Decentralized environment and communication protocol for grid-intensive computing

Bilal Fakih

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *09/11/2018* **par :**

Bilal FAKIH

Environnement décentralisé et protocole de communication pour le calcul intensif sur grille

---

JURY

M. J.-F. MEHAUT	Professeur d'Université	Rapporteur
M. P. BERTHOU	Maître de Conférence	Examinateur
Mme. N. EMAD	Professeur d'Université	Examinateur
M. D. EL BAZ	Chargé de Recherche	Directeur de thèse
M. A. DONCESCU	Maître de Conférence	Invité

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes
Directeur de Thèse :
M. Didier EL BAZ
Rapporteurs :
M. Jean-françois MEHAUT et *M. Romeo SANCHEZ*

Remerciements

Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Les travaux de thèse présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS). Je tiens à remercier Monsieur Jean ARLAT et Monsieur Liviu Nicu, directeurs successifs du LAAS - CNRS, pour m'avoir accueilli au sein de ce laboratoire.

Je tiens à remercier mon encadrant, Monsieur Didier EL BAZ, pour son écoute et ses conseils tout au long de cette thèse et qui m'a fait partager ses brillantes intuitions. Qu'il soit aussi remercié pour sa gentillesse, sa disponibilité permanente et pour les nombreux encouragements qu'il m'a prodiguée. Je le remercie aussi pour les nombreuses réflexions que nous avons pu mener ensemble et au travers desquelles il a partagé une partie de son expérience avec moi.

J'adresse tous mes remerciements à Monsieur Jean-francois Méhaut, Professeur à l'université de Grenoble, ainsi qu'à Monsieur Roméo SANCHEZ NIGENDA , Professeur à l'université de Nuevo Leon , de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse. Ils ont pris le temps de m'écouter et de discuter avec moi. Leurs remarques et conseils m'ont permis d'améliorer la clarté de la présentation des idées véhiculées par le présent manuscrit.

J'exprime ma gratitude à Madame Nahid Emad, Professeur à l'université de Versailles, et à Monsieur Pascal Berthou, Maitre de conférences à l'Université Toulouse III, qui ont bien voulu être examinateurs dans mon jury de thèse.

Je remercie également Monsieur Andreï DONCESCU pour l'honneur qu'il me fait d'être dans mon jury de thèse.

Mes remerciements vont naturellement à l'ensemble des membres du groupe CDA: Bastien Plazolles, Zhuli et Jia avec lesquels j'ai partagé ces années de thèse. Un grand merci à mes amis Libanais : Abbas, Mouhannad, Iman, Fatima, Mouhammad Issa, Mouhammad Hourani, et tout les membres du groupe 'Muntada Essudfa' pour leur aide précieuse durant ces années.

Enfin, je souhaite remercier ma famille : mes frères et soeur Ali, Mohammad et jamila, mes parents Hussein et Mariam pour leur encouragement durant ces années, y compris dans les moments difficiles.

Résumé

Dans cette thèse nous présentons un environnement décentralisé pour la mise en oeuvre des calcul intensif sur grille. Nous nous intéressons à des applications dans les domaines de la simulation numérique qui font appel à des modèles de type parallélisme de tâches et qui sont résolues par des méthodes itératives parallèles ou distribuées; nous nous intéressons aussi aux problèmes de planification. Mes contributions se situent au niveau de la conception et la réalisation d'un environnement de programmation GRIDHPC. GRIDHPC permet l'utilisation de tous les ressources de calcul, c'est-à-dire de tous les coeurs des processeurs multi-coeurs ainsi que l'utilisation du protocole de communication RMNP pour exploiter simultanément différents réseaux hauts débits comme Infiniband, Myrinet et aussi Ethernet. Notons que RMNP peu se reconfigurer automatiquement et dynamiquement en fonction des exigences de l'application, comme les schémas de calcul, c.-à-d, les schémas itératifs synchrones ou asynchrones, des éléments de contexte comme la topologie du réseau et le type de réseau comme Ethernet, Infiniband et Myrinet en choisissant le meilleur mode de communication entre les noeuds de calcul et le meilleur réseau.
Nous présentons et analysons des résultats expérimentaux obtenus sur des grappes de calcul de la grille Grid5000 pour le problème de l'obstacle et le problème de planification.

Abstract

This thesis aims at designing an environment for the implementation of high performance computing applications on Grid platforms. We are interested in applications like loosely synchronous applications and pleasingly parallel applications. For loosely synchronous applications, we are interested in particular in applications in the domains of numerical simulation that can be solved via parallel or distributed iterative methods, i.e., synchronous, asynchronous and hybrid iterative method; while, for pleasingly parallel applications, we are interested in planning problems. Our thesis work aims at designing the decentralized environment GRIDHPC. GRIDHPC exploits all the computing resources (all the available cores of computing nodes) using OpenMP as well as several types of networks like Ethernet, Infiniband and Myrinet of the grid platform using the reconfigurable multi network protocol RMNP. Note that RMNP can configure itself automatically and dynamically in function of application requirements like schemes of computation, i.e., synchronous or asynchronous iterative schemes, elements of context like network topology and type of network like Ethernet, Infiniband and Myrinet by choosing the best communication mode between computing nodes and the best network.
We present and analyze a set of computational results obtained on Grid5000 platform for the obstacle and planning problems.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Contents

