneous clusters with thousands of computers which some time have more than 16-core processors. The primary benefit of this many and multi-core heterogeneity is to get increased computational power. However, there are many challenges associated with these heterogeneous systems in terms of performance, scalability, algorithm development, and programmability for parallel processing. A considerable work has been done for algorithms [35]–[37], programming models [26], performance improvement [38], and tools [39], [40] for heterogeneous systems. Despite all this work, heterogeneous HPC systems are still large, complex and difficult to use optimally. This is a broad and open research area; how to model, program and execute parallel applications optimally on these complex, large scale and diverse heterogeneous HPC platforms.

2.2 Programming Challenges on HPC systems

The importance of high performance computing is continually rising and has been emerged as one of the foremost fields of research. The big question is how data parallel complex scientific problems with high computational requirements can be efficiently adapted over current and upcoming heterogeneous high performance architectures [41]. This brings up many challenges to scientific programmers while programming and implementing these problems on heterogeneous HPC platform. Performance optimization, fault tolerance and dealing with arithmetic heterogeneity are perhaps the most challenging issues of heterogeneous parallel and distributed programming [1]. In this section, we briefly review performance optimization as one of the important and challenging issue with respect to heterogeneous parallel and distributed programming.

2.2.1 Performance Optimization for Parallel Applications

Performance optimization is perhaps the utmost important challenge of high performance distributed computing and it becomes more challenging when programming for parallel heterogeneous networks. In heterogeneous HPC systems, heterogeneity of processors, memory heterogeneity, heterogeneity of integration of the processors into the network, and heterogeneity in performance of the processors and the underlying communication networks further complicate the performance optimization task [1], [4], [42]. Considering these heterogeneity, performance optimization broadly divides into two major areas. First, optimal data partitioning and load balancing and second is minimizing the communication overhead to improve the overall performance.

2.2.1.1 Data Partitioning and Load Balancing

While designing parallel algorithm two main issue are, how to sub-divide the main computation task into smaller computation tasks, and how to assign them to different processors for parallel execution. Data decomposition is commonly used method to deal with these issues. First, it partitions the data and then this data partitioning is used to partition the main computation task into smaller sub-tasks. After that these sub-tasks are mapped onto processes. In order to achieve small execution time, overhead of executing the tasks in parallel should be minimized. Load imbalance and inter-process communication are two major sources of this overhead. There is a trade-off between these two objectives. Finding a good mapping is a non-trivial problem. Most of the partitioning problems discussed in literature are either NP-hard or NP-complete [33], [34] . However the authors of these works solve these problems either in polynomial time by applying some constraints, or they propose sub-optimal solutions.

On heterogeneous platform, processors might have different performance speeds. If computation tasks are equally divided on these heterogeneous processors, then fastest processors will quickly perform the computation task and wait for slower processors. Which results in slow execution time due to load imbalance. While dealing with heterogeneous systems, the question arises, how to partition data across the heterogeneous processors to efficiently balance the load so that all processors can achieve near-optimal load balance and finish the computation tasks at the same time? A well written parallel algorithm must takes into account the difference in processors speed. The faster the processor is, the more computations it has to perform.

Data partitioning and load balancing area has been well studied in literature [43]–[45]. The work has been broadly classified into two categories: static and dynamic. Static algorithm of data partitioning [46]–[48] distributes the computation tasks among the processes prior to the execution of application. Static algorithms are useful when data locality is important because they do not require redistribution of data; hence, result in improved data access and transfer within application. It is the case with parallel applications dealing with large amount of data. However, on non-dedicated platform these algorithms are unable to balance the workload. Dynamic algorithms, on the other hand, distribute the tasks among processes during the time of execution of the application [49]–[51]. These techniques incur significant communication overhead on distributed memory platforms due to data migration which may eleminate their benefits. It was shown that static distribution techniques are more stable and can offer better performance than traditional dynamic techniques on heterogeneous distributed systems [52], [53]

Performance models of processors are crucial for efficient data partitioning. For heterogeneous systems, two types of models were shown in the literature: *Constant Performance Model* (CPM) and *Functional Performance Model* (FPM). Constant performance model assumes that relative speed of heterogeneous processors does not depend on the size of the computational task solved by the processors and it remain constant and represented by a single positive number. However, in reality the relative speed changes due to processor and memory heterogeneity [48]. Which make CPM inaccurate and unrealistic. CPM is used in many load balancing data partitioning and scheduling algorithms which target heterogeneous platforms [7], [54]–[56]. For more accurate performance modelling, Functional Performance Model (FPM) has been proposed in [48], [57]. In FPM, the speed of each process is represented by a continuous function of problem size. The speed is defined as the number of computation units performed by the process per one time unit. The problem of data partitioning using FPM was solved in many works [9], [43], [45], [48], [58]. FPM is more elaborately discuss in Section 3.

2.2.1.2 Communication Optimization

In the broad overview of optimization of communications on heterogeneous HPC platforms [10], all existing techniques are classified as topology or performance-aware. In high performance computing, two basic programming models are the shared memory model and the distributed memory model, depending on the programmer's view of the system memory. In distributed memory model, application runs as a collection of autonomous processes, each with its own local memory. Processes communicate with other processes by sending and receiving messages and it is common to have explicit communication between processes through these messages passing. The most popular message-passing programming interface use for this purpose is MPI [59], [60]. Originally, MPI was designed for distributed memory architectures. However, as architecture trends changed, and shared memory (SMPs) were combined over networks creating hybrid distributed memory / shared memory systems. MPI handles any kind of underlying memory architectures seamlessly. Which makes it popular choice for the message-passing programming paradigm on distributed-memory high-performance computing systems since last two decade. MPI communication primitives (both point-to-point and collectives) are extensively used across various scientific HPC applications [60], [61]. There is a large body of research on optimizing these MPI operations for communication optimization of the parallel applications.

2.2.1.3 Performance-Aware Communication Optimization

The main idea of performance-aware optimization is to increase data throughput by a scheduling based on the performance of individual links. This approach is used when the topology information is not available. Performance of individual link is evaluated with the help of point-to-point communication performance models, which are estimated from communication benchmarks.

A number of performance-aware implementations of MPI collective communication operations have been proposed in the literature [62]–[66]. The execution time of collective communication operations can be predicted with the help of analytical communication performance models. These models capture the behaviour of MPI collective operations under significantly large number of physically established parameters. Several benchmarking libraries and tools are available to estimate the parameters of the model [67], [68]. In [69], a model for capturing the congestion on Ethernet clusters for collective operations is developed. [70] proposed a new congestion model for hierarchical Ethernet networks. The predictions can be used for optimization of collective communications. One approach is to choose the collective algorithm with the minimal predicted time from a given set of algorithms [63]. Another approach is to use the predictions for building more optimal communication trees for a collective communication [71].

2.2.1.4 Topology-Aware Communication Optimization

Topology-aware is a term that most probably originates from the networking domain [72]. Topology-aware optimization is use to reduce communication traffic and contention by placing communicating tasks on physically nearby processors. Communication traffic is quantified by the number of links a message traverses. Contention is caused by multiple messages sharing a network link. Performance of an application heavily depends upon how the MPI library has been designed and optimized to take into account the system architecture. Researchers have demonstrated that network topology plays a critical role in the performance of MPI communication primitives [73], [74]. However, designing topology-aware MPI libraries that manage the communication for both point-to-point and collectives based on the underlying network topology, is still an open challenge. A number of topology-aware implementations of MPI collective communication operations have been proposed in the literature. Most MPI libraries such as MPICH2[75], OpenMPI [76] use multi-core aware, shared-memory based techniques to optimize the latency of collective operations [77]. A number of node level topology-aware optimization techniques have already been proposed [78], [79]. However, these techniques are limited to node level hardware topology and do not consider the network topology. In [80] portable hardware locality tool, hwloc, has been proposed that is widely used now a days for complex intra-node topology discovery.

For interconnected clusters, a two-level communication graph is constructed [14] so that the clusters communicate via selected nodes, coordinators, which form the inter-cluster communicator. All nodes within a cluster communicate with the cluster coordinator, forming the intra-cluster communicator. These optimized implementations send the minimal amount of data over the slow wide-area links.

Collective operations are optimized for multilevel hierarchical heterogeneous networks and Grid [15] and [81]. Hierarchical approach is applied to optimize collectives for multi-core clusters: inter- and intra-node communications are overlapped, using offloading and pipelining techniques [16]. Homogeneous supercomputers with complex network topologies, like BlueGene and Cray, can also benefit from topology-aware collectives [13].

Moreover with topology-aware MPI primitives, the placement of logical MPI ranks on a heterogeneous HPC system can also significantly affect overall application performance. A random rank assignment can result in poor locality of communication. Thus, it is important to design optimal mapping schemes with topology information and communication pattern of application to improve the overall application performance and scalability. Even with the best MPI library, if topology-aware mapping of logical MPI ranks is not done, the performance of parallel application can suffer.

MPI implementations try to exploit target architectures as efficiently as possible by using the most suitable communication channels and best algorithms for collective communication operations. Therefore, many existing MPI applications can be executed efficiently on hierarchical heterogeneous HPC platforms, without any modifications of the source code. However, the approach of topology- or performance-aware collectives does not address applications based on point-to-point exchanges.

Many high performance computing applications are designed using MPI point-to-point communication to transfer large amounts of data between various processes. Nearest neighbour applications and multi-dimensional stencil based applications [30] are the common examples that extensively use MPI point-to-point communication. In these applications, each process communicates with its neighbours after each time step. If neighbour processes are not topologically closed together, these same time messages exchange generates a significant pressure on the network. For such applications, it is an open challenge whether it is possible to achieve performance improvement by optimizing point-to-point messaging with topology information.

The problem of topology-aware optimization of point-to-point communications can be solved by introducing a graph that represents the application communication flow and is mapped onto the network topology. This approach has been applied to the mesh and graph virtual MPI topologies on SMP clusters [18] and to the mesh topology on BlueGene/L [12]. A tool for automatic profile-guided process placement has been developed for interconnected clusters [19]. In all these work, the heterogeneity of processors is not taken into account and therefore the processes are placed freely to processors in order to minimize the communication cost. In this context, it is crucial to design network-topology-aware mapping mechanisms to optimize the performance of point-to-point operations on large scale heterogeneous systems.

2.3 Simulation in HPC Systems

Simulation is a well known field in the world of computer science. It is used for designing the model of real or theoretical physical system, executing that model, and analysing the execution output. This field has been flourished in the period of 1970 to 1981, during which computer scientist developed many enhanced modeling and analytical tools [82]. This section cover the role of simulation in high performance heterogeneous computing platform.

2.3.1 Parallel and Distributed System Simulation

The role of simulation has been exploded in the last decade for parallel and distributed systems. Simulation is a popular approach for predicting the performance of large-scale parallel scientific applications on large-scale platforms that are not at one's disposal. This performance prediction and profiling is helpful for developing and maintaining the HPC application code that is expected to scale for current and next generation large-scale HPC systems.

There are many constraints while access to these HPC platforms in real. They are expensive to access, in terms of access charges to user, time restriction, number of resource allocation etc. The primary concern behind simulation of these high performance complex parallel and distributed system is to analyse the behaviour of scientific applications and full-scale implementation on such platforms without real access to these platforms [83]. However, simulation of an application on such platforms may also be useful even when the platforms are available. For example, simulation may bypass the actual computation, and only simulate the communication pattern and delays of these computations. Based on simulated result, one can easily perform the performance tuning of application in simulation without real execution on a large scale . It will not only save the time but also money and resources.

Accuracy, scalability and speed are the three possibly main challenges for simulating parallel applications [84]. A Lot of work has been done to address some or all of these challenges [84]–[87]. This simulation work falls into two categories: off-line simulation and on-line simulation. Off-line simulation also called trace-based simulation relies on application log, or trace that is composed of communication events. These traces are obtained by running the parallel application on a real-world platform. Based on these off-line traces, simulator then re-execute the application on simulated platform. A number of trace-based simulators have been found in the literature [88]–[92]. However, the big challenge for off-line simulation is the large size of the traces, which sometimes prevent running the simulation on a single node. On-line simulation, also called simulation via direct execution of application avoids this challenge. In on-line simulation the application is executed but part of the execution takes place within a simulation component. This approach is more general because it does not require traces or log obtained for any application and platform configuration. The popular on-line parallel application simulators are MPI-SIM, MPI-NetSim, PEVPM, SimGrid etc. [29], [93]–[95] . Among them, SimGrid [29] is the latest framework that is helpful to study the behaviour of modern large-scale distributed systems such as Grids, Clouds, HPC or Point-to-Point (P2P) systems. It provides ready to use models and API to simulate many different distributed systems. SimGrid toolkit provides three main models: MSG, Simulated Direct Acyclic Graph (SimDag) and SMPI. MSG, was the first distributed programming environment provided within SimGrid. It is a simple parallel application level simulator. For on-line simulation of MPI based parallel application, they developed SMPI simulator [84]. It helps to run application on top of any virtual environment. Whereas, SimDag is a framework for DAG's of parallel tasks. It provides some functionalities to simulate parallel task scheduling with DAGs models (Direct Acyclic Graphs). In our work, we use SMPI to run MPI matrix multiplication application.

Chapter 3

Design and Evaluation of Topology-Aware Mapping Algorithms for Heterogeneous Hierarchical HPC Platforms

3.1 Introduction

We identify in Chapter 2, many high performance computing applications are designed using MPI point-to-point communication to transfer large amounts of data between various processes. For such applications, it is an open challenge whether it is possible to achieve performance improvement by providing topology-aware mapping. Because placement of logical MPI ranks on heterogeneous HPC system also has a significant impact on performance as certain mappings will be more advantageous than others. In this chapter, we take up this challenge. We employ a case-study to clearly motivate the need for topology-aware process mapping to reduce the communication overheads.

We propose communication optimizations of point-to-point communications for parallel matrix multiplication on hierarchical heterogeneous platforms. Assuming that the data have been optimally partitioned between the processors, this optimization is based on re-arrangement of the rectangles of the matrix partition. In general, finding the optimal arrangement of processors in a 2-D grid is an NP-complete combinatorial optimization problem [34], but it can be approximately solved by using heuristics [96]. We propose two heuristic solutions based on evaluation of the application communication flow on the given network topology. To evaluate the communication cost of an arrangement, we propose the cost function that estimate cost by using number of message hops between clusters and the bandwidth information. We also demonstrate the accuracy and efficiency of the proposed solution on experiments with interconnected clusters.

3.2 Driving Example

As a case study, we consider parallel matrix multiplication application for heterogeneous platforms. Matrix multiplication is a very important computation kernel and a building block of many scientific applications, for example Gaussian elimination and LU decomposition, which in turn used to solve many scientific problems. All such applications will benefit from any optimization made in matrix multiplication. Furthermore, if algorithm can be applied successfully to parallel matrix multiplication, it can be scale to other tightly coupled parallel applications. Because we design the heuristic algorithm with wider applicability in mind.

In this section, we describe parallel matrix multiplication algorithms based on SUMMA, with the emphasis on their communication flow. We consider these algorithms because of their applicability to a wide range of HPC platforms. These algorithms can be executed on the platforms that do not form a 2D grid of processors. The workload in these algorithms can be balanced by irregular matrix partitioning, proportional to the speed of processors. The volume of communications can be minimized. We also demonstrate that communication performance of parallel matrix multiplication on modern hierarchical HPC platforms can be improved further by taking into account information about network topology. However, to the best of the our knowledge, all existing modifications of SUMMA are topology-unaware.

3.2.1 Parallel Matrix Multiplication on Heterogeneous Platforms

The SUMMA [27] is designed for homogeneous platforms and implements parallel matrix multiplication $C = A \times B$. In this algorithm, dense matrices are partitioned over a 2D grid of processors. Each processor is a part of two MPI communicators that combine all processors in the same row and column. To take advantage of processor cache, a blocking factor, b, has been introduced, so that each matrix consists of $b \times b$ blocks. The algorithm iterates over the columns of blocks of matrix A and over the rows of blocks of matrix B. At each iteration, a column of blocks (the pivot column), $A_{(b)}$, is broadcast horizontally, and a row of blocks (the pivot row), $B_{(b)}$ is broadcast vertically. Then, matrix C is updated on all processors in parallel: $C_i + = A_{(b)} \times B_{(b)}$. At the end of each iteration, the pivot column and row move horizontally and vertically respectively.