1.1 Context

The domains of high-performance computing (HPC) uses parallel processing methods and architectures for running advanced application programs efficiently and quickly. The most popular solutions to solve HPC applications are to use supercomputers that are composed of hundreds thousands of processor cores connected by a local high-speed computer bus. The system, called the Summit, at Oak Ridge, United States, presently keeps the top position of TOP500 list of world's supercomputers [80]. Supercomputers were the leaders in the field of computing, but due to the fact that supercomputers are very expensive and consume a lot of energy; new concepts have been proposed like peer-to-peer, volunteer, grid and global computing. In this thesis, we concentrate on grid computing concept. Grid computing is the collection of computer resources (desktop or cluster nodes) to handle long-running computational tasks in order to solve large scale HPC applications.
This thesis aims at designing an environment for the implementation of high performance computing applications on Grid platforms. We are interested in applications like loosely synchronous applications and pleasingly parallel applications. Note that, loosely synchronous applications present frequent data exchange between computing nodes; while pleasingly parallel applications do not need any data exchange, i.e., each component works independently. Note also that, for loosely synchronous applications, we are interested in particular in applications in the domains of numerical simulation and optimization that can be solved via parallel or distributed iterative methods, i.e., synchronous, asynchronous or hybrid iterative method; while, for pleasingly parallel applications, we are interested in planning problems. Our thesis work extends the P2PDC decentralized environment [11 ; 17] with Grid Computing capabilities. Originally, P2PDC solves loosely synchronous applications via a peer-to-peer network. The proposed version, called GRIDHPC, exploits all the computing resources (all the available cores of computing nodes) as well as several types of network like Ethernet, Infiniband and Myrinet of the grid platform. Our environment is built on a decentralized architecture whereby computing nodes can exchange data directly via multi-network configurations. Finally, we note that our approach is developed in C language that is very efficient for HPC applications.

1.2 Contributions

Our contributions include the following points

- The design and implementation of a reconfigurable multi-network communication protocol (RMNP) that permits one to allow rapid update exchange between computing nodes in a multi-network configurations (Ethernet, Infiniband and Myirnet). We note that RMNP can configure itself automatically and dynamically in function of application requirements like schemes of computation (synchronous, asynchronous or hybrid iterations), elements of context like network topology and type of network like Ethernet, Infiniband or Myrinet by choosing the best communication mode between computing nodes and the best network.

- We design and develop the decentralized environment GRIDHPC for high performance computing applications on Grid platforms. In particular, we detail the global topology and the general architecture of the decentralized environment GRIDHPC with its main functionalities.
  GRIDHPC facilitates the use of multi-cluster and grid platform for loosely synchronous applications and embarrassingly parallel application. It exploits all the computing resources (all the available cores of computing nodes) as well as several types of network like Ethernet, Infiniband and Myrinet in the same application. This contribution is divided into two phases :
  The first phase corresponds to the use of several network simultaneously like Ethernet, Infiniband and Myrinet. This feature is particularly important since we consider loosely synchronous applications that present frequent data exchanges between computing nodes. Note that this objective is done via the reconfigurable multi-network protocol RMNP.
  The second phase corresponds to the use of all the computing resources of modern muli-core CPUs, i.e., all CPU cores. This objective is done via OpenMP.
  In summary, we note that the main originalities of our approach are:

  - Facilitate programming by hiding the choice of communication mode.
  - a decentralized environment developed in C language that is very efficient for HPC applications;
  - Hierarchical master-worker mechanism with communication between computing nodes. This mechanism accelerates task allocation to computing nodes and avoids connection bottleneck at submitter. Note that the computing nodes are organized in groups to optimize inter-cluster communications.

  - a reconfigurable multi-network protocol (RMNP) that permits one to have efficient and frequent direct communications between computing nodes in a multi-network configurations.
  - a decentralized environment (GRIDHPC) that permits to exploits all the computing resources of multi-cluster and grid platforms. In particular, it exploits all the available cores of computing nodes using OpenMP and search the best underlying network (high speed and low latency network like Infiniband and Myirnet) to perform communications between computing nodes via multi-network configurations using RMNP.

- The use of GRIDHPC environment for the solution of a numerical simulation problem, i.e. the obstacle problem and the test of this application on GRID5000 platform with up to 1024 computing cores. In particular, we have considered a decomposition method and a termination method of the problem.

- The use of GRIDHPC environment for planning problem and the test of this application on GRID5000 platform with up to 40 computing cores. In particular, a decomposition method has been implemented in order to response to the requirements of GRIDHPC.

1.3 Overview

This thesis is organized as follows:

- **Chapter 2** presents the state of the art of the domains that inspire the contribution of this thesis. In particular, we concentrate on approaches related to high performance and distributed computing, i.e. grid computing, global computing, peer-to-peer high performance computing and volunteer computing. An overview on existing environments and softwares for these approaches are also presented in this chapter.

- **Chapter 3** describes the reconfigurable multi-network communication protocol dedicated to HPC applications. We display the global architecture of RMNP with its main functionalities to allow rapid update exchange between computing nodes in multi-network configurations via distributed iterative algorithms. We detail also the RMNP mechanisms of the protocol that permits to choose the best network and the best micro-protocol for the configuration of RMNP.

- **Chapter 4** presents the decentralized environment GRIDHPC. In this chapter, we describe the global topology and the general architecture of GRIDHPC with its mains functionalities. We present the hierarchical task allocation mechanism that accelerates task allocation to computing nodes and avoids connection bottleneck at submitter. Moreover, a programming model for GRIDHPC that is suited to high performance computing applications and more particularly applications solved by iterative algorithms is presented in this chapter.

- **Chapter 5** presents an application to obstacle problem with the decentralized environment GRIDHPC. In particular, a set of computational experiments with GRIDHPC for the obstacle problem are presented and analyzed; we study also the combination of GRIDHPC and distributed synchronous or asynchronous iterative schemes of computation for the obstacle problem in a multi-core and multi-network context. The experiments are carried out on the Grid5000 platform with up to 1024 computing cores. We presents also the decomposition method of the obstacle problem and the different termination method that has been implemented.

- **Chapter 6** presents an application to a planning problem with GRIDHPC. In particular, a set of computational experiments with GRIDHPC for the planning problem are presented and analyzed. The experiments are carried on the Grid5000 platform with up to 40 computing cores. We presents also the decomposition method and the implementation of the problem with GRIDHPC.

- **Chapter 7** gives some conclusions on our work and deals also with future work.

CHAPTER 2

State of the art

Contents

2.1 Introduction

This chapter presents the state of the art of the domains that inspire the contribution of this thesis. Section 2.2 presents an overview of HPC and HTC applications. Section 2.3 deals with parallel programming models. Section 2.4 presents an overview on existing softwares and middlewares for volunteer, peer-to-peer, grid and global computing. Section 2.5 deals with runtime systems. Finally, section 2.6 concludes this chapter.

2.2 HPC and HTC applications

High Performance Computing (HPC) aggregates computing power to deliver higher performance in order to solve complex or large scale problems in science or engineering. In particular, HPC tasks require large amounts of computing power for short periods of time.

High-throughput computing (HTC) is a computer science term to describe the use of many computing resources over long periods of time to accomplish a computational task. In particular, HTC tasks require large amounts of computing power for long periods of time.

There are many kinds of applications for parallel or distributed systems. Hwang, Fox and Dongarra [1] provide a classification of such applications in their book. Table 2.1 shows the classification in [1] where important categories of applications for parallel and distributed systems are listed and described. Categories 1 to 5 are related to computational intensive applications or High Performance Computing. The last category is relative to High-Throughput Computing, that is, data intensive computing applications. Category 1 (Synchronous processing) describes applications that can be parallelized with lock-step operations controlled by hardware like signal processing or image processing; this class of processing, that was very popular 25 years ago, is no more noteworthy. Category 2 (Loosely synchronous processing) consists of compute–communicate phases whereby computations are synchronized by communication steps. Category 3 (Asynchronous processing) consists of asynchronously interacting objects and it is often considered the people's view of a typical parallel problem. This class of processing can be found for example in asynchronous preconditioning techniques for large systems of equations as well as integer programming and nonlinear optimization [23–26]. Category 4 (Pleasingly parallel processing) is the simplest algorithmically, with disconnected parallel components. This class can be found in parallel Monte-Carlo simulations [86]. Category 5 (Metaproblems processing) refers to coarse-grained linkage of different "atomic" problems. It is a com-

Category	Class	Description	Machine Architecture
1	Synchronous	The problem class can be implemented with instruction-level lockstep operation as in SIMD architectures.	SIMD
2	Loosely synchronous (BSP or bulk synchronous processing)	These problems exhibit iterative compute communication stages with independent compute (map) operations for each CPU that are synchronized with a communication step.	MIMD on MPP (massively parallel processor)
3	Asynchronous	Illustrated by Compute Chess and Integer Programming; combinatorial search is often supported by dynamic threads. This is rarely important in scientific computing, but it is at the heart of operating systems and concurrency in consumer applications such as Microsoft Word.	Shared memory
4	Pleasingly parallel	Each component is independent. In 1988, Fox estimated this at 20 percent of the total number of applications, but that percentage has grown with the use of grids and data analysis applications including, for example, the Large Hadron Collider analysis for particle physics.	Grids moving to clouds
5	Metaproblems	These are coarse-grained (asynchronous or data flow) combinations of categories 1-4 and 6.	Grids of clusters
6	MapReduce++ (Twister)	This describes file (database) to file (database) operations which have three subcategories : 6a) Pleasingly Parallel Map Only (similar to category 4) 6b) Map followed by reductions 6c) Iterative 'Map followed by reductions' (extension of current technologies that supports linear algebra and data mining)	Data-intensive clouds a) Masterworker or MapReduce b) MapReduce c) Twister