The update operation can be performed efficiently by invoking a highly optimized general matrix multiplication (GEMM) routine, available for most HPC platforms. This operation can be considered as a computation kernel of the application because it represents the computation performance of the entire application. Fig. 3.1 shows the communication flow of SUMMA, which consists of the broadcasts in the row and column communicators. The broadcasts pass the pivot column and row in rings, pipelining computations and communications.

Heterogeneous modifications of SUMMA are based on the approach to optimization of linear algebra computations on heterogeneous platforms [54]. In this approach, to balance the load of heterogeneous processors, the matrices are partitioned into uneven rectangles such that faster processors will process larger rectangles. Ideally, this way each processor should receive the workload proportional to its computing power. In the case of heterogeneous SUMMA, the amount of computations related to the *i*-th rectangle will be proportional to d_i, the number of blocks it contains. A

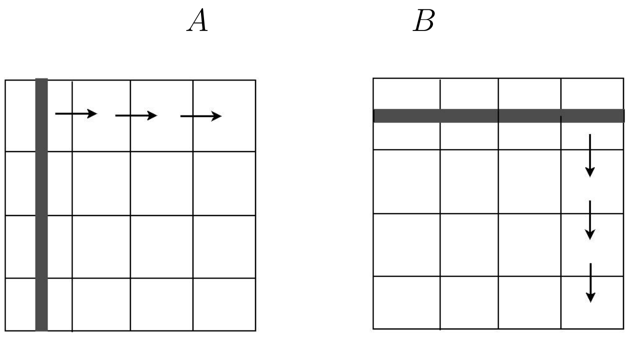

Figure 3.1: Communication flow of SUMMA

number of efficient matrix partitioning algorithms have been proposed, returning matrix partitions with different arrangements of rectangles [98], [7], [99], [100]. However, the most popular heterogeneous matrix multiplication algorithms implement column-based partitioning, when processors are arranged into columns, and all processors in a column are allocated rectangles of the same width. The widths of all the columns sum to the width of the matrix. The heights of rectangles in a column sum to the height of the matrix. More elaborated irregular matrix partitioning [101], [102] is out of the scope of this work. In the overview of the heterogeneous column-based algorithms [9], two main directions of development are defined: minimization of the volume of communications, and data partitioning based on accurate computation performance models of processors.

The algorithm minimizing the total volume of communication [7] arranges the processes into columns and sets the rectangles' dimensions (m_i, n_i), using the relative cycle times of processors as input. The total volume of communication is proportional to the sum of half-perimeters $\sum_{i=1}^{p}(m_i + n_i)$. The shape and ordering of rectangles are calculated to minimize this sum. The algorithm returns the optimal number of columns, the optimal number of rectangles in each column and the optimal dimensions of rectangles. The resulting rectangles are sorted in the order of increasing area, $d_i = m_i \times n_i$, with the shape as square as possible.

Fig. 3.2 describes the communication flow of the algorithm, which will be denoted by BR for the rest of the text. It consists of one-to-all non-blocking point-to-point communications in horizontal and vertical directions. In the horizontal direction, these communications are irregular: each processor holding a part of the pivot column sends multiple messages of different sizes to all processors in horizontal direction, whose rectangles are overlapped with the sender's rectangle. The size of each message is equal to the block size times the height of the overlap between the sender and receiver. In other words, the overlap is the maximum part of the pivot column required on the receiver to perform its local update operation. It should be noted that this communication pattern is not scalable if the number of communicating processors increases.

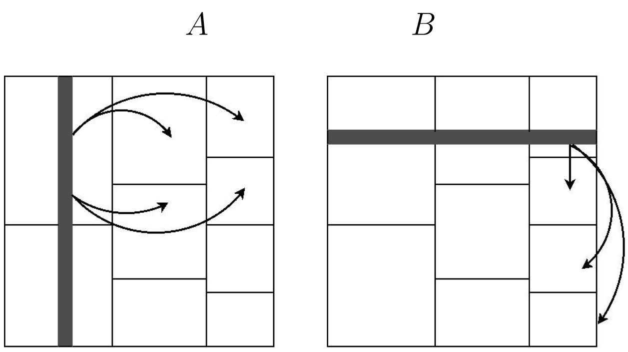

Figure 3.2: Comm. flow of heterogeneous SUMMA: one-to-all

The main shortcoming of the BR algorithm is that it uses simplistic performance model of processors, where processor speed is represented by a single positive number. This approach may fail to balance the load, especially for highly heterogeneous platforms and self-adaptable applications. More reliable solution is data partitioning based on accurate performance models, such as functional performance model FPM [48]. It is built empirically and integrates many important features characterizing the performance of both the architecture and the application.

Under the functional performance model, the speed of each process is represented by a continuous function of problem size. The speed is defined as the number of computation units performed by the process per one time unit. The computation unit can be defined differently for different applications, but it is required not to vary during the execution of the application. For SUMMA-based matrix multiplication, it can be defined as one update of one $b \times b$ matrix block: $C_{b\times b} += A_{b\times b} \times B_{b\times b}$. In this case, the problem size assigned to a process is given by the number of $b \times b$ blocks. The amount of computations assigned to the process is proportional to the area of the rectangle formed by these blocks.

The processor speed is found experimentally by measuring the execution time over a range of problem sizes. This time can be found by benchmarking the full application. This benchmarking can be done more efficiently by using a serial code, the speed of execution of which is the same as that of the application but the execution time of which is significantly less. A benchmark made of one such core computation can be representative of the performance of the whole application and can be used as a kernel. The speed function of the application can be built more efficiently by timing this kernel. For SUMMA-based matrix multiplication, one update of a rectangle $C_i += A_{(b)} \times B_{(b)}$, implemented by highly optimized GEMM and performed many times for different pivot rows and columns, can be used as a kernel.

The problem of data partitioning using functional performance models was formulated and solved in [48]. Then, FPM-based data partitioning was applied to the BR algorithm [9]. We will refer to this modification of matrix multiplication as The 2D-FPM based Matrix Partitioning Algorithm (FPM-BR) for the rest of the text. For FPM-BR algorithm, another communication scheme was implemented, which consists of non-blocking point-to-point communications in rings, in horizontal and vertical directions (see Fig. 3.3). In contrast to the one-to-all communication flow, each processor communicates only with the processors from its neighbouring columns and rows. In horizontal direction, the partition is irregular, and the processor holding the pivot row sends multiple messages to its right column. These messages can be addressed to the same processor. The size of each message is equal to the block size times the height of the overlap between all rectangles in the horizontal direction. Here the overlap is the maximum part of the pivot column that can be transmitted over the ring of processors.

Table 3.1 summarizes the above-mentioned matrix multiplication algorithms based on SUMMA. The FPM-BR algorithm better balances the

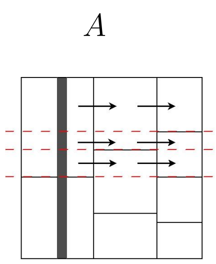

Figure 3.3: Comm. flow of heterogeneous SUMMA: ring

workload and minimizes the total volume of communications. However, none of the algorithms takes into account the underlying networks topology, so that they are not communication-optimal. In this work, we propose to rearrange a given heterogeneous data partition in order to reduce the number of message hops and better use the available network bandwidth.

Table 3.1: Comparison of some SUMMA-based algorithms

Algorithm	Data partitioning	Comm. vol.	Comm. flow
SUMMA	homogeneous	–	broadcasts
BR	constant speeds	min	nb-p2p one-to-all
FPM-BR	speed functions	min	nb-p2p one-to-all/ring

In this work, we propose both topology- and performance-aware optimizations of point-to-point communications for parallel matrix multiplication on hierarchical heterogeneous platforms. In the target application, the data (matrices) is distributed in proportion to the speed of processors. Assuming that the workload is balanced among the processors, we propose to rearrange the given heterogeneous data partition in order to reduce the number of message hops and increase data throughput. This rearrangement is based on network topology, network properties, and communication pattern of the application. This approach is also non-intrusive to the source code but application-specific.

3.2.2 Communication-Optimal Matrix Partitioning

In this section, we formulate the problem of communication-optimal matrix partitioning for heterogeneous SUMMA on interconnected heterogeneous HPC clusters. To minimize communication cost, we use information about the network topology and the application communication flow.

In our target platform, interconnected heterogeneous HPC clusters, the network can be represented as a two-level rooted tree with faster communications within sub-trees (clusters) and slower communications between. Within each cluster, a single network switch provides no-contention point-to-point communications, appropriately forwarding packets between sources and destinations. Inter-cluster links may be shared by multiple processors from different clusters communicating with each other.

Our goal is to minimize communication cost of the parallel application that implements the FPM-BR matrix multiplication algorithm. In this application, each processor is assigned a matrix rectangle of the area and shape that balance the workload and minimize the communication volume. The communication flow of this application is based on non-blocking point-to-point communications in rings. Changing the position of a rectangle within the matrix does not affect the load balance and the communication volume, but the rectangles can be arranged so as to minimize the cost of communications between the processors. This forms the optimization problem we solve in this work.

Since column widths are different, we cannot move a rectangle to another column unless the whole columns are interchanged. In a column, there are no restrictions on interchanges of rectangles. All these limit the solution space of our optimization problem to a certain number of combinations. Let c be the number of columns and r_i be the number of rectangles in column i, $1 \leq i \leq c$. Then the number of combinations will be equal to the product $r_1! \times \ldots \times r_c!$. Which arrangement of rectangles is communication-optimal? This is an NP-complete problem.

We performed exhaustive search by running the application with all possible arrangements of rectangles on a small platform of three interconnected heterogeneous clusters. Each cluster consisted of several heterogeneous nodes, which were assigned rectangles proportional to their speed. From the exhaustive search, we found several arrangements that reduced (Fig. 3.4) and increased (Fig. 3.5) the communication cost (different colors/fillings correspond to different clusters). We observed some regularity in the communication-optimal arrangements, which was related to the topology. In the optimal arrangements, the rectangles were grouped by clusters, whereas, in the worst cases, the rectangles assigned to the same cluster were dispersed vertically and horizontally. With the optimal arrangements, the application, which is based on non-blocking point-to-point communications in rings, performs less inter-cluster communications in horizontal and vertical directions. In addition, in the optimal cases, data throughput in rings is higher due to less use of slow inter-cluster links.

Figure 3.4: Some of the communication-optimal arrangements

Figure 3.5: Some of the worst case arrangements

The factorial design of the exhaustive search leads to a large number of trials, which becomes infeasible for large platforms. If topology information is available, we can avoid exhaustive search by applying some heuristic that efficiently finds a near-optimal arrangement.

3.3 Cost Functions

Heuristic search requires to estimate the communication cost incurred by each partition. Communication cost can be estimated by taking into account the application communication flow and the network topology or network properties. Using the observations from the exhaustive search, we propose two cost functions for the FPM-BR matrix multiplication with the ring communication flow and two-level network hierarchy. One function estimates the number and volume of inter-cluster communications incurred by an arrangement of matrix rectangles. Another estimates the communication time, using the bandwidth properties of individual links.

3.3.1 Cost Function Based on Message Hops

In the FPM-BR-ring algorithm, the point-to-point communications in the vertical direction are related to matrix B (see Fig. 3.3). The volume of communications in each column is proportional to the column width. The number of communicating clusters in the vertical direction remains the same for any arrangement of matrix rectangles. The number of inter-cluster communications is proportional to the number of message hops between clusters. In the communication-optimal arrangements, the rectangles are grouped by clusters in each column. In this configuration, the number of message hops between clusters is minimal in each column. In the worst cases, the rectangles belonging to the same group are dispersed.

To estimate the inter-cluster communication cost, we take the upper bound of the number of hops made to send the pivot row over the ring in the column. The rightmost column of the optimal arrangement in Fig. 3.6 illustrates the upper bound of the number of hops. Namely, when the pivot row is on the top of the matrix, there will be only one communication between clusters: between the processors holding the second and third rectangles. The same happens when the pivot row is in the third rectangle: a part of the pivot row is sent between the processors from different clusters that hold the fourth and first rectangles. In other cases, when the pivot row is in second and fourth rectangles, two inter-cluster communications are performed.

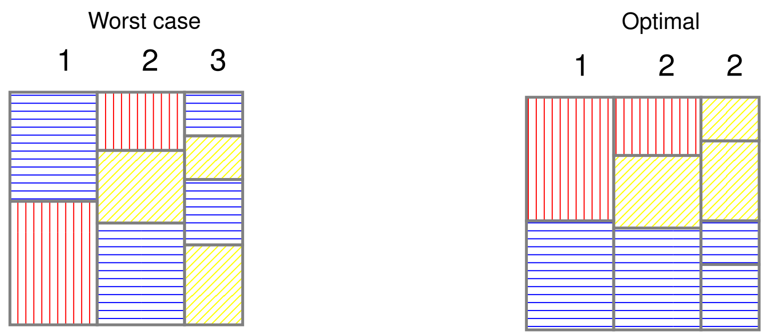

Figure 3.6: Inter-cluster communications related to matrix B

We define the cost function for the inter-cluster communications related to matrix B as follows:

$$cost_B = \sum_{i=1}^{c} h(i) \times v(i), \tag{3.1}$$

where variable i iterates over the columns of matrix rectangles, functions h and v return the number of inter-cluster communications in a column and the column width respectively. This cost corresponds to one iteration of parallel matrix multiplication. Communications in columns are performed in parallel, therefore, each inter-cluster link may be used for multiple simultaneous exchanges. For example, in Fig. 3.6, inter-cluster links are shared by the processors from two columns. Therefore, instead of maximum, which represents parallelism, we use sum, which represents the case when all the inter-cluster links are used simultaneously for communications in all columns. The cost of the arrangements in Fig. 3.6 is then calculated as follows:

- Worst case: $cost_B = (1 \times 12) + (2 \times 12) + (3 \times 9) = 63$
- Optimal: $cost_B = (1 \times 12) + (2 \times 12) + (2 \times 9) = 54$

The point-to-point communications in the horizontal direction are related to matrix A (see Fig. 3.3). The number of communicating clusters and the volume of inter-cluster communications depend on the arrangement. The number of inter-cluster communications along the pivot column varies. The volume of inter-cluster communications is proportional to the height of overlaps of matrix rectangles. The overlap is the maximum part of the pivot column that can be transmitted over the ring of processors in the horizontal direction. Fig. 3.7 illustrates both the numbers of inter-cluster communications in overlaps and the heights of overlaps. In the optimal arrangement, the rectangles assigned to the same cluster are grouped in rows as much as possible, while in the worst case, they are scattered over the matrix.

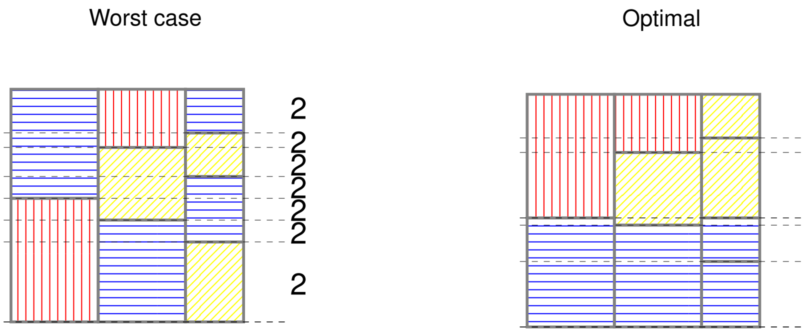

Figure 3.7: Inter-cluster communications related to matrix A

Similarly to the communications related to matrix B, we use the upper bound of the number of inter-cluster communications. For example, in the optimal arrangement in Fig. 3.7, the number of inter-cluster communications over the upper part of matrix A varies from one to two, depending on the location of the pivot column. If the pivot column is in the second column of matrix rectangles, there will be two inter-cluster communications in two top rings.