Table 2.1: Application classification for parallel and distributed systems taken from [1]

bination of categories 1 to 4 and 6. Category 6 (MapReduce++ processing) describes file to file operations which have three subcategories : 'map only' applications similar to pleasingly parallel (category 4) [81–84]; map followed by reductions; and a subcategory that extends MapReduce version and that supports linear algebra and data mining.
In this thesis, we are interested in applications that belong to the class of loosely synchronous applications (category 2) and pleasingly parallel applications (category 4). The decentralized environment P2PDC has been proposed in 2008 [11 ; 17]. It was dedicated to solve loosely synchronous applications via a peer-to-peer network. In this thesis, we extend P2PDC to grid computing. In particular, the decentralized environment (GRIDHPC) exploits all the computing resources (all the available cores of computing nodes) as well as several types of network like Ethernet, Infiniband and Myrinet of the grid platform. Note that, for loosely synchronous applications, we are interested in particular in applications in the domains of numerical simulation and optimization that can be solved via parallel or distributed iterative methods; while, for pleasingly parallel applications, we are interested in planning problems. We provide an evaluation of our platform using such domains in Chapters 5 and 6.

2.3 Parallel programming models

A parallel programming model is an abstraction for parallel computing, which is convenient to express parallelism of algorithms.
The classifications of parallel programming models can be divided into two areas: problem decomposition and process interaction. Problem decomposition relates to the way in which the constituent processes are formulated; while process interaction relates to the mechanisms by which parallel processes are able to communicate with each other. The most common forms of interaction are shared memory and message passing.

2.3.1 Shared Memory

This model assumes that programs will be executed on different processors that share the same memory (see Figure 2.1). Shared-memory programs are typically executed by multiple independent threads; the threads share data but may also have private data. Shared-memory approaches to parallel programming must provide means for starting up threads, assigning work to them and coordinating their accesses to shared data ensuring that certain operations are performed by only one thread at a time. Multi-core processors directly support shared memory, which many parallel programming languages and

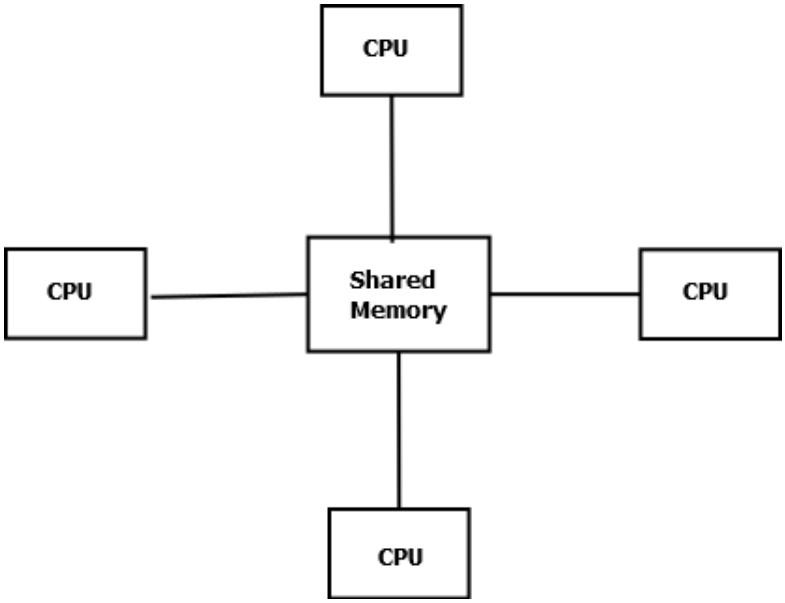

Figure 2.1: Shared Memory

libraries like OpenMP are designed to exploit. In this subsection, we will present in details OpenMP.

2.3.1.1 OpenMP

Before describing OpenMP, we give a simple example as shown in Figure 2.2. Figure 2.2 shows an OpenMP program, i.e., sequential and parallel regions. When the master thread enters to the parallel region, it creates child processes to perform different computation tasks in parallel composed of the same code. At the end of the parallel region, the execution becomes sequential.

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-processing and enable portable shared memory. It provides an approach very easy to learn as well as apply. It enables programmers to work with a single source code: if a single set of source files contains the code for both the sequential and the parallel versions of a program, then program maintenance is much simplified. It is not a new programming language. Rather it is notation that can be added to a sequential program in C for example to describe how the work is to be shared among threads that will execute on different processors and to order access to shared data as needed.

2.3.1.2 OpenMP Program

An OpenMP directive is an instruction in a special format that is understood by OpenMP compilers only. It looks like a pragma to a C/C++ compiler, so that the program may

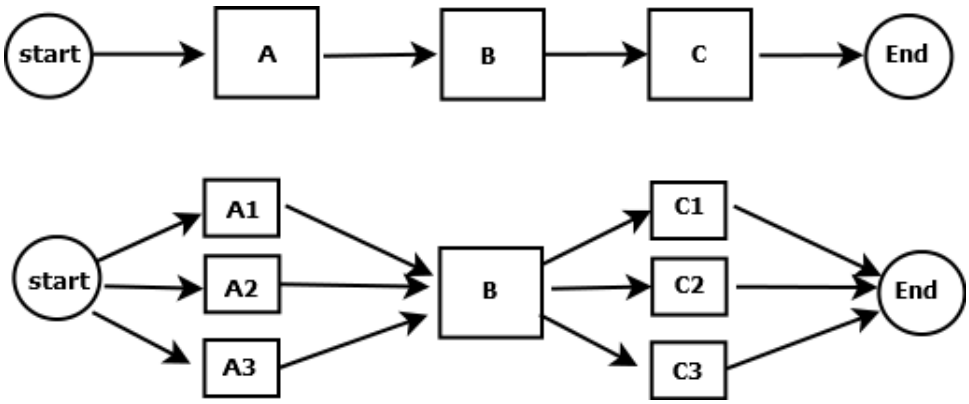

Figure 2.2: Parallel Programming With OpenMP

run just as it did beforehand if a compiler is not OpenMP-aware. The API does not have many different directives, but they are powerful enough to cover a variety of needs.
The first step in creating an OpenMP program from a sequential one is to identify the parallelism it contains. This means, finding a region of code that may be executed concurrently by different processors. Hence, the developer must reorganize portions of a code to obtain independent instruction sequences or replace an algorithm with an alternative one that accomplishes the same task but offers more exploitable parallelism.

2.3.2 Distributed Memory

This model assumes that programs will be executed by different processes, each of which has its own private space (see Figure 2.3). Message-passing approaches to parallel programming must provide a means to manage the processes to send and receive messages, and to perform special operations across data distributed among the different processes. Note that MPI corresponds to the distributed memory programming model, i.e., message passing. In this subsection, we will present MPI and MPICH-Madeleine, i.e., a middleware developed using MPI.

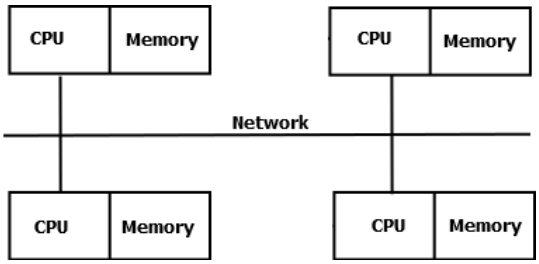

Figure 2.3: Distributed memory

2.3.2.1 MPI

MPI [89] is an interface of communication which provides high performance communications on different network architectures, i.e. Gigabit-Ethernet, Myrinet, Infiniband, GigaNet and SCI. It is the primary programming standard used to develop parallel programs to run on a distributed memory architectures. It is essentially a library of subprograms that can be called from C or FORTRAN to write parallel programs running on a distributed system.

2.3.2.2 MPICH/Madeleine

MPICH-Madeleine [6] is an implementation of the MPI standard based on the MPICH implementation and the multi-protocol communication library called Madeleine.
MPICH-Madeleine permits to use the underlying communication software and hardware functionalities for distributed applications. It is able to exploit clusters of clusters with heterogeneous networks.
Madeleine [4 ; 5 ; 7] uses objects called channels in order to virtualize the available networks in a given configuration. There are two types of channels: physical channels which are the real existing networks and virtual channels which are built above physical channels and can be used to create heterogeneous networks. Figure 2.4 shows the architecture of MPICH-Madeleine.
The implementation of MPICH-Madeleine is based on a device called ch_mad, which handles several Madeleine channels in parallel and allows the use of several networks at the same time within the same application, i.e., handles any inter-node communication. Another device is used for handling intra-node communications, i.e., smp_plug. This device uses the same concepts of ch_mad, that is, a thread is responsible for executing the polling of incoming communications. Note that MPICH-Madeleine benefits also from the advanced polling mechanisms available within the Marcel library (see figure 2.4). Another device for handling intra-process communications is name ch_self. Note also that MPICH-Madeleine has a component named Abstract Device Interface [65 ; 66] which provides a portable message passing interface to the generic upper layer.