We define the cost function for the inter-cluster communications related to matrix A as follows:

$$cost_A = \sum_{i=1}^{o} h(i) \times v(i), \tag{3.2}$$

where variable i iterates over the o overlaps of matrix rectangles, functions h and v return the number of inter-cluster communications in an overlap and the height of the overlap. This cost corresponds to one iteration of parallel matrix multiplication. Similarly to $cost_B$, we use sum as the upper bound of the communication cost. The cost of the arrangements in Fig. 3.7 is calculated as follows:

- Worst case: $cost_A = 2 \times (11 + 3 + 3 + 3 + 4 + 2 + 6) = 64$
- Optimal: $cost_A = 1 \times (6 + 8) + 2 \times (1 + 9 + 2 + 6) = 50$

To conclude, the inter-cluster communication cost associated with arrangement M is represented by two values $(cost_A(M), cost_B(M))$. The problem of finding the communication-optimal arrangement can be formulated as minimization of their sum:

$$cost_A(M) + cost_B(M) \to \min . \tag{3.3}$$

This sum represents a combined cost and can be used to compare any two arrangements. The combined cost of the above arrangements is equal to $64 + 63 = 127$ and $50 + 54 = 104$ respectively. Table 3.2 summarizes their execution time and inter-cluster communication cost.

Table 3.2: Exhaustive search experimental results

	Cost		Exec time (sec)	
	Worst case	Optimal	Worst case	Optimal
Exhaustive search	127	104	6.00	2.78

3.3.2 Cost Function Based on Network Bandwidth

Communication cost can be estimated more accurately if information on the network performance is available along with the network topology. For example, let us consider four interconnected clusters, numbered from 0 to 3, such that inter-cluster links 0-1 and 2-3 are significantly slower than 0-2 and 1-3. Hence, sending a message in ring 0-1-2-3 will take more time than in ring 0-2-1-3. The cost function presented in previous section does not distinguish such cases, returning the same $cost_B$ value for different arrangements of clustered rectangles. That function estimates the volume of inter-cluster communications. If information on network performance is available, it is possible to estimate the communication time. Moreover, if such information is available for individual links, the estimate can include not only inter-cluster but also intra-cluster contributions. For example, a message sent in a ring of four processors 0-0-1-2, with two processors from the same cluster, will traverse both intra- and inter-cluster links. This fact can be reflected in a cost function.

We define the performance-aware cost function for the communications related to matrix B as follows:

$$cost_B = \sum_{i=1}^{c} \left(v(i) \times \sum_{j=1}^{r_i} \frac{1}{b(j, j+1)} \right), \tag{3.4}$$

where variable i iterates over the columns, and variable j iterates over the matrix rectangles in each column. Function $v(i)$ returns the width of column i (in bytes). Function $b(j, j+1)$ returns the bandwidth (in bytes per second) between the processors holding rectangles j and $j+1$. For $j = r_i$, it is defined as $b(j, j+1) := b(j, 1)$. Therefore, this cost function estimates communication time in seconds. The inner sum represents sending a part of the pivot row in a ring. The outer sum represents the upper bound of communication time required to send the whole pivot row over all column rings. We use the upper bound because the bandwidth of some links may be divided between multiple communications corresponding to different columns.

Fig. 3.8 shows the worst case and optimal arrangements for 16 processors from 4 interconnected clusters. Blue, red, yellow, and green colors specify rectangles assigned to cluster 0, 1, 2, and 3 respectively. Table 3.3 summarizes the bandwidths of different links, including intra-cluster links. Communication cost $cost_B$ of the arrangements in Fig. 3.8 is calculated in Table 3.4 and Table 3.5.

Table 3.3: Bandwidth of communicating links (MB/sec)

	0	1	2	3
0	891.20	55.01	249.53	56.39
1	55.01	893.13	68.96	90.88
2	249.53	68.96	6850.27	71.93
3	56.39	90.88	71.93	896.17

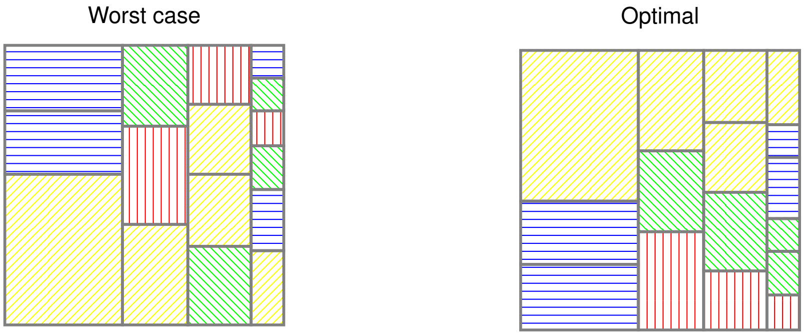

Figure 3.8: Worst case and optimal arrangements for 16 heterogeneous processors from 4 clusters

Table 3.4: Communication cost $cost_B$ computed for the worst case in Fig. 3.8

Column width $\times$ data size (byte)	Communication cost per byte (sec/byte)
54 $\times$ 512	1/249.53 + 1/891.20 + 1/249.53
30 $\times$ 512	1/68.96 + 1/90.88 + 1/56.39
29 $\times$ 512	1/71.93 + 1/6850.27 + 1/68.96 + 1/90.88
15 $\times$ 512	1/249.53 + 1/56.39 + 1/90.88 + 1/90.88 + 1/36.39
Communication cost (sec)	1948.15

We define the performance-aware cost function related to matrix A in the same way:

$$cost_A = \sum_{i=1}^{o} \left(v(i) \times \sum_{j=1}^{c} \frac{1}{b(j, j+1)} \right), \tag{3.5}$$

where variable i iterates over the overlaps, and variable j iterates over matrix rectangles in each overlap. Function v returns the height of an overlap i (in bytes). Function $b(j, j+1)$ returns the bandwidth (in bytes per second) between the processors holding the rectangles j and $j+1$. For $j = c$, it is defined as $b(j, j+1) := b(j, 1)$. Therefore, this cost function estimates communication time in seconds. The inner sum represents sending a part of the pivot column in a ring. The outer sum represents the upper bound of communication time required to send the whole pivot column over all overlap rings. We use the upper bound because the bandwidth of some links may be divided between multiple communications corresponding to different overlap.

Table 3.5: Communication cost $cost_B$ computed for the optimal case in Fig. 3.8

Column width $\times$ data size (byte)	Communication cost per byte (sec/byte)
54×512	1/891.20 + 1/249.53 + 1/249.53
30×512	1/90.88 + 1/56.39 + 1/68.96
29×512	1/90.88 + 1/71.93 + 1/6850.27 + 1/68.96
15×512	1/90.88 + 1/896.17 + 1/56.39 + 1/891.20 + 1/249.53 + 1/68.96
Communication cost (sec)	1825.23

Communication cost $cost_A$ of the arrangements in Fig. 3.8 is calculated in Table 3.6 and Table 3.7.

Table 3.6: Communication cost $cost_A$ computed for the worst case in Fig. 3.8

Overlap height $\times$ data size (byte)	Communication cost per byte (sec/byte)
15×512	1/56.38 + 1/90.87 + 1/55.00 + 1/891.20
12×512	1/56.38 + 1/90.87 + 1/90.87 + 1/56.38
3×512	1/56.38 + 1/71.93 + 1/71.93 + 1/56.38
7×512	1/56.38 + 1/71.93 + 1/68.95 + 1/55.00
9×512	1/55.00 + 1/68.95 + 1/68.95 + 1/55.00
13×512	1/55.00 + 1/68.95 + 1/71.93 + 1/56.38
7×512	1/68.95 + 1/68.95 + 1/71.93 + 1/71.93
16×512	1/68.95 + 1/68.95 + 1/249.52 + 1/249.52
10×512	1/6850.27 + 1/6850.27 + 1/249.52 + 1/249.52
2×512	1/6850.27 + 1/71.93 + 1/56.38 + 1/249.52
34×512	1/6850.27 + 1/71.93 + 1/71.93 + 1/6850.27
Communication cost (sec)	2854.13

The communication cost associated with arrangement M is represented by two values $(cost_A(M), cost_B(M))$. The problem of finding the communication-optimal arrangement can be formulated as minimization of their sum:

$$cost_A(M) + cost_B(M) \to \min . \tag{3.6}$$

The combined costs of the above arrangements are $2854.13 + 1948.15 = 4802.28$ and $1784.56 + 1825.23 = 3609.79$ respectively.

Table 3.7: Communication cost $cost_A$ computed for the optimal case in Fig. 3.8

Overlap height × data size (byte)	Communication cost per byte (sec/byte)
33 × 512	1/6850.27 + 1/6850.27 + 1/6850.27 + 1/6850.27
1 × 512	1/6850.27 + 1/6850.27 + 1/6850.27 + 1/6850.27
12 × 512	1/6850.27 + 1/6850.27 + 1/249.52 + 1/249.52
3 × 512	1/71.93 + 1/71.93 + 1/249.52 + 1/249.52
16 × 512	1/71.93 + 1/71.93 + 1/249.52 + 1/249.52
4 × 512	1/71.93 + 1/896.16 + 1/56.38 + 1/249.52
8 × 512	1/56.38 + 1/896.16 + 1/56.38 + 1/891.20
6 × 512	1/56.38 + 1/896.16 + 1/896.16 + 1/56.38
9 × 512	1/55.00 + 1/90.87 + 1/896.16 + 1/56.38
6 × 512	1/55.00 + 1/90.87 + 1/896.16 + 1/56.38
3 × 512	1/55.00 + 1/90.87 + 1/896.16 + 1/56.38
11 × 512	1/55.00 + 1/893.13 + 1/90.87 + 1/56.38
16 × 512	1/55.00 + 1/893.13 + 1/893.13 + 1/55.00
Communication cost (sec)	1784.56

In the next section, we show how these intuitive and based on the observations cost functions can be used in a heuristic solution of the combinatorial problem of topology-aware optimization of communication cost in the heterogeneous matrix multiplication application.

3.4 Heuristic Search

In this section, we use information about the network topology/performance and the application communication flow in two heuristics that efficiently construct a near-optimal arrangement. These heuristics do not require to run the application to collect information about its communication performance. They use different cost functions and therefore have slightly different designs. Their main idea is to reduce the search space of rectangle arrangements and find the one that minimizes the communication cost of the application.

3.4.1 Heuristic Based on the Hop-count Cost Function

The first heuristic uses the cost function estimating the volume of inter-cluster communications and can be summarized as follows. First, in columns, we group the rectangles assigned to the same subnetwork. This will minimize the inter-cluster communication cost related to matrix B. Then, we rearrange the groups of rectangles in columns to minimize the inter-cluster communications related to matrix A. Let us present the rationale for such a solution and describe the solution in detail.

Finding the optimal arrangement is complicated by irregularity of communications over rows, which is related to matrix M. We propose to apply cost function $cost_A$ not to the whole matrix but to some of its columns. In such a way, $cost_A(M_1, \ldots, M_i)$ estimates the cost of communications between the first i columns of rectangles. Here M_i is the i-th column of matrix rectangles. We will construct the near-optimal arrangement by minimizing this cost function for successive submatrices that consist of two, three or more columns of rectangles: $(M_1, M_2), (M_1, M_2, M_3), \ldots$.

Let us assume that the rectangles in the first $i-1$ columns have been rearranged to minimize the cost: $cost_A(M_1, \ldots, M_{i-1}) = \min$. With these columns fixed, we can estimate the cost of i columns, with different permutations of rectangles in the i-th column. The permutation providing the minimal combined cost can be added to the solution. This approach reduces the number and volume of inter-cluster communications but does not guarantee finding a global minimum. It allows us to test a significantly smaller number of combinations of rectangles, which is equal to the sum (not the product) of permutations: $r_2! + \ldots + r_c!$, where r_i is the number of rectangles in column i.

We observed that in communication-optimal arrangements the matrix rectangles assigned to the same network subtree were grouped (Fig. 3.4). Indeed, this minimizes the number of hops between subnetworks, $g(i)$, in each column i, and therefore, reduces the communication cost related to matrix B, $cost_B$. If the rectangles in columns are grouped, we will have to estimate $cost_A(M_1, \ldots, M_i)$ for significantly less number of combinations. For

g_i communicating clusters in column i, there will be $g_i!$ permutations of the grouped matrix rectangles. The number of combinations of groups $g_i!$ is significantly smaller than the number of combinations of individual rectangles $r_i!$.

In one of the optimal cases, namely, in the left picture of Fig. 3.4, one group of rectangles in the third column is split between the top and bottom of the column. In this case, the upper bound on the number of inter-cluster communications remains minimal, two, providing better communication performance of the parallel matrix multiplication application. Consideration of such cases significantly increases the search space and complicates the construction of the near-optimal arrangement. We exclude such cases from the search and only test permutations of the non-split groups of rectangles in each column. Nevertheless, our heuristic can find arrangements close to the one in the right picture of Fig. 3.4.

The heuristic based on the hop-count cost function is summarized in Algorithm 1. First, in each column, we group the rectangles by subnetworks. We denote each permutation of the groups in a column i as M_i^k, $k = 1 \dots g_i!$, and the permutation with the minimum submatrix cost as M_i^*, $cost_A(M_1, \dots, M_i^*) = \min$. Let us show how the near-optimal arrangement is constructed by selecting the optimal permutations for each column. For the submatrix consisting of only one column (M_1), we have nothing to test because there are no communications in the horizontal direction. Therefore, we accept this column as the optimal permutation ($M_1^* := M_1$) and add it in the resulting arrangement.

Let us assume that we have found the optimal permutations in the first $i - 1$ columns, and hence $cost_A(M_1^*, \dots, M_{i-1}^*) = \min$. We add another column of rectangles and estimate the communication cost for the extended submatrix, trying different permutations M_i^k. The permutation with the minimal cost, M_i^*, such that $cost_A(M_1^*, \dots, M_{i-1}^*, M_i^*) = \min$, is added to the resulting arrangement. We repeat this step for all columns of rectangles. For the final arrangement, we try different permutations of the columns and find the one that further minimizes inter-cluster communications. The result of the algorithm is the near-optimal arrangement for parallel matrix multiplication.

Algorithm 1 Heuristic based on the hop-count cost function

for each column $i := 1$ to c **do**
  group rectangles by clusters $\rightarrow g_i$ groups
end for
$M_1^* := M_1$
for each column $i := 2$ to c **do**
  generate group permutations of $M_i \rightarrow M_i^1, \dots, M_i^{g_i!}$
  for each permutation $k := 1$ to $g_i!$ **do**
    find k such that $cost_A(M_1^*, \dots, M_{i-1}^*, M_i^k) = \min$
  end for
  $M_i^* := M_i^k$
end for
generate column permutations of matrix $M^* \rightarrow M^*(1), \dots, M^*(c!)$
for each permutation $j := 1$ to $c!$ **do**
  find j such that $cost_A(M^*(j)) = \min$
end for
$M^* := M^*(j)$

In total, this heuristic requires to test $g_2! + \dots + g_c! + c!$ arrangements of submatrices. This is significantly smaller than the solution space of the exhaustive search, which is equal to the product of the numbers of permutations of rectangles in each column $r_1! \times \dots \times r_c!$. In addition, this heuristic does not require to run the application or any benchmarks to compare the communication cost of the application for different arrangements. Instead, it uses information about the network topology and the application communication flow.

By minimizing the cost, this algorithm reduces the number and volume of inter-cluster communications. However, it does not guarantee finding the global minimum, and therefore, it provides only some near-optimal solution. By fixing the communication-optimal submatrices, we reduce the search space but may loose the optimal solution. M_i^*, the intermediate result of the search, may change if we rearrange groups in one of the first $i - 1$ columns, and this configuration all together may be a better solution. Nevertheless, for the small platform used in Section 3.2.2, the result of the heuristic search coincided with the communication-optimal arrangement found by the exhaustive search.