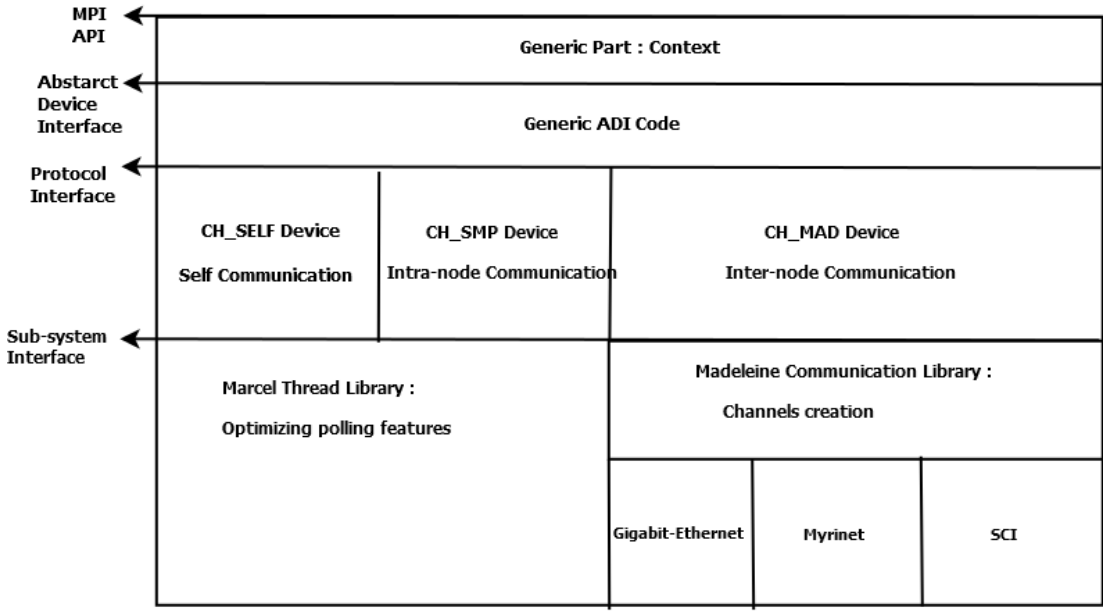

Figure 2.4: Architecture of MPICH-Madeleine

2.4 Computing Concepts

In this section, we briefly present four recent parallel or distributed computing concepts : peer-to-peer, volunteer, global, and grid computing. In particular, this section presents an overview on existing softwares and middlewares related to these computing concepts.

2.4.1 Peer-to-peer and Volunteer computing

Peer-to-Peer (P2P) systems have known great developments thanks to file sharing systems on the Internet like Gnutella [87] or FreeNet [88], video streaming and distributed database. Thanks to the progress in network technology, peer-to-peer computing can now be used for HPC applications. Volunteer computing is a concept whereby volunteers provide computing resources to projects, which use the resources to perform parallel or distributed computing.
In this subsection, we shall describe the different architectures of peer-to peer systems that can be encountered. We shall present also several middleware for peer-to-peer and volunteer computing.

2.4.1.1 Architecture of Peer-To-Peer systems

We can classify peer-to-peer systems into three categories : centralized, decentralized and hybrid architectures. In the sequel, we will describe each one.

-Centralized architecture

Figure 2.5 shows a diagram of a centralized peer-to peer system. In this model, a central server stores information about all the peers of the system. When it receives a request of communication from a peer, it selects another peer from its directory that matches the request in order to perform the communications.

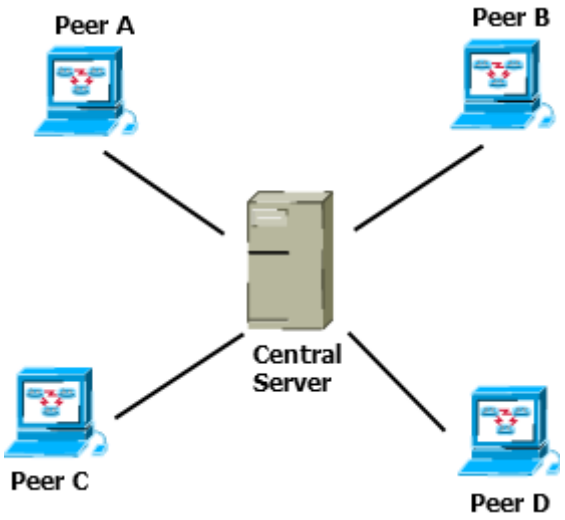

Figure 2.5: Centralized P2P system

-Decentralized architecture

Figure 2.6 shows a diagram of a decentralized peer-to peer system. This architecture does not rely on any server and the communication are carried out directly between peers.

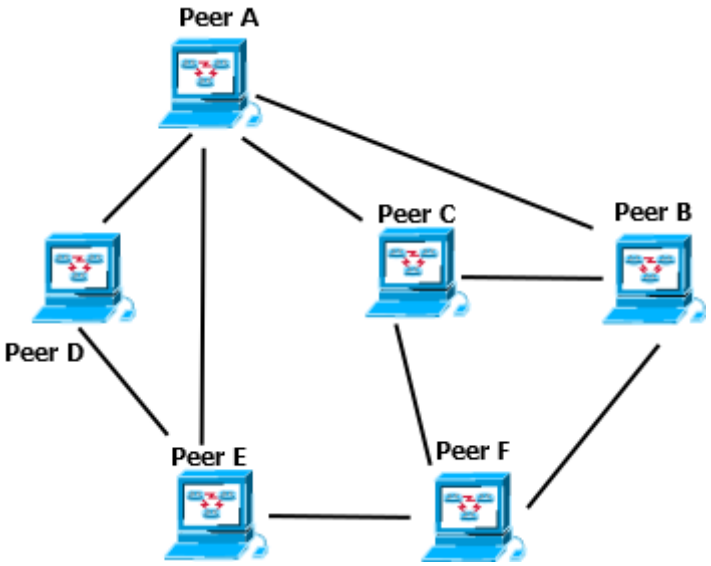

Figure 2.6: Decentralized P2P system

-Hybrid architecture

Figure 2.7 shows a diagram of hybrid peer-to peer system. It is the combination of both centralized and decentralized architectures. Note that, it uses super nodes that monitor a set of peers connected to the system.

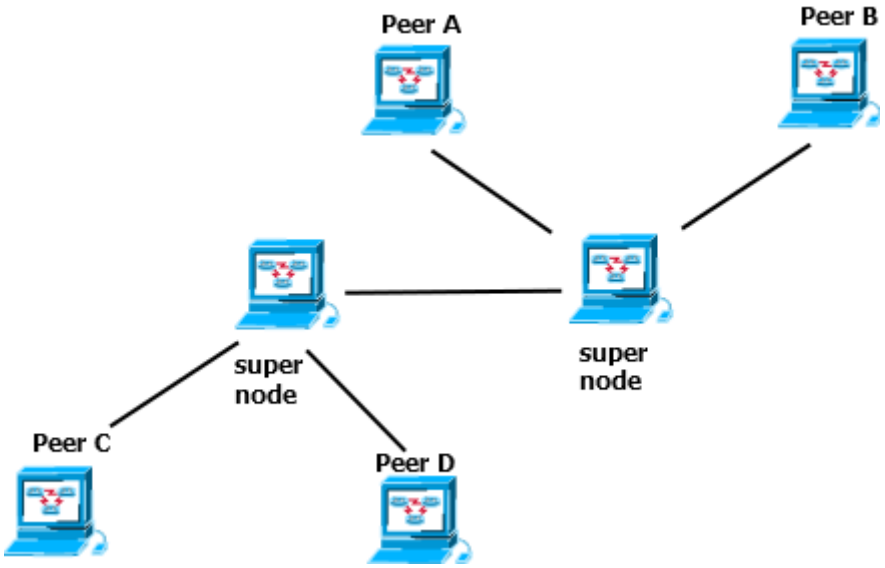

Figure 2.7: Hybrid P2P system

2.4.1.2 Software And Middleware For Peer-To-Peer and Volunteer Computing

-BOINC

BOINC [8], i.e., Berkeley Open Infrastructure for Network Computing is a distributed computing platform for volunteer computing and desktop Grid computing developed at the University of California, Berkeley. It allows volunteers to participate to many projects, i.e., volunteers control their resources among these projects. It is open-source and is available at http://boinc.berkeley.edu. It is designed to support applications that have large computation requirements. Note that the requirement of a given application is that it is divisible into a large number of jobs that can be performed independently. It provides security features, which allow to protect against attacks. For example, public-key encryption protects against the distribution of viruses. Existing applications in common languages like C/C++ or Fortran can run as a BOINC application with little or no modification.

The BOINC server software can handle millions of tasks per day. It is very efficient and

easy to increase server capacity by adding more machines. Note that, data distribution and collection in BOINC can be spread across many servers. The BOINC core client is available for several platforms like Windows, Linux, Mac OS X, etc. It provides interfaces that help developers to create software that extend BOINC and also provides web-based tools that help volunteers to form online communities.

-OurGrid

OurGrid [9] is an open source grid middleware based on a peer-to-peer architecture developed at the Federal University of Campina Grande (Brazil) (see Figure 2.8). In the sequel we will describe the main components of OurGrid : OurGrid Broker, Peers and Workers.

- **OurGrid Broker** MyGrid or the OurGrid Broker is the scheduling component of OurGrid. In particular, it is responsible of scheduling the execution of tasks and transfer data to and from grid machines since it acts as a grid coordinator during the processing of jobs. Note that, a machine running MyGrid is called the home machine. Note also that the grid configuration and the job specification are done on the home machine since it is the central point of a grid.
  The Broker provides support to execute and monitor jobs. It's the OurGrid's user frontend. The Broker gets Workers from its associated Peer during the execution of a job. Note that, a machine running OurGrid Workers is responsible of running the job processing. Note also that the Broker schedules the tasks to run on the Workers and retrieves all data to/from Workers.

- **Peers** A machine running OurGrid Peer is called a peer machine. It provides worker machines that belong to the same administrative domain. In particular, a Peer is a Worker provider that provides Workers for task execution. Note that, a Peer determines which machines can be used as workers.

- **Workers** Each machine available for task execution is used to run the OurGrid Worker. The worker provides support for fault handling and access functionality to the home machine. It allows for the use of machines in private network when combined with the OurGrid Peer. Note that, any computer connected to the Internet can be used as a worker machine. Note also that, administrative domains in Figure