3.4.2 Heuristic Based on the Bandwidth Cost Function

The second heuristic uses the cost function based on network bandwidth and is summarized in Algorithm 2. Since this cost function is sensitive to any permutations in columns, we modify the heuristic. Similarly to the previous heuristic, first, in columns, we group rectangles assigned to the same subnetwork. Then, in the first column, we try different permutations of the clustered rectangles and add the one with minimum $cost_B$ to the resulting arrangement: M_1^*. This provides the fastest route for communications in the first column. Another modification in the heuristic is related to the search of optimal permutations of groups in other columns: instead of $cost_A$, we use the combined cost $cost(M_1^*, \ldots, M_{i-1}^*, M_i^*) \rightarrow \min$. This guarantees that while improving communications horizontally, we will not deteriorate the vertical routes. The second heuristic is concluded by the similar search for the most efficient permutation of columns.

Algorithm 2 Heuristic based on the bandwidth cost function

```
for each column i := 1 to c do
    group rectangles by clusters → g_i groups
end for
generate group permutations of M_1 → M_1^1, ..., M_1^{g_1!}
for each permutation k := 1 to g_1! do
    find k such that cost_B(M_1^k) = min
end for
M_1^* := M_1^k
for each column i := 2 to c do
    generate group permutations of M_i → M_i^1, ..., M_i^{g_i!}
    for each permutation k := 1 to g_i! do
        find k such that cost(M_1^*, ..., M_{i-1}^*, M_i^k) = min
    end for
    M_i^* := M_i^k
end for
generate column permutations of matrix M^* → M^*(1), ..., M^*(c!)
for each permutation j := 1 to c! do
    find j such that cost(M^*(j)) = min
end for
M^* := M^*(j)
```

The complexity of the second heuristic is equal to $g_1! + g_2! + \ldots + g_c! + c!$ tests of different arrangements of submatrices. This is still significantly smaller than the solution space of the exhaustive search. This heuristic can be more efficient than the previous one, especially on highly heterogeneous networks with significant number of subnetworks. Not only it reduces unnecessary exchanges between the clusters but also employs the fastest routes between them.

3.5 Experimental Results

In this section, we demonstrate that the communication performance of the heterogeneous matrix multiplication application can be significantly improved by rearranging the matrix partition with the hop-count and bandwidth heuristics. We show that the proposed heuristics provide better matrix partitions for both ring and one-to-all communication flows.

In our experiments, we used FuPerMod, a software tool for optimal data partitioning on dedicated heterogeneous HPC platforms [58]. In addition to the programming interface for balancing the computational workload in data-parallel scientific applications, this tool provides two implementations of the FPM-BR heterogeneous matrix multiplication algorithm, based on the one-to-all and ring communication flows respectively (see Section 3.2.1). We improve the communication performance of these applications on a two-level network hierarchy by rearranging the result of the FPM-BR matrix partitioning, using the proposed heuristics.

We performed experiments on the Grid'5000 infrastructure, which consists of a number of clusters distributed between 10 sites in France and connected via the Renater network. Each site hosts several clusters of identical nodes. Table 3.8 shows the specifications of the clusters used in the experiments. The interconnected clusters form a two-level hierarchy, with very heterogeneous inter-cluster links. We had a priori information about the network topology and bandwidth. We performed experiments on different subsets of Grid'5000, forming clusters of both highly heterogeneous and relatively homogeneous computing nodes.

Table 3.8: Specification of the Grid'5000 nodes used in the experiments

Cluster	Site	Processor	Cores	Memory
Edel	Grenoble	2.27 GHz Xeon	8	24GB
Genepi	Grenoble	2.5GHz Xeon	8	8GB
Suno	Sophia	2.26GHz Xeon	8	32GB
Graphene	Nancy	2.53GHz Xeon	4	16GB
Griffon	Nancy	2.5GHz Xeon	4	16GB
Granduc	Luxembourg	2GHz Xeon	8	16GB
Taurus	Lyon	2.3GHz Xeon	12	32GB
Orion	Lyon	2.3GHz Xeon	12	32GB
Chimint	Lille	2.4 GHz Xeon	8	16GB
Chinqchint	Lille	2.83 GHz Xeon	8	8GB

3.5.1 Experiments on Highly Heterogeneous Nodes

The first set of experiments was performed on six clusters with 90 nodes in total: Edel (17), Chinqchint (17) , Graphene (15), Granduc (16), Taurus (12), and Suno (13). We spawned one MPI process per node, with different numbers of threads to increase heterogeneity. The bandwidth of inter- and intra-cluster communications is shown in Table 3.9.

The heterogeneous matrix multiplication applications were configured with the block size 64 and the problem size 90,000. Fig. 3.9 shows the original topology-unaware data partitioning. Fig. 3.10 shows the arrangements obtained from the hop-count and bandwidth heuristics. Table 3.10 shows the communication cost of all arrangements, found with the hop and bandwidth cost functions, and the total execution time of the applications based on the ring and one-to-all communication flows.

The arrangement found by the hop-count heuristic reduces the total execution time of the ring and one-to-all implementations by 20% and 15% respectively. While minimizing the number of hops, the hop-count heuristic groups rectangles that belong to the same cluster. The grouping occurs in both horizontal and vertical directions. In addition, the partition obtained from the hop-count heuristic has lower bandwidth cost than the original partition.

The number of hops returned by the bandwidth heuristic is larger, and the

Table 3.9: Heterogeneous: Bandwidths of communicating links (MB/sec)

	Edel	Chinqchint	Graphene	Granduc	Taurus	Suno
Edel	891.20	59.51	55.01	44.29	249.53	83.54
Chinqchint	59.51	892.90	92.13	71.00	75.12	43.20
Graphene	55.01	92.13	892.35	313.51	68.96	39.30
Granduc	44.29	71.00	313.51	894.53	56.78	28.90
Taurus	249.53	75.12	68.96	56.78	6850.27	111.23
Suno	83.54	43.20	39.30	28.90	111.23	895.41

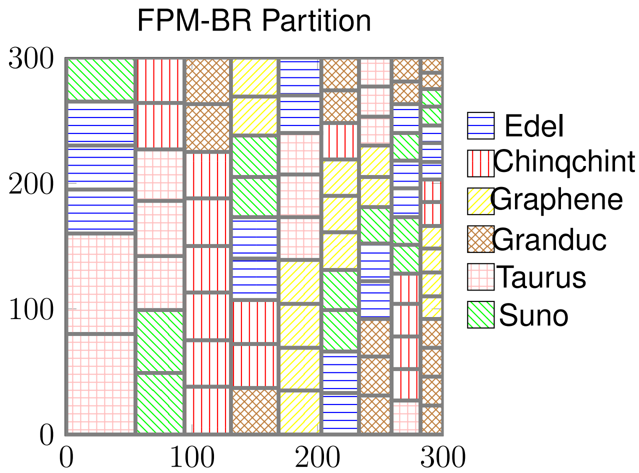

Figure 3.9: Heterogeneous: FPM-BR matrix partition for 90 nodes on 6 clusters

arrangement of rectangles does not look intuitively optimal. Nonetheless, it yields a better partition because it is aware not only of the topology but also of the performance of the network (25% and 20% improvement for the ring and one-to-all communication flows respectively). The experimental results can be interpreted as follows:

- The bandwidth heuristic favors communications between the "nearest" clusters, that is, the clusters that have the faster interconnect. Indeed, in the second arrangement, Edel and Taurus nodes communicate much more than in the first arrangement. The bandwidth between these clusters is 2-9 times higher than between any other pair of clusters.

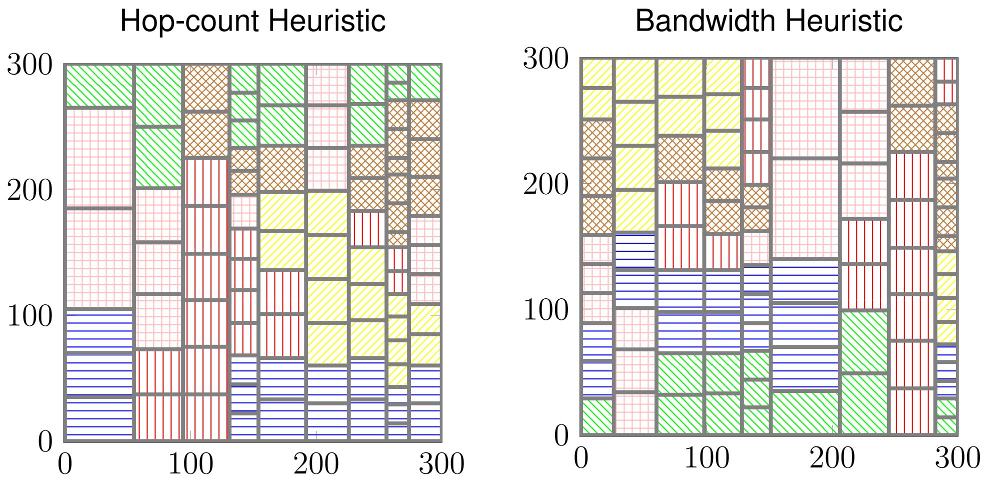

Figure 3.10: Heterogeneous: Arrangements obtained from the heuristics for 90 nodes on 6 clusters

Table 3.10: Heterogeneous: Experimental results on heterogeneous clusters

Partition	Hop cost	Bandwidth cost	Exec. time (ring)	Exec. time (one-to-all)
FPM-BR Partition	2677	29205.22	2.445955e+02	2.026377e+02
Hop-count Heuristic	1751	21695.06	2.000589e+02	1.763399e+02
Bandwidth Heuristic	1948	18840.53	1.805339e+02	1.641144e+02

- The bandwidth heuristic avoids the slow links, such as Granduc-Suno. In the second arrangement, these links are not engaged at all, whereas in the first arrangement there is significant traffic between Granduc and Suno nodes in both horizontal and vertical directions.

Being designed primarily for the matrix multiplication application based on the ring communication flow, the heuristics are equally good for the one-to-all communication flow. Indeed, rearrangement of submatrices speeds up the propagation of messages not only in rings but also in flat trees.

3.5.2 Experiments on Relatively Homogeneous Nodes

The second set of experiments was performed on relatively homogeneous nodes. We took 90 nodes of similar per-core performance from 8 clusters:

Genepi (7), Edel (11), Griffon (16), Graphene (19), Chinqchint (17), Chimint (13), Orion (4), and Taurus (3). We spawned one single-threaded MPI process per node to even the processing speeds as much as possible. The properties of the heterogeneous network are presented in Table 3.11. The applications were configured with the block size 64 and the problem size 90,000.

Table 3.11: Homogeneous: Bandwidths of communicating links (MB/sec)

	Genepi	Edel	Griffon	Graphene	Chinqchint	Chimint	Taurus	Orion
Genepi	893.14	891.02	56.72	56.72	59.51	59.51	249.53	249.53
Edel	891.02	891.20	56.72	56.72	59.51	59.51	249.53	249.53
Griffon	56.72	56.72	893.13	892.17	92.13	92.13	68.96	68.96
Graphene	56.72	56.72	892.17	892.35	92.13	92.13	68.96	68.96
Chinqchint	59.51	59.51	92.13	92.13	892.90	894.35	75.12	75.12
Chimint	59.51	59.51	92.13	92.13	894.35	896.17	75.12	75.12
Taurus	249.53	249.53	68.96	68.96	75.12	75.12	6850.27	892.17
Orion	249.53	249.53	68.96	68.96	75.12	75.12	892.17	6908.83

The algorithm minimizing the total volume of communication [7] is designed for an arbitrary number of processors, and, while arranging the matrix rectangles in columns, it may return an irregular partitioning even for relatively homogeneous processors. This was the case of our experiments (Fig. 3.11). Fig. 3.12 shows the partitions returned by the hop-count and bandwidth heuristics. This is another illustration of how the hop-count heuristic groups the rectangles by clusters and how the bandwidth heuristic rearranges based on the communication "closeness".

Table 3.12 gives the communication cost and the total execution time of the application with different arrangements of matrix rectangles. The hop-count heuristic improves performance of matrix multiplication with ring and one-to-all communications by 10% and 5% respectively; the bandwidth heuristic – by 20% and 10% respectively.

Both heuristics were less effective on relatively homogeneous nodes. For such a platform, the FPM-BR matrix partitioning algorithm returns a relatively regular matrix partition, which is characterized by a less number of overlaps in the horizontal direction (see Section 3.3.1). Consequently, both ring and

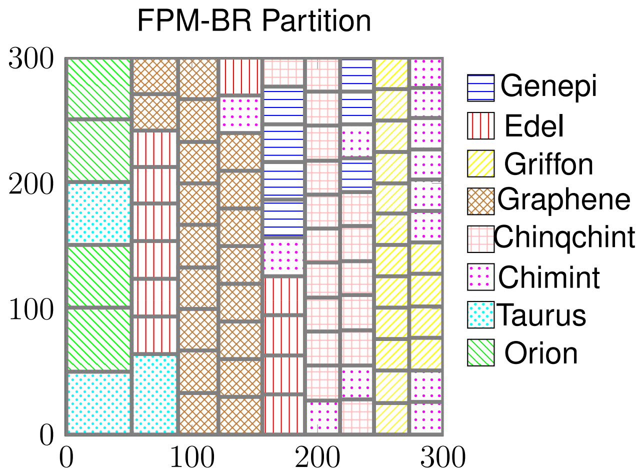

Figure 3.11: Homogeneous: FPM-BR matrix partition for 90 nodes on 8 clusters

Table 3.12: Homogeneous: Experimental results on relatively homogeneous nodes

Partition	Hop cost	Bandwidth cost	Exec.time (ring)	Exec. time (one-to-all)
FPM-BR Partition	2248	19394.13	1.963404e+02	1.655698e+02
Hop-count Heuristic	1987	17087.58	1.767677e+02	1.557851e+02
Bandwidth Heuristic	2083	12497.47	1.600700e+02	1.500844e+02

one-to-all matrix multiplication applications require less point-to-point communications and become less communication-intensive.

Another factor that reduced the effect of the heuristics on homogeneous nodes is the better, in terms of inter-cluster communications, quality of the initial FPM-BR partition. The design of the FPM-BR partitioning is such that for relatively homogeneous nodes there is a higher chance that the rectangles will be grouped by clusters in columns. This minimizes inter-cluster communications in vertical direction and may also reduce inter-cluster communications in horizontal direction. Indeed, in Fig. 3.11, matrix rectangles are better ordered than in Fig. 3.9.

Table 3.12 shows that the bandwidth heuristic provides good improvement even for relatively homogeneous nodes. This results from minimization of both the number and the performance of inter-cluster communications.

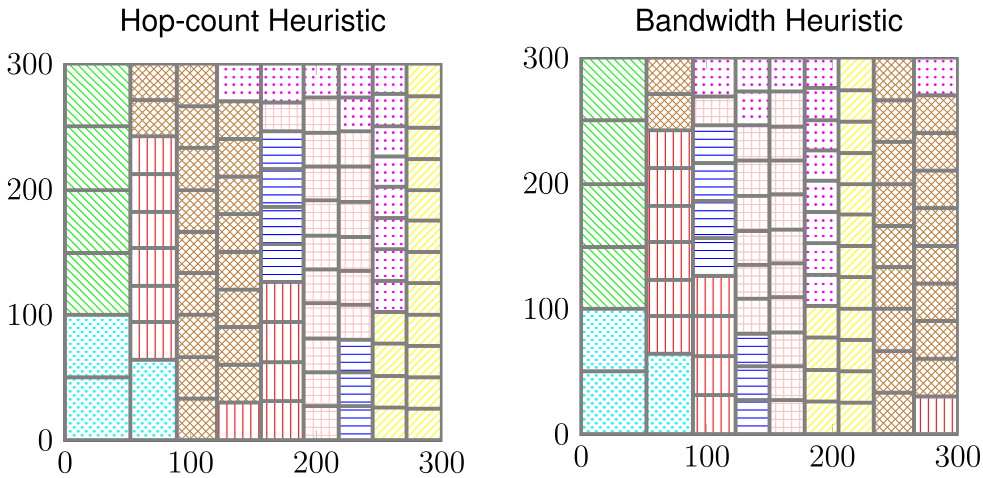

Figure 3.12: Homogeneous: Arrangements obtained from the heuristics for 90 nodes on 8 clusters

Scalability study is out of the scope of this work. First, the scalability problem is related to SUMMA on heterogeneous platforms. It has been shown that the traditional (homogeneous) scalability metrics for linear algebra can be used on heterogeneous clusters but some strategies of efficient heterogeneous distribution of computations may favor heterogeneous efficiency over scalability [103]. Second, scalability analysis will be significantly complicated by the necessity to include in consideration both different communication patterns and the hierarchy and heterogeneity of communication network. To the best of our knowledge, such a holistic approach has not been in the focus of research on scalability of parallel applications on heterogeneous platforms. The scalability analysis of the heuristics proposed in this work only makes sense if it takes both different communication patterns and the hierarchy and heterogeneity of communication network into consideration.

While we have managed to significantly reduce the search space for the optimization problem (see Section 3.2.2), the proposed heuristics still have a factorial design (see Section 3.4). The volume of computations to find the optimal arrangements depends on the number of clusters communicating in columns, which may not be related to the total number of clusters. In the presented experiments, the cost incurred by the heuristics was negligible in comparison to the execution time of the application. The heuristics do not require execution of any computation kernels of the parallel application and implemented in serial code.

3.6 Conclusion

In this chapter, we presented two heuristics aimed to minimize the communication cost of data-parallel applications using information about network topology/performance and application communication flow. This a priori information allows us to reduce significantly the search space of the optimization problem. As a test case we chose heterogeneous matrix multiplication, with irregular but deterministic communications. As a test platform we took the clusters of Grid'5000, with highly heterogeneous two-level network and good variation of processor performance. The intuition behind the proposed heuristics was that a heterogeneous data partitioning can be improved in terms of communications by grouping the parts by clusters or the speed of interconnect. We validated this approach in experiments.

Our heuristics can be applied to other data-parallel applications. In next chapter we extend this work to parallel CFD (Computational Fluid Dynamics) applications, where partitioning is typically found by minimizing the total volume of communications, ignoring the topology and network properties of underlying execution platform.

Chapter 4

Topology-aware Optimization of MPDATA on Homogeneous Multi-core Clusters with Heterogeneous Network

In this Chapter, we propose a new algorithm that is built on top of cost functions and heuristics of one of our previously proposed algorithms. This algorithm reduces overall message hops and increases data throughput for a wider range of applications, and we apply it to a real-life CFD application. We also present experimental results demonstrating performance gains due to this optimization.

4.1 Introduction

Heterogeneity appears not only in the computing devices but also in networks. Even with homogeneous processors, efficient execution of data-parallel applications is a big challenge due to ever increasing heterogeneity and complexity of the underlying networks. In this work, we consider the network heterogeneity rather than the processor heterogeneity. Thus, the target platform comprises homogeneous processors connected with a heterogeneous network. Assuming that the workload is balanced among the processors, we propose a mapping approach that optimizes the overall communication performance of a parallel computational fluid dynamics application on such a platform.

The CFD application we consider in this work is the MPDATA, which is one of the major parts of the dynamic core of the EULAG geophysical model [30], [31]. This geophysical model can be used for simulating thermo-fluid flows across a wide range of scales and physical scenarios, including the numerical weather prediction. The MPDATA belongs to the group of non-oscillatory forward-in-time algorithms, and performs a sequence of stencil computations. The original version of MPDATA has been implemented in FORTRAN 77 and parallelized using MPI library. In [32], it was proposed to rewrite the MPDATA code and replace conventional HPC systems with modern homogeneous and heterogeneous multi- and many-core based platforms. A new version of MPDATA allowed us to much better exploit the available computational features of novel processors and Intel Xeon Phi coprocessors.

However, the communication cost of MPDATA on modern HPC clusters has not been properly optimized. The current approach to mapping of the partitions of the MPDATA computational domain onto computing resources take into account neither the actual properties of the MPDATA communication flow nor the heterogeneity, hierarchy and performance of the communication network.

In this work, we first study and analyse the communication pattern of the MPDATA application. The analysis reveals that MPDATA is very sensitive to the choice of logical topology of processes as the cost per byte of horizontal communications is higher than that of vertical communications even for homogeneous communication networks. This property of MPDATA further complicates the task of partitioning of the MPDATA computational domain and mapping of the sub-domains to the processors in a way that minimizes the cost of communications between different levels of the network hierarchy. For MPDATA, we propose a new heuristic algorithm based on one of our general heuristic approach presented in Chapter 3 and apply it to optimization of the communication cost of MPDATA. This algorithm is non-intrusive to the source code of the application and, compared to previously discused algorithms, is not application specific. Our previous algorithms deal with two-dimensional symmetric communication patterns that is why we tested these algorithms in the context of the parallel matrix multiplication application. With this new algorithm, any data-parallel application with two-dimensional homogeneous computational domain and asymmetric heterogeneous communication pattern can benefit. We demonstrate the accuracy and efficiency of the proposed solution using experiments on two-level hierarchical networks, namely, interconnected nodes (intra- and inter-node communication levels) and interconnected clusters (intra- and inter-cluster communication levels).

4.2 MPDATA

The MPDATA application is used to solve the advection equation on a moving grid according to the subsequent time steps [104], [105]. This real-life application offers several advanced options that allow for modeling a wide range of complex geophysical flows. Depending on the type of modeled phenomena, this application can demand a high computing performance of HPC clusters. Therefore, the configurable code of MPDATA was developed and delivered over the years [30], [104], [106]. This code was implemented in FORTRAN 77 and parallelized using MPI library, however, without taking into account of the features of today‘s computing architectures.

The MPI parallelization of the MPDATA computations on x86-based clusters as a part of the EULAG model was thoroughly studied in [106], [107], using tens of thousands of cores, or even more than 100K cores in the case of IBM Blue Gene/Q. The parallelization strategy of this implementation is based on 3D domain decomposition, and executes computations according to the distributed memory model where each core is assigned to a single MPI_rank. This approach ignores the advantages of shared memory systems available in modern multi core platforms. Moreover, it also does not take into account the network-aware partitioning of communications across computing resources.

The MPDATA code has been recently re-written and optimized for execution on modern CPU and Intel co-processors based high performance computing platforms. The new C++ implementation proposed in [32], [108], [109] allows for more efficient distribution of computational tasks on the available resources. It makes use of the (3+1)D decomposition strategy for the stencils computation, that transfers the data traffic from the main memory to cache hierarchy by proper reusing of the cache memory. Additionally, to improve the computational efficiency the algorithm groups the cores (threads) into independent work teams in order to reduce inter-cache communication overheads due to the communications between neighbouring threads/cores, and synchronizations.

4.3 MPDATA on Clusters

One of the common methods for exploiting the multi core clusters is to employ the hybrid programming model, that allows for efficient usage of the distributed and shared memory hierarchies of these systems. This implies to combine different programming paradigms, such as MPI and OpenMP. Such a mixture is successfully utilized for the MPDATA computation, where a single MPI_rank is assigned to every multi core node while OpenMP threads are employed to utilize the multi core computational resources.

The 3D $n \times m \times l$ MPDATA domain is firstly partitioned in two dimensions n and m into equal sub-domains that are further one-to-one mapped to adequate nodes of the homogeneous clusters. Every sub-domain of size $nB \times mB \times l$ is decomposed according to the (3+1)D decomposition proposed in [32]. This strategy contributes to ease the main-memory and communications bounds, that characterize MPDATA, and to better exploit modern computational resources such as cores and vector units.

Since the (3+1)D strategy allows for independent calculation of every sub-domain for a single time step, the inter-node communications and synchronization points have to take place only between subsequent time steps in order to exchange the required partial outcomes. The exchanged data corresponds to the halo regions determined by data dependencies of

MPDATA computations. These regions take place on the border of the MPDATA domain partitioning. As a result, the data traffic is generated only between nodes that are mapped onto adjacent sub-domains in both directions: vertical and horizontal. Figure 4.1 illustrates the data flow between nodes of MPDATA application.

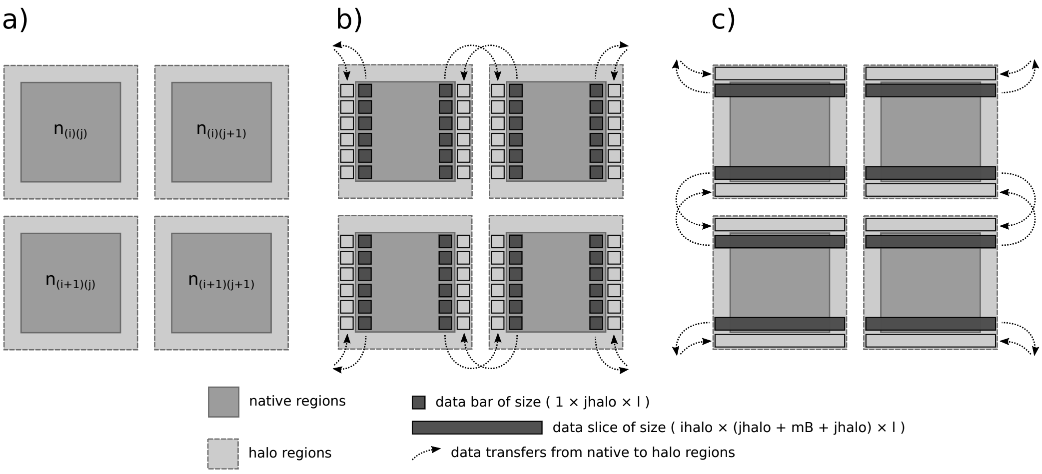

Figure 4.1: Data flow between nodes for the MPDATA application: a) 2D domain decomposition between computing nodes: $n_{ij}, n_{ij+1}, ...$, b) the communication pattern for the horizontal direction, c) the communication pattern for the vertical direction

After every time step each node has to send/receive in horizontal direction the adequate halo regions to/from adjacent nodes placed on the left and right sides (Figure 4.1b). Since the necessary halo regions for this direction are periodically placed in the main memory, each node exchanges nB data bar of size $1 \times jhalo \times l$ to the left node, and to the right one. Then, the same node is responsible for sending/receiving in vertical direction the adequate halo regions to/from adjacent nodes placed on the top and bottom sides (Figure 4.1c). Transferred data in this communication path is placed in the contiguous memory areas, thus this node moves the data slices of size $ihalo \times (jhalo + mB + jhalo) \times l$ to/from the top and bottom nodes.

4.4 Communication Optimal Mapping Arrangement for MPDATA

In this section, first we propose an extension of the network-bandwidth-based cost function presented in Chapter 3 to accurately measure the communication cost of the MPDATA application. Then we formulate the heuristic solution that efficiently constructs a near-optimal arrangement for MPDATA based on the extended cost function by using information about network topology and the application communication flow. This heuristic solution reduces the search space of sub-domain arrangements and finds the one that minimizes the communication cost of the MPDATA.

4.4.1 Cost Function Based on Asymmetric Bandwidth

In previous Chapter, we defined the cost function based on network bandwidth. The main idea was to estimate the communication cost accurately by using information about the network topology and the application communication flow. That cost function proved to work well with applications having symmetric communication patterns. However, MPDATA has asymmetric communication behaviour, namely, even in the case of a homogeneous communication layer the effective bandwidth of horizontal communications is higher than that of the vertical ones. One of the reasons behind this phenomenon is that data communicated vertically is stored in a contiguous region of memory while the data communicated horizontally is not. As a result, this cost function fails to accurately characterize the communication cost of MPDATA.

Therefore, we propose to extend this bandwidth-based cost function to account for applications with asymmetric communication patterns. The proposed extension characterizes the communication time, using the asymmetric bandwidths properties. We call it a cost function based on asymmetric bandwidth in the rest of the Chapter. The function takes into account two bandwidth values, one for horizontal communication and the other is for vertical one. The problem of finding the communication-optimal arrangement can be formulated as minimization of the sum of the horizontal and vertical communication costs.

Assuming that the data is equally partitioned among the processors, so that the size of each sub-domain is same, we define the asymmetric cost function for horizontal communication as follows:

$$cost_H = \sum_{i=1}^{r} \left(h \times \sum_{j=1}^{c} \frac{1}{b_H(Q_{ij}, Q_{i,(j+1)\%c})} \right), \tag{4.1}$$

where i iterates over the rows and j iterates over the partitioned sub-domains in each row. h is the height of a row (in bytes) that is same for each row because data is equally partitioned. Function $b_H(X, Y)$ returns the horizontal bandwidth (in bytes per second) between processors X and Y, and Q_{ij} designates the processor holding the j-th sub-domain in row i. Thus, this cost function estimates the communication time in seconds. The inner sum represents sending a part of the pivot column in a row. The outer sum represents the upper bound on the communication time required to send the whole pivot column to all rows. We use the upper bound because the bandwidth of some links may be divided between multiple communications corresponding to different rows.

We define the asymmetric cost function for vertical communication in a similar way:

$$cost_V = \sum_{j=1}^{c} \left(w \times \sum_{i=1}^{r} \frac{1}{b_V(Q_{ij}, Q_{i,(j+1)\%r})} \right), \tag{4.2}$$

Here j iterates over the columns, and i iterates over the partitioned sub-domains in each column. w is the width of a column (in bytes) that is same for each column because data is equally partitioned. Function $b_V(X, Y)$ returns the vertical bandwidth (in bytes per second) between processors X and Y.

The communication cost associated with arrangement A is represented by two values $(cost_H(A), cost_V(A))$. The problem of finding the communication-optimal arrangement can be formulated as minimization of their sum:

$$cost_H(A) + cost_V(A) \to \min . \tag{4.3}$$

4.4.2 Heuristic Based on Asymmetric Bandwidth Cost Function

The heuristic algorithm using the asymmetric bandwidth cost function for estimating the volume of communications is built on top of the bandwidth-based heuristic presented in Chapter 3. It assumes that the target platform consists of p interconnected homogeneous processors. The processors are naturally partitioned into a number of groups based on their communication proximity, which reflects the two-level hierarchy of the communication layer. If processors x_0, x_1, y_0 and y_1 belong to the same group then $b_H(x_0, y_0) = b_H(x_1, y_1)$ and $b_V(x_0, y_0) = b_V(x_1, y_1)$. The heuristic based on the asymmetric bandwidth cost function is summarized in Algorithm 3.

The algorithm starts with any initial arrangement $P_1, P_2, \ldots, P_p$ of the processors such that processors from the same group will follow one other in this linear arrangement. Note, the orders naturally determined by application configuration files typically satisfy this assumption. Alternatively, a simple clustering algorithm guided by functions $b_H(x, y)$ and $b_V(x, y)$ can be applied to re-order the original arrangement if it does not satisfy this assumption.

The algorithm then repeatedly executes the following two steps. The first step finds the optimal two-dimensional arrangement of the processors, $m \times n$, which preserves their linear order as follows. For each factor pair $r \times c = p$, the processors are arranged column-wise and row-wise into r rows and c columns forming arrangement A. The cost of these arrangements are estimated as $cost(P_1, \ldots, P_p, r, c) = cost_H(A) + cost_V(A)$, and the optimal pair $m \times n$ is found as the one that minimizes this cost, $cost(P_1, \ldots, P_p, m, n) = \min_{r,c} cost(P_1, \ldots, P_p, r, c)$.

The second step applies the bandwidth-based algorithm from Chapter 3 slightly modified by the use of the asymmetric cost function to this 2D

Algorithm 3 Heuristic based on the asymmetric bandwidth cost function

Input:
Processors, $P_1, P_2, \dots, P_p \in \mathbb{Z}_{>0}$
Horizontal bandwidth, $b_H(x, y), \forall x, y \in [1, p], b_H(x, y) \in \mathbb{Z}_{>0}$
Vertical bandwidth, $b_V(x, y), \forall x, y \in [1, p], b_V(x, y) \in \mathbb{Z}_{>0}$
Output:
Optimal 2-D arrangement of the processors

Repeat
STEP 1:
for each factor pair $r \times c = p$ **do**
  arrange $P_1, ..., P_p$ in r and c by row ranking order $\rightarrow A$
  arrange $P_1, ..., P_p$ in r and c by column ranking order $\rightarrow A$
  find A such that $cost(A, m, n) = \min\limits_{r,c} cost(A, r, c)$
end for
STEP 2:
generate group permutations of $A_1 \rightarrow A_1^1, \dots, A_1^{g_1!}$
for each permutation $k := 1$ to $g_1!$ **do**
  find k such that $cost_V(A_1^k) = \min$
end for
$A_1^* := A_1^k$
for each column $i := 2$ to n **do**
  generate group permutations of $A_i \rightarrow A_i^1, \dots, A_i^{g_i!}$
  for each permutation $k := 1$ to $g_i!$ **do**
    find k such that $cost(A_1^*, \dots, A_{i-1}^*, A_i^k) = \min$
  end for
  $A_i^* := A_i^k$
end for
generate column permutations of arrangement $A^* \rightarrow A^*(1), \dots, A^*(n!)$
for each permutation $j := 1$ to $n!$ **do**
  find j such that t $cost(A^*(j)) = \min$
end for
$A^* := A^*(j)$

arrangement. This step may changes the linear order of the processors within the arrangement in order to reduce its communication cost while preserving the shape of the arrangement, $m \times n$. The reordering is guided by the 2D partitioning of the computational domain induced by the 2D processor arrangement and uses the fact that within each column of the domain, sub-domains held by processors from the same group will also make a group of adjacent sub-domains. In brief, we first try permutations of the groups in the first column and pick the one that minimizes the vertical communication cost for this column. We denote each permutation of the groups in a column i as A_i^k, $k = 1 \dots g_i!$, and the permutation with the minimum sub-arrangement cost as A_i^*, $cost(A_1, \dots, A_i^*) = \min$. We accept this column as the optimal permutation ($A_1^* := A_1$) and add it in the resulting arrangement. Then, for each following column $i = 2, \dots, n$, we try permutations of the groups in this column A_i^k and pick the one that minimizes the sum of vertical and horizontal costs for first i columns. This guarantees that while improving communications horizontally, we will not deteriorate the vertical routes. Permutation of groups rather than individual processors in a column will significantly reduce the solution space that otherwise would be $p!$. Finally, we try all permutations of whole columns and pick the one that minimizes the sum of horizontal and vertical communication costs for the whole domain.

This step can change our original linear arrangement of the processors. If this is the case, we will feed the new arrangement to the first step of next iteration of our heuristic algorithm that will find the optimal $m \times n$ arrangement for this new order. Then, this 2D arrangement will be re-arranged by the second step of this iteration. This procedure continues until we find a fixed point of the transformation performed by one iteration of the algorithm.

The presented iterative algorithm does not require to run the application or any benchmarks to compare the communication cost of the application for different arrangements. Instead, it uses information about the network topology and the application communication flow. This heuristic is efficient for applications having 2D communication pattern on heterogeneous networks. Not only it reduces unnecessary exchanges between the sub-networks but also employs the fastest routes between them.

4.5 Experimental Results

In this section, we demonstrate that the communication performance of MPDATA can be significantly improved due to optimization proposed by the asymmetric bandwidth heuristic not only for heterogeneous but also for a perfectly homogeneous communication network.

We perform experiments on the Grid'5000 infrastructure, which is a large scale distributed platform. It consists of a number of clusters distributed between 10 sites in France and connected via the Renater network. Each site hosts several clusters of identical nodes. For our experiments, we choose two clusters, Grisou and Grimoire, from the Nancy site and the other two, Paravance and Parasilo, from the Rennes site. All clusters have identical Intel Xeon E5-2630 v3 processors with 8 cores per node. To demonstrate performance gains, we first perform two types of experiments on interconnected clusters. These interconnected clusters form a two-level hierarchy, with very heterogeneous inter-cluster links. Then, we conduct experiments on a single fully homogeneous cluster, with homogeneous processors and a homogeneous communication network. We have a priori information about the network topology and asymmetric bandwidths of MPDATA. We have tried ten different initial mappings as an input and our experiment shows that all of these mappings converges to the optimal solutions have same communication cost after applying asymmetric bandwidth heuristic. It has been noted that there is more than one optimal solutions exist. However, the communication cost and execution time of all optimal solutions are same. To make sure the experimental results are reliable, the application is repeatedly executed until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved and results follows the normal distribution. We also make sure the nodes are fully reserved and dedicated to our experiments.

4.5.1 Inter-Cluster experiments

In these experiments, we use four clusters with 12 nodes in total: Grimoire(3), Parasilo(4), Grisou(2), Paravance(3). We spawn one MPI process per node. Because logical communication links of MPDATA has different bandwidths, we have two bandwidth values for each link. Horizontal and vertical bandwidths are shown in Table 4.1. MPDATA is configured with problem size $512 \times 512 \times 64$.

Table 4.1: Horizontal/Vertical bandwidths of communicating links(GB/sec)

	Grimoire	Parasilo	Grisou	Paravance
Grimoire	0.03963/0.48068	0.00007/0.00056	0.03889/0.49341	0.00007/0.00056
Parasilo	0.00007/0.00056	0.03876/0.48858	0.00007/0.00056	0.03732/0.45943
Grisou	0.03889/0.49341	0.00007/0.00056	0.03834/0.48916	0.00007/0.00056
Paravance	0.00007/0.00056	0.03732/0.45943	0.00007/0.00056	0.03920/0.46808

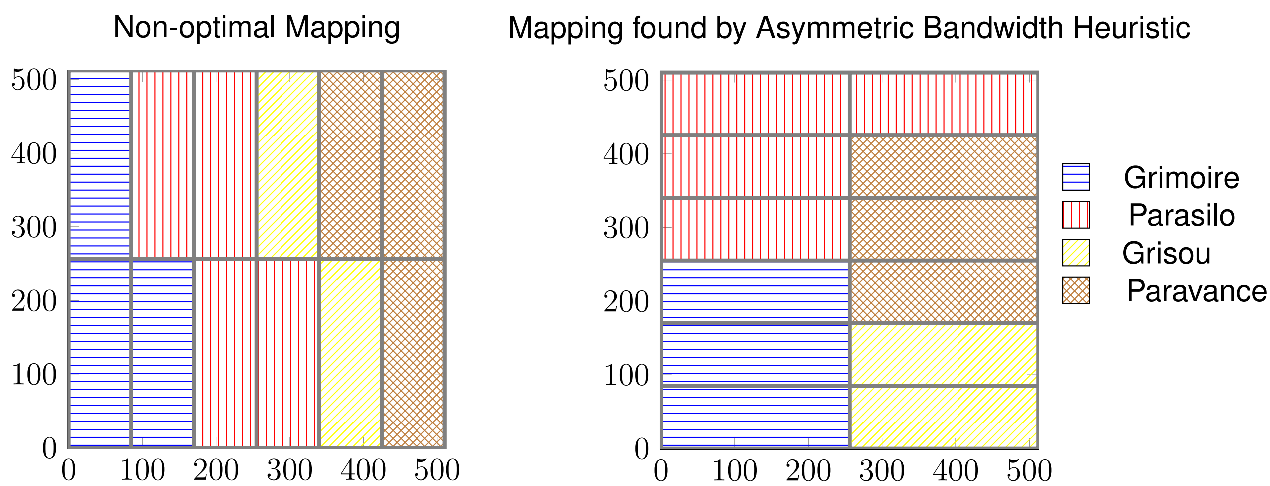

Figure 4.2: One of the non-optimal mappings and the mapping returned by the asymmetric bandwidth heuristic for the heterogeneous platform.

Fig. 4.2 shows one of the considered default initial mappings and the optimal mapping found by the asymmetric bandwidth heuristic. Table 4.2 shows the communication cost of these mappings, calculated using the cost function, and the measured total execution time of MPDATA. To find the

Table 4.2: inter-cluster experimental results

Nodes	Cost		Ratio	Exec. time (sec)		Ratio
	Non-optimal	Heuristic		Non-optimal	Heuristic	
12	22424946	2143978	10.46	994.02	154.20	6.44

optimal mapping, the asymmetric bandwidth heuristic took 1.130000e-03 sec. The mapping found by the asymmetric bandwidth heuristic is more then 6 times faster then the non-optimal case mapping.

4.5.2 Intra-Cluster Experiments

We also perform experiments on a homogeneous multi-core cluster to check the effect of asymmetric bandwidth of MPDATA on the communication performance with a perfectly homogeneous network. We use 12 nodes from the Grisou cluster. MPDATA is configured with problem size $512 \times 512 \times 64$.

Non-optimal Mapping

P_1	P_3	P_5	P_7	P_9	P_{11}
P_2	P_4	P_6	P_8	P_{10}	P_{12}

Mapping found by Asymmetric bandwidth Heuristic

P_1	P_7
P_2	P_8
P_3	P_9
P_4	P_{10}
P_5	P_{11}
P_6	P_{12}

Figure 4.3: One of the non-optimal mappings and the mapping returned by the asymmetric bandwidth heuristic for the fully homogeneous platform.

Fig. 4.3 shows one of the non-optimal mappings and the mapping returned by the asymmetric bandwidth heuristic. Table 4.3 shows the calculated communication cost of both mappings and the measured total execution time of MPDATA. The mapping found by the asymmetric bandwidth

Table 4.3: intra-cluster experimental results

Nodes	Cost		Ratio	Exec time (sec)		Ratio
	Non-optimal	Heuristic		Non-optimal	Heuristic	
12	65658	18535	3.5	3.86	1.32	3.0

heuristic is 3 times faster then the non-optimal mapping. Asymmetric bandwidth heuristic took 3.730000e-04 sec to find this optimal mapping.

Chapter 5

Conclusion and Future Work

This thesis focused on the problem of communication optimization of parallel scientific applications for execution on heterogeneous HPC platforms. We addressed this problem by performing topology-aware communication optimization. We presented heuristic based algorithms that took into account the application communication pattern and underlying network topology; hence, result in communication optimal mapping.

Based on topology and performance information, communications on hierarchical heterogeneous HPC platforms can be optimized. For MPI, a major programming tool for such platforms, a number of topology- and performance-aware implementations of collective operations have been proposed for optimal scheduling of messages. These approaches improve performance of application and do not require to modify application source code. However, they are applicable to collective operations only and do not affect the parts of the application that are based on point-to-point exchanges. We addressed the problem of efficient execution of point-to-point based data-parallel applications on interconnected clusters and proposed a general approach and two approximate heuristic algorithms aimed at minimization of the communication cost of data parallel applications which have two-dimensional symmetric communication pattern on heterogeneous hierarchical networks, and tested these algorithms in the context of the parallel matrix multiplication application. We also demonstrated the correctness and efficiency of the proposed approaches by experimental results on multi-core nodes and interconnected heterogeneous clusters.

The communication layer of modern HPC platforms is becoming increasingly heterogeneous and hierarchical. As a result, even on platforms with homogeneous processors, the communication cost of many parallel applications will significantly vary depending on the mapping of their processes to the processors of the platform. For applications having asymmetric communication pattern, we proposed a new algorithm that was based on cost functions of one of our general heuristic algorithms and applied it for optimization of the communication cost of MPDATA, which has asymmetric heterogeneous communication pattern. We also presented experimental results demonstrating performance gain due to this optimization. Furthermore, our experiments on Grid5000 involved significant efforts for resolving technical issues. These experiments were performed on multi-sites which lead to many synchronization issues for software‘s, application and data. SimGrid simulation experiments, presented in appendix, also explain our efforts for running the experiments on simulated platform. Here, we also discussed the possible problems faced and their causes and also identified the factors that influenced the realistic measurement of execution time on SMPI. The results presented in Chapter 3 have been published in [110] and [111], and the results presented in Chapter 4 have been published in [112].

We have seen promising results from these topology-aware algorithms, and see further opportunities in this area. These algorithms can be scaled to other complex scientific data-parallel applications. We can modify the proposed algorithms in a number of ways. Despite our study focuses on two-level hierarchical optimization, the algorithms can be applied in a multi-level hierarchical way. Asymmetric Bandwidth based algorithm initially worked with homogeneous multi-core clusters with heterogeneous network, but can be modified also for heterogeneous multi-core clusters.

With the advancement of multi-level hierarchical heterogeneous platforms, the requirement for topology-aware mapping will become stronger and challenging. Mapping approaches, which take into account the application communication flow and platform topology, result in efficient execution of application at reduced communication cost.

Bibliography

[1] Lastovetsky A. and Dongarra J., *High Performance Heterogeneous Computing*. Wiley, 2009, p. 267 (cit. on pp. 1, 13, 14).

[2] H. J. Siegel, L. Wang, V. P. Roychowdhury, and M. Tan, “Computing with heterogeneous parallel machines: Advantages and challenges,” in *Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on*, 1996, pp. 368–374. DOI: `10.1109/ISPAN.1996.509012` (cit. on p. 1).

[3] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, “State-of-the-art in heterogeneous computing,” *Sci. Program.*, vol. 18, no. 1, pp. 1–33, Jan. 2010, ISSN: 1058-9244 (cit. on p. 1).

[4] A. Plaza, J. Plaza, and D. Valencia, “Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data,” *The Journal of Supercomputing*, vol. 40, no. 1, pp. 81–107, 2007, ISSN: 1573-0484 (cit. on pp. 1, 14).

[5] *Top 500 supercomputer sites*. [Online]. Available: `http://www.top500.org/` (cit. on p. 1).

[6] J. Shalf, S. Dosanjh, and J. Morrison, in (cit. on p. 2).

[7] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, “Matrix multiplication on heterogeneous platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1033–1051, 10 2001, ISSN: 1045-9219 (cit. on pp. 2, 7, 8, 15, 25, 47).

[8] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, "Heterogeneous matrix-matrix multiplication or partitioning a square into rectangles: Np-completeness and approximation algorithms," in *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on*, IEEE, 2001, pp. 298–305 (cit. on p. 2).

[9] D. Clarke, A. Lastovetsky, and V. Rychkov, "Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models," in *Euro-Par 2011: Parallel Processing Workshops*, ser. LNCS, vol. 7155, Springer, 2012, pp. 450–459, ISBN: 978-3-642-29736-6 (cit. on pp. 2, 7, 8, 16, 25, 27).

[10] K. Dichev and A. Lastovetsky, "Optimization of collective communication for heterogeneous hpc platforms," in, Wiley-Interscience, 2013 (cit. on pp. 2, 16).

[11] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, "Effects of communication latency, overhead, and bandwidth in a cluster architecture," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 25, 1997, pp. 85–97 (cit. on p. 2).

[12] T. Agarwal, A. Sharma, A. Laxmikant, and L. Kale, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *IPDPS 2006*, 2006, p. 10 (cit. on pp. 2, 3, 19).

[13] E. Solomonik, A. Bhatele, and J. Demmel, "Improving communication performance in dense linear algebra via topology aware collectives," in *SC '11*, Seattle, Washington: ACM, 2011, 77:1–77:11, ISBN: 978-1-4503-0771-0 (cit. on pp. 2, 18).

[14] T. Kielmann, R. F. Hofman, H. E. Bal, A. Plaat, and R. A. Bhoedjang, "Magpie: MPI's collective communication operations for clustered wide area systems," in *ACM Sigplan Notices*, ACM, vol. 34, 1999, pp. 131–140 (cit. on pp. 3, 18).

[15] N. Karonis, B. De Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *IPDPS 2000*, 2000, pp. 377–384 (cit. on pp. 3, 18).

[16] T. Ma, G. Bosilca, A. Bouteiller, and J. Dongarra, "Hierknem: An adaptive framework for kernel-assisted and topology-aware collective communications on many-core clusters," in *IPDPS 2012*, 2012, pp. 970–982 (cit. on pp. 3, 18).

[17] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, "Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010, pp. 1–8. DOI: 10.1109/IPDPSW.2010.5470853 (cit. on p. 3).

[18] J. Traff, "Implementing the MPI process topology mechanism," in *Supercomputing 2002*, 2002, pp. 28–28 (cit. on pp. 3, 19).

[19] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, "Mpipp: An automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06, Cairns, Queensland, Australia: ACM, 2006, pp. 353–360, ISBN: 1-59593-282-8 (cit. on pp. 3, 19).

[20] R. C. Agarwal, F. G. Gustavson, and M. Zubair, "A high-performance matrix-multiplication algorithm on a distributed-memory parallel computer, using overlapped communication," *IBM Journal of Research and Development*, vol. 38, no. 6, pp. 673–681, 1994 (cit. on p. 7).

[21] R. A. Van De Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency-Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997 (cit. on p. 7).

[22] J. Choi, J. J. Dongarra, and D. W. Walker, “Parallel matrix transpose algorithms on distributed memory concurrent computers,” *Parallel Computing*, vol. 21, no. 9, pp. 1387–1405, 1995 (cit. on p. 7).

[23] B. A. Becker and A. Lastovetsky, “Matrix multiplication on two interconnected processors,” in *Cluster Computing, 2006 IEEE International Conference on*, IEEE, 2006, pp. 1–9 (cit. on p. 7).

[24] L. S. Blackford, J. Choi, A Cleary, E D’Azevedo, J. Demmel, I Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, *et al.*, “Scalapack: A linear algebra library for message-passing computers,” in *In SIAM Conference on Parallel Processing*, Citeseer, 1997 (cit. on p. 7).

[25] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing*. Benjamin/Cummings Redwood City, 1994, vol. 110 (cit. on p. 7).

[26] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, “Fupermod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous hpc platforms,” in *Parallel Computing Technologies: 12th International Conference, PaCT 2013, St. Petersburg, Russia, September 30 - October 4, 2013. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 182–196, ISBN: 978-3-642-39958-9 (cit. on pp. 7, 13).

[27] R. A. Van De Geijn and J. Watts, “Summa: Scalable universal matrix multiplication algorithm,” *Concurrency-Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997 (cit. on pp. 7, 24).

[28] *Simgrid- versatile simulation of distributed systems*. [Online]. Available: `http://simgrid.gforge.inria.fr/` (cit. on p. 9).

[29] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014 (cit. on pp. 9, 21).

[30] R. Wyrzykowski, L. Szustak, and K. Rojek, “Parallelization of 2d mpdata eulag algorithm on hybrid architectures with gpu accelerators,” *Parallel Computing*, vol. 40, pp. 425–447, 2014 (cit. on pp. 9, 19, 52, 53).

[31] R. Wyrzykowski, L. Szustak, K. Rojek, and A. Tomas, “Towards efficient decomposition and parallelization of mpdata on hybrid cpu-gpu cluster,” *Lect. Notes in Comp. Sci.*, vol. 8353, pp. 434–444, 2014 (cit. on pp. 9, 52).

[32] L. Szustak, K. Rojek, R. Wyrzykowski, and P Gepner, “Toward efficient distribution of mpdata stencil computation on intel mic architecture,” in *Proceedings of the 1st International Workshop on High-Performance Stencil Computations*, 2014, pp. 51–56 (cit. on pp. 10, 52, 54).

[33] H. Casanova, A. Legrand, and Y. Robert, *Parallel Algorithms*, 1st. Chapman & Hall/CRC, 2008, ISBN: 9781584889458 (cit. on pp. 10, 14).

[34] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, “Heterogeneous matrix-matrix multiplication or partitioning a square into rectangles: Np-completeness and approximation algorithms,” in *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on*, 2001, pp. 298–305. DOI: `10.1109/EMPDP.2001.905056` (cit. on pp. 10, 14, 23).

[35] V. Boudet, F. Rastello, and Y. Robert, “Algorithmic issues for (distributed) heterogeneous computing platforms,” In Rajkumar Buyya, Toni Cortes, editors, Cluster Computing Technologies, Environments, and Applications (CC-TEA’99). CSREA, Tech. Rep., 1999 (cit. on p. 13).

[36] K. Hasanov, J.-N. Quintin, and A. Lastovetsky, “Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms,” *The Journal of Supercomputing*, vol. 71, no. 11, pp. 3991–4014, 2015, ISSN: 1573-0484 (cit. on p. 13).

[37] A. Plaza, J. Plaza, and H. Vegas, “Improving the performance of hyperspectral image and signal processing algorithms using parallel, distributed and specialized hardware-based systems,” *Journal of Signal Processing Systems*, vol. 61, no. 3, pp. 293–315, 2010, ISSN: 1939-8115 (cit. on p. 13).

[38] F. Sierra-Pajuelo, A. Paz-Gallardo, and A. Plaza, “Perfomance optimizations for an automatic target generation process in hyperspectral analysis,” in *Architecture of Computing Systems. Proceedings, ARCS 2015 - The 28th International Conference on*, 2015, pp. 1–6 (cit. on p. 13).

[39] L.-C. Canon and E. Jeannot, “Wrekavoc: A tool for emulating heterogeneity,” *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 129, 2006. DOI: `http://doi.ieeecomputersociety.org/10.1109/IPDPS.2006.1639386` (cit. on p. 13).

[40] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, “Starpu: A unified platform for task scheduling on heterogeneous multicore architectures,” *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011, ISSN: 1532-0626 (cit. on p. 13).

[41] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, *et al.*, “The opportunities and challenges of exascale computing,” *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pp. 1–77, 2010 (cit. on p. 13).

[42] M. J. Zaki, W. Li, and M. Cierniak, “Performance impact of processor and memory heterogeneity in a network of machines,” in *In 4th Heterogeneous Computing Wkshp, also TR574*, 1995 (cit. on p. 14).

[43] Z. Zhong, V. Rychkov, and A. Lastovetsky, “Data partitioning on heterogeneous multicore platforms,” in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 580–584. DOI: `10.1109/CLUSTER.2011.64` (cit. on pp. 15, 16).

[44] D. Clarke, A. Ilic, A. Lastovetsky, V. Rychkov, L. Sousa, and Z. Zhong, "Design and optimization of scientific applications for highly heterogeneous and hierarchical hpc platforms using functional computation performance models," in *High-Performance Computing on Complex Environments*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2014, ch. 13, pp. 235–260 (cit. on p. 15).

[45] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore and multi-gpu systems using functional performance models of data-parallel applications," in *2012 IEEE International Conference on Cluster Computing*, 2012, pp. 191–199. DOI: 10.1109/CLUSTER.2012.34 (cit. on pp. 15, 16).

[46] C. K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 45–55 (cit. on p. 15).

[47] A. Lastovetsky and R. Reddy, "Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models," in *Euro-Par 2009, August 25-28, 2009*. 2010 (cit. on p. 15).

[48] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *International Journal of High Performance Computing Applications*, vol. 21, pp. 76–90, 2007. DOI: 10.1177/1094342006074864 (cit. on pp. 15, 16, 26, 27).

[49] A. Ilic and L. Sousa, "On realistic divisible load scheduling in highly heterogeneous distributed systems," in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2012, pp. 426–433. DOI: 10.1109/PDP.2012.56 (cit. on p. 15).

[50] J.-N. Quintin and F. Wagner, “Hierarchical work-stealing,” in *Euro-Par 2010 - Parallel Processing: 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 217–229, ISBN: 978-3-642-15277-1 (cit. on p. 15).

[51] I. Galindo, F. Almeida, and J. M. Badía-Contelles, “Dynamic load balancing on dedicated heterogeneous systems,” in *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Dublin, Ireland: Springer-Verlag, 2008, pp. 64–74, ISBN: 978-3-540-87474-4 (cit. on p. 15).

[52] Y. Zhang, H. Kameda, and S. L. Hung, “Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems,” *IEE Proceedings - Computers and Digital Techniques*, vol. 144, no. 2, pp. 100–106, 1997, ISSN: 1350-2387. DOI: `10.1049/ip-cdt:19970951` (cit. on p. 15).

[53] F. Song, S. Tomov, and J. Dongarra, “Enabling and scaling matrix computations on heterogeneous multi-core and multi-gpu systems,” in *Proceedings of the 26th ACM International Conference on Supercomputing*, ser. ICS '12, San Servolo Island, Venice, Italy: ACM, 2012, pp. 365–376, ISBN: 978-1-4503-1316-2 (cit. on p. 15).

[54] A. Kolinov and A. Lastovetsky, “Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers,” in *High-Performance Computing and Networking: 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12–14, 1999 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 189–200, ISBN: 978-3-540-48933-7 (cit. on pp. 15, 24).

[55] A. Lastovetsky and R. Reddy, “A novel algorithm of optimal matrix partitioning for parallel dense factorization on heterogeneous processors,” in *Parallel Computing Technologies: 9th International Conference, PaCT 2007, Pereslavl-Zalessky, Russia, September 3-7,*

2007. Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 261–275, ISBN: 978-3-540-73940-1 (cit. on p. 15).

[56] L. Zhuo and V. K. Prasanna, "Optimizing matrix multiplication on heterogeneous reconfigurable systems.," in *PARCO*, Citeseer, vol. 7, 2007, pp. 561–568 (cit. on p. 15).

[57] A. Lastovetsky and J. Twamley, "Towards a realistic performance model for networks of heterogeneous computers," in *High Performance Computational Science and Engineering: IFIP TC5 Workshop on High Performance Computational Science and Engineering (HPCSE), World Computer Congress, August 22–27, 2004, Toulouse, France*. Boston, MA: Springer US, 2005, pp. 39–57, ISBN: 978-0-387-24049-7 (cit. on p. 16).

[58] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "Fupermod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous HPC platforms," in *Parallel Computing Technologies*, ser. LNCS, vol. 7979, Springe, 2013, pp. 182–196, ISBN: 978-3-642-39957-2 (cit. on pp. 16, 43).

[59] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19 - 22, 2004. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 97–104, ISBN: 978-3-540-30218-6 (cit. on p. 16).

[60] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd. (Revised). Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262692155 (cit. on p. 16).

[61] B. Barker, "Message passing interface (mpi)," in *Workshop: High Performance Computing on Stampede*, 2015 (cit. on p. 16).

[62] Y. Gong, B. He, and J. Zhong, "An overview of cmpi: Network performance aware mpi in the cloud," in *ACM SIGPLAN Notices*, ACM, vol. 47, 2012, pp. 297–298 (cit. on p. 17).

[63] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra, "Performance analysis of mpi collective operations," English, *Cluster Computing*, vol. 10, no. 2, pp. 127–143, 2007, ISSN: 1386-7857 (cit. on p. 17).

[64] M. Martinasso and J.-F. Méhaut, "A contention-aware performance model for hpc-based networks: A case study of the infiniband network," in *European Conference on Parallel Processing*, Springer, 2011, pp. 91–102 (cit. on p. 17).

[65] K. Hasanov and A. Lastovetsky, "Hierarchical optimization of mpi reduce algorithms," in *13th International Conference on Parallel Computing Technologies (PaCT-2015)*, Lecture Notes in Computer Science 9251, Springer, Petrozavodsk, Russia: Lecture Notes in Computer Science 9251, Springer, 2015, pp. 21–34 (cit. on p. 17).

[66] K. Hasanov, J.-N. Quintin, and A. Lastovetsky, "High-level topology-oblivious optimization of mpi broadcast algorithms on extreme-scale platforms," in *Euro-Par 2014: Parallel Processing Workshops, Vol. 8806 of Lecture Notes in Computer Science*, Springer, Porto, Portugal: Springer, 2014, pp. 413–425 (cit. on p. 17).

[67] Q. O. Snell, A. R. Mikler, and J. L. Gustafson, "Netpipe: A network protocol independent performance evaluator," in *In IASTED International Conference on Intelligent Information Management and Systems*, 1996 (cit. on p. 17).

[68] A. Lastovetsky, V. Rychkov, and M. O'Flynn, "Mpiblib: Benchmarking mpi communications for parallel computing on homogeneous and heterogeneous clusters," in *15th European PVM/MPI User's Group Meeting*, ser. Lecture Notes in Computer Science. Recent Advances

in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag Berlin Heidelberg, vol. 5205, Dublin, Ireland: Springer-Verlag Berlin Heidelberg, 2008, pp. 227–238, ISBN: 978-3-540-87474-4 (cit. on p. 17).

[69] A. Lastovetsky and M. O'Flynn, "A performance model of many-to-one collective communications for parallel computing," in *2007 IEEE International Parallel and Distributed Processing Symposium*, IEEE, 2007, pp. 1–8 (cit. on p. 17).

[70] J. Zhu, A. Lastovetsky, S. Ali, and R. Riesen, "Communication models for resource constrained hierarchical ethernet networks," in *European Conference on Parallel Processing*, Springer, 2013, pp. 259–269 (cit. on p. 17).

[71] K. Dichev, V. Rychkov, and A. Lastovetsky, "Two algorithms of irregular scatter/gather operations for heterogeneous platforms," English, in *EuroMPI-2010*, ser. LNCS, vol. 6305, Springer, 2010, pp. 289–293, ISBN: 978-3-642-15645-8 (cit. on p. 17).

[72] M. Kwon and S. Fahmy, "Topology-aware overlay networks for group communication," in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, ACM, 2002, pp. 127–136 (cit. on p. 17).

[73] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the international conference on Supercomputing*, ACM, 2011, pp. 75–84 (cit. on p. 17).

[74] A. Bhatele, "Automating topology aware mapping for supercomputers," 2010 (cit. on p. 17).

[75] J Liu, W Jiang, P Wyckoff, D. Panda, D Ashton, D Buntinas, B Gropp, and B Tooney, "High performance implementation of mpich2 over infiniband with rdma support," *IPDPS, Santa Fe*, 2004 (cit. on p. 18).

[76] O. M. Team *et al.*, *Open mpi: Open source high performance computing*, 2011 (cit. on p. 18).

[77] R. L. Graham and G. Shipman, “Mpi support for multi-core architectures: Optimized shared memory collectives,” in *European Parallel Virtual Machine/Message Passing Interface UsersâĂŹ Group Meeting*, Springer, 2008, pp. 130–140 (cit. on p. 18).

[78] M. Deveci, S. Rajamanickam, V. J. Leung, K. Pedretti, S. L. Olivier, D. P. Bunde, U. V. Catalyürek, and K. Devine, “Exploiting geometric partitioning in task mapping for parallel computers,” in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, IEEE, 2014, pp. 27–36 (cit. on p. 18).

[79] J. Hursey, J. M. Squyres, and T. Dontje, “Locality-aware parallel process mapping for multi-core hpc systems,” in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 527–531. DOI: 10.1109/CLUSTER.2011.59 (cit. on p. 18).

[80] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, “Hwloc: A generic framework for managing hardware affinities in hpc applications,” in *PDP 2010-The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, 2010 (cit. on p. 18).

[81] C. Coti, T. Herault, and F. Cappello, “MPI applications on grids: A topology aware approach,” in *Euro-Par 2009 Parallel Processing*, Springer, 2009, pp. 466–477 (cit. on p. 18).

[82] D. Goldsman, R. E. Nance, and J. R. Wilson, “A brief history of simulation,” in *Winter Simulation Conference*, ser. WSC ’09, Austin, Texas: Winter Simulation Conference, 2009, pp. 310–313, ISBN: 978-1-4244-5771-7 (cit. on p. 20).

[83] R. M. Fujimoto, *Parallel and Distribution Simulation Systems*, 1st. New York, NY, USA: John Wiley & Sons, Inc., 1999, ISBN: 0471183830 (cit. on p. 20).

[84] P. N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, “Single node on-line simulation of mpi applications with smpi,” in *Parallel Distributed Processing Symposium (IPDPS), 2011*

IEEE International, 2011, pp. 664–675. DOI: 10.1109/IPDPS.2011.69 (cit. on pp. 20, 21).

[85] C. M. Dobre and V. Cristea, "A simulation model for large scale distributed systems," in *Innovations in Information Technology, 2007. IIT '07. 4th International Conference on*, 2007, pp. 526–530. DOI: 10.1109/IIT.2007.4430426 (cit. on p. 21).

[86] C. J. Paredis, C. Bishop, D. Bodner, and F. G. Gonzalez, "2013 conference on systems engineering research real-time simulation and control of large scale distributed discrete event systems," *Procedia Computer Science*, vol. 16, pp. 177 –186, 2013, ISSN: 1877-0509 (cit. on p. 21).

[87] P. Bédaride, A. Degomme, S. Genaud, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau, "Toward better simulation of mpi applications on ethernet/tcp networks," in *High Performance Computing Systems. 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013.* Cham: Springer International Publishing, 2014, pp. 158–181, ISBN: 978-3-319-10214-6 (cit. on p. 21).

[88] T. Hoefler, T. Schneider, and A. Lumsdaine, "Loggopsim: Simulating large-scale applications in the loggops model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10, Chicago, Illinois: ACM, 2010, pp. 597–604, ISBN: 978-1-60558-942-8 (cit. on p. 21).

[89] M. A. Hermanns, M. Geimer, F. Wolf, and B. J. N. Wylie, "Verifying causality between distant performance phenomena in large-scale mpi applications," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 78–84. DOI: 10.1109/PDP.2009.50 (cit. on p. 21).

[90] J. Zhai, W. Chen, and W. Zheng, "Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node," in *Proceedings of the 15th ACM SIGPLAN Symposium on*

Principles and Practice of Parallel Programming, ser. PPoPP '10, Bangalore, India: ACM, 2010, pp. 305–314, ISBN: 978-1-60558-877-3 (cit. on p. 21).

[91] A. Nunez, J. Fernandez, J. D. Garcia, and J. Carretero, "New techniques for simulating high performance mpi applications on large storage networks," in *2008 IEEE International Conference on Cluster Computing*, 2008, pp. 444–452. DOI: 10.1109/CLUSTR.2008.4663806 (cit. on p. 21).

[92] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely, "Psins: An open source event tracer and execution simulator," in *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, 2009, pp. 444–449. DOI: 10.1109/HPCMP-UGC.2009.73 (cit. on p. 21).

[93] R. Bagrodia, E. Deelman, and T. Phan, "Parallel simulation of large-scale parallel applications," *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 1, pp. 3–12, Feb. 2001, ISSN: 1094-3420 (cit. on p. 21).

[94] D. A. Grove and P. D. Coddington, "Communication benchmarking and performance modelling of mpi programs on cluster computers," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, pp. 249–. DOI: 10.1109/IPDPS.2004.1303309 (cit. on p. 21).

[95] B. Penoff, A. Wagner, M. TÃijxen, and I. RÃijngeler, "Mpi-netsim: A network simulation module for mpi," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, 2009, pp. 464–471. DOI: 10.1109/ICPADS.2009.116 (cit. on p. 21).

[96] A. Lastovetsky, "Heterogeneous parallel computing: From clusters of workstations to hierarchical hybrid platforms," *Supercomputing Frontiers and Innovations*, vol. 1, pp. 70–87, 2014 (cit. on p. 23).

[97] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers," in *HPCN'99*, 1999.

[98] A. Kalinov and A. Lastovetsky, “Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers,” *Journal of Parallel and Distributed Computing*, vol. 61, no. 4, pp. 520–535, 2001 (cit. on p. 25).

[99] E. Dovolnov, A. Kalinov, and S. Klimov, “Natural block data decomposition for heterogeneous clusters,” in *IPDPS 2003*, 2003 (cit. on p. 25).

[100] A. Lastovetsky, “On grid-based matrix partitioning for heterogeneous processors,” in *6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, IEEE Computer Society, Hagenberg, Austria: IEEE Computer Society, 2007, pp. 383–390 (cit. on p. 25).

[101] A. DeFlumere, A. Lastovetsky, and B. Becker, “Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution,” in *IPDPSW 2012*, 2012, pp. 125–139 (cit. on p. 25).

[102] A. DeFlumere and A. Lastovetsky, “Optimal data partitioning shape for matrix multiplication on three fully connected heterogeneous processors,” in *Euro-Par 2014: Parallel Processing Workshops: Porto, Portugal, August 25-26, 2014,* Cham: Springer International Publishing, 2014, pp. 201–214, ISBN: 978-3-319-14325-5 (cit. on p. 25).

[103] A. Kalinov, “Scalability analysis of matrix-matrix multiplication on heterogeneous clusters,” *Parallel and Distributed Computing, International Symposium on*, vol. 0, pp. 303–309, 2004. DOI: `http://doi.ieeecomputersociety.org/10.1109/ISPDC.2004.45` (cit. on p. 49).

[104] P. Smolarkiewicz, “Multidimensional positive definite advection transport algorithm: an overview,” *Int. J. Numer. Meth. Fluids*, vol. 50, pp. 1123–1144, 2006 (cit. on p. 53).

[105] P. Smolarkiewicz and L Margolin, "Mpdata: a finite-difference solver for geophysical flows," *Journal of Computational Physics*, vol. 140, pp. 459–480, 1998 (cit. on p. 53).

[106] Z. Piotrowski, A. Wyszogrodzki, and P. Smolarkiewicz, "Towards petascale simulation of atmospheric circulations with soundproof equations," *Acta Geophysica*, vol. 59, pp. 1294–1311, 2011 (cit. on p. 53).

[107] A. Wyszogrodzki, Z. Piotrowski, and G. W., "Parallel implementation and scalability of cloud resolving eulag model," *Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2011, Part II. LNCS*, vol. 7204, pp. 252–261, 2012 (cit. on p. 53).

[108] L. Szustak, K. Rojek, T. Olas, L. Kuczynski, K. Halbiniak, and P. Gepner, "Adaptation of mpdata heterogeneous stencil computation to intel xeon phi coprocessor," *Scientific Programming*, p. 14, 2015 (cit. on p. 54).

[109] B Rosa, L Szustak, A. Wyszogrodzki, K Rojek, D. Wojcik, and R Wyrzykowski, "Adaptation of multidimensional positive definite advection transport algorithm to modern high-performance computing platforms," *International Journal of Modeling and Optimization*, vol. 5, no. 3, pp. 171–176, 2015. DOI: 10.7763/IJMO.2015.V5.456 (cit. on p. 54).

[110] T. Malik, V. Rychkov, A. Lastovetsky, and J. N. Quintin, "Topology-aware optimization of communications for parallel matrix multiplication on hierarchical heterogeneous hpc platform," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, 2014, pp. 39–47. DOI: 10.1109/IPDPSW.2014.10 (cit. on pp. 66, 85, 86).

[111] T. Malik, V. Rychkov, and A. Lastovetsky, "Network-aware optimization of communications for parallel matrix multiplication on hierarchical hpc platforms," *Concurrency and Computation: Practice and Experience*,

vol. 28, no. 3, pp. 802–821, 2016, cpe.3609, ISSN: 1532-0634 (cit. on p. 66).

[112] T. Malik, L. Szustak, R. Wyrzykowski, and A. Lastovetsky, "Network-aware optimization of mpdata on homogeneous multi-core clusters with heterogeneous network," in *Workshop on Theoretical Approaches to Performance Evaluation, Modeling and Simulation (ICA3PP-TAPEMS), 2016 Springer*, 2016 (cit. on p. 66).

[113] *Execo-g5k grid5000 tool*. [Online]. Available: `http://execo.gforge.inria.fr/doc/latest-stable/execo_g5k.html` (cit. on p. 87).

[114] *Topo5k tool*. [Online]. Available: `https://github.com/lpouillo/topo5k` (cit. on p. 87).

[115] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut, "Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures," *Concurrency and Computation: Practice and Experience*, p. 16, May 2015 (cit. on p. 90).

Appendix A

Simulation Experiments

This section describes our efforts for running experiments on a simulated platform. There are many constraints, related to time and resources allocation, involved while running large scale experiments on a Grid5000 real platform. Therefore, we decided to try SimGrid to simulate the Grid5000 platform and to run applications in a simulated environment. The objective of these experiments was to validate the performance of our proposed algorithms on a different kind of complex and large-scale platform by doing large scale experiments. Our work [110] is rich with Grid5000 platform experimental results representing a good starting point for simulation based evaluation. Here, we discuss how we run SMPI experiments and what limitations and difficulties we have faced.

A.1 SimGrid-SMPI Experiments

SimGrid is an active developing software and has undergone many adaptations since the day we have started working on it. Its current version fixes many issues that we faced with its early versions. The SMPI module of SimGrid is now considered as stable and has a very decent coverage of the MPI interface. However, our experiments have shown that it still restricts the user to run some applications due to many design constraints and limitations. Currently, any MPI application that is written in C, Fortran and Java can run unmodified within SMPI provided that (1) application only uses MPI calls that are implemented in SMPI, and (2) it does not use global variables.

In order to run any simulation, SimGrid must be provided with three things:

- An application to run: That application must use one of the communication APIs provided by SimGrid and must be written in C, Fortran or Java.
- Platform description: where the user wants to simulate the execution of the application.
- Application deployment information: For example: Which process should be executed onto which processor/core.

First step towards executing our MPI application on SPMI is to compile application with the SimGrid MPI interface. This is done with the $smpicc$ compiler of SimGrid. The other important step before running the simulation is modeling of the underlying platform. This is the most crucial step in simulation experiments. Accuracy and speed also depend on the complexity of the simulated platform and the accurate of its modeling. The widely used available format is XML for platform description, and Lua Support is also provided. The XML checking is done based on the $simgrid.dtd$ Document Type Definition (DTD) file. The deployment information can also be provided as an xml file or as parameters to the script running the application. Once modeling of the platform and the deployment information are ready, the MPI application compiled with the $smpicc$ compiler can be executed by the simulator. This is done using the $smpirun$ script. In addition to the platform description and deployment information, the user also needs to provide an MPI hostfile that contains the names of nodes where the processes should run, one per line. These hosts must be present in the provided platform description. The command to run application on SMPI is:

smpirun -hostfile my_hostfile.txt -platform my_platform.xml ./my_program -arg my_program_arguments

For the test case, we reproduced on SMPI one of our real platform experiments with the matrix multiplication application presented in [110]. In these grid5000 experiments, we used four heterogeneous clusters with different numbers of nodes. We spawn one MPI process per node, with different numbers of threads to increase the heterogeneity. We started the experiments with 16 nodes, 4 nodes on the “Edel" cluster, 5 nodes on the “Taurus" cluster, 3 nodes on the “Sol" cluster and 4 nodes on the “Chimint" cluster. The problem size was 16384, which is the area of a square s128×128 matrix, given in the number of $b \times b$ blocks. While reproducing the same experiments on SMPI, we observed non-realistic and non-deterministic computation and communication times. Our experiments show that due to the complexity and multiple design constraints, SimGrid cannot measure the realistic communication cost on highly heterogeneous complex platforms with applications having asynchronous point-to-point communication operations.

A.2 Issues during SMPI Experiments

In this section, we discuss the challenges that we faced and most important the factors that influenced the realistic measurement of execution time on SMPI.

- We found that most of the published SimGrid simulation work is done using MSG or SimDag. We did not find much work using SMPI, even the documentation examples are mainly focused on MSG.
- The first problem we faced was how to automatically map an existing Grid5000 platform? Currently, SimGrid does not provide any tool to accurately map any existing platform. However, the SimGrid team is working on a tool called ‘ALNeM‘, to automatically discover the topology of an existing network, and the output will be a platform description file following the SimGrid syntax. This tool is however not ready yet. In the absence of it, manual mapping has proved to be a tedious, complicated and error prone task. We used the execo_g5k [113] and topo5k [114] tools for topology generation of Grid5000. However, these tools are still under development and cannot properly handle complex sites.
- In order to provide accurate timings for SMPI simulations on any

platform, it is vital to take several idiosyncrasies of that platform into account. SimGrid has recently provided a calibration method that runs several MPI benchmarks on the machine, and statistical analysis of the collected data then determines several platform parameters used by SimGrid. However, running this point-to-point based calibration procedure is extremely time consuming. It requires tweaking of parameters and will consider the state of the system at the time of experiments. It means that different runs using the same configuration may yield different results. Characterization of the simulated clusters in a way when you can have a realistic replay with SMPI is still an unpaved way. Right now, the information for the Graphene cluster of Grid5000 is the most accurate file that SimGrid provides, but such an accurate file is only available for one cluster.

- In our real experiments we create heterogeneity inside nodes of the cluster by using MKL threads and benchmark the speed of each node using FPM. While modeling platform for SimGrid, we were not able to create heterogeneous clusters using XML description. This is simply impossible with the XML format, and can turn to be rather complex in the code. If the goal is to have "heterogeneous cluster", the lua mechanism would work but it is quite new and currently lacking a proper documentation. As an alternative, we created a regular Asynchronous System ‘AS‘ with a set of separated hosts to represent heterogeneous cluster and used ’Full’ routing to specify each and every route. However this made our platform file much more complex and larger.

- In SMPI, for accurate simulation the user must have to provide accurate flop rates for hosts both in the deployment argument and in the platform file. We used a functional performance model based benchmark to measure the speed of nodes which measured the speed of each node as a continuous function of problem size. However, SimGrid relies on a simplistic CPU model $(time = size/power)$. So for running the simulation we measured the speed of nodes using the FPM benchmark on the real Grid5000 platform and used maximum speed values as the host speed parameter in the platform description file.

- In SimGrid, communications are synchronous. If we measure the time before and after the communication, we get the time spent in real communication plus the transmission time, and it also includes the time spent in waiting for other party to be ready. The best way to get the realistic computation time is to run the simulation on one of the nodes of the target cluster and specify the exact same rate in the platform file and in the command line. In this way, real timing is used in the simulation without re-scaling. However, this only works in the case of homogeneous cluster. Our experiments are inter-cluster, and even the same cluster has heterogeneous nodes. In that case, internal simulation technique is not much helpful in getting the realistic simulation.

- In SMPI, by default computations are benchmarked and then the computed times are used instead of computing everything over and over again. The actual timing is measured on the host machine and then scaled to the power of the corresponding simulated machine. We can specify the power of the host machine by using the variable $smpi/running - power$. We observed that it affects the computation time and also affects waiting time in some MPI calls. Hence, communication time will be affected too. For example, if we have a very low value for the running-power, for example (5.5Mflops), which means that when the simulator runs something in 1 second on host computer, and needs to translate it to a simulated host with a huge amount of power, for example 23492000000 for one of simulated host, it will multiply the timing by $55000000/23492000000 = 0.0023$. This, in the end, will have very small computing time. In the resulting time, you will then have communication time and vice versa. Moreover, simulating asynchronous communication is very tricky in SMPI.

- In SMPI, MPI processes run as real UNIX processes. Thus, the application written using threads or OpenMP would not be simulated.

Currently there is no way to simulate OpenMP applications in SimGrid. The main issue is that SimGrid cannot intercept thread interactions in OpenMP applications. For MPI, they re-implemented all the library calls, but there is no library call in OpenMP that directs thread interactions.

- SimGrid does not support multi-level parallelization. Any MPI application that uses MPI+ OpenMP cannot be simulated in SimGrid. It restricts us to simulate the MPDATA application in SimGrid that uses multi-level parallelization to gain the maximum benefit of available resources. The SimGrid team is currently working on simulating multi-level parallelization, but only based on MPI+StarPU.

- As SMPI also does not support multi-core experiments, the user cannot perform GPU based experiments in SMPI. Currently it only supports CPU based experiments. In [115] they worked on modeling GPUs, but it was ad-hoc and the corresponding abstractions have not been integrated into SimGrid yet.

Appendix B

List of abbreviations

The following describes the significance of various acronyms and terms used throughout this thesis. The page on which each one is defined or used is also given.