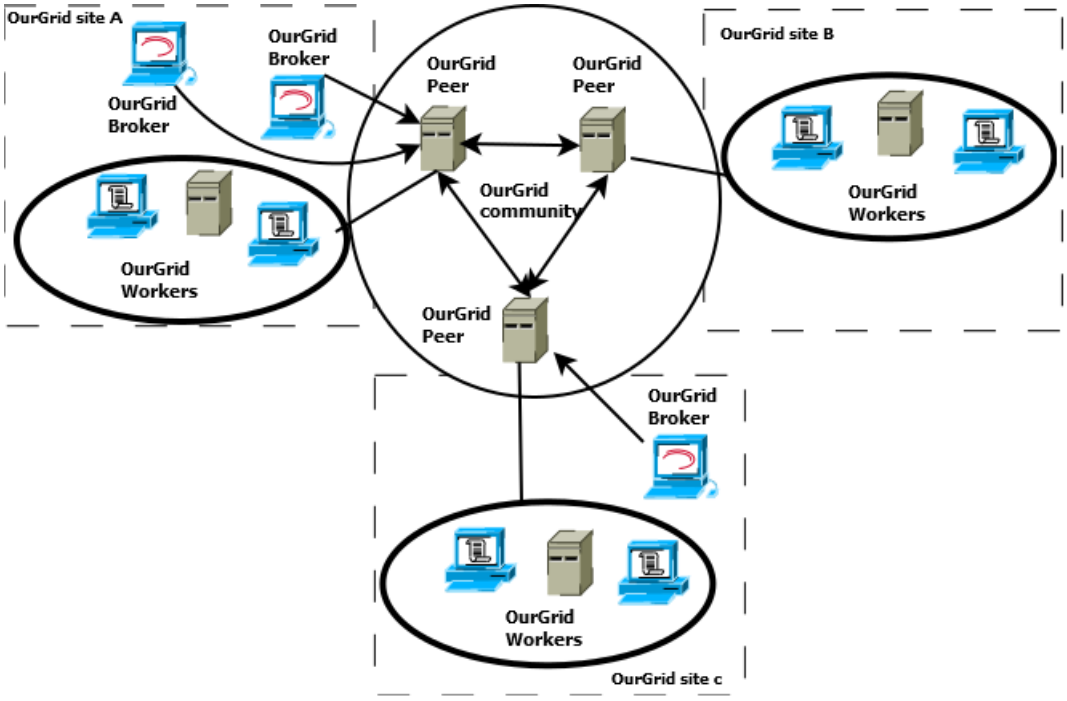

Figure 2.8: OurGrid Main Components

2.8 are illustrated as rectangles containing Workers that can uses their own intranets.

-P2PDC

The P2PDC decentralized environment [10] was designed for peer-to-peer high performance computing and distributed computing applications and more particularly loosely synchronous applications that require frequent communication between peers. It relies on the P2PSAP self adaptive communication protocol to allow direct communication between computing nodes. It is suited to the solution of large scale numerical simulation problems via distributed iterative methods. Reference is made to [10] and [11] for more details about P2PDC and P2PSAP.

-ParCop

ParCop [76] is a decentralized peer-to-peer computing system. It supports Master/-Worker style of applications which can be decomposed into independent tasks. A peer in ParCop can be a Master or a Worker, but not both at the same time (see Figure 2.9). A Master distributes tasks to workers, collects and returns the results to the user. There are two kinds of communication pathways: permanent and temporary pathways.

The permanent pathway is used to maintain the topology of Peer-to-peer overlay; while temporary pathways are used to send tasks and results between Master and Workers; it will be closed when the computation finishes.

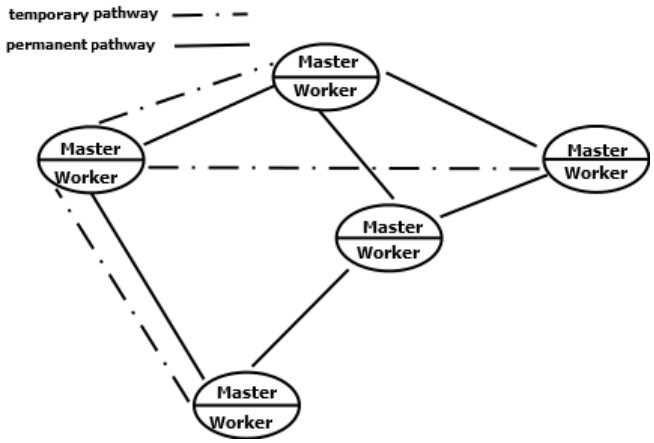

Figure 2.9: Communications between peers in ParCop

-MapReduce

MapReduce [70] is a framework for processing large datasets on large clusters. In particular, it is a programming model for generating and processing large datasets with a distributed algorithm on a large number of computers. Note that, processing can take place on data stored in a database or in a filesystem. Note also that MapReduce is extended in [71] to be used in Peer-to-Peer networks.
MapReduce framework is composed of three operations:

- **Map:** Each worker node applies the map function (that process a key/value pair) to the local data, and writes the output to a temporary storage. Note that, a master node ensures that one copy of redundant input data is processed.

- **Shuffle:** Data is redistributed by worker nodes according to the output keys generated by the map function, such that all data that have the same key belong to the same worker node.

- **Reduce:** Each group of output data is processed in parallel and per key by worker nodes.

-P2P-MPI

P2P-MPI [72] is a framework which offers a programming model based on message passing in order to execute applications on clusters (see Figure 2.10). It is developed in Java for portability purpose. It provides an MPJ [73] (Message Passing for Java) communication library and a middleware in order to manage the computing resources. Note that the middleware of P2P-MPI is based on peer-to-peer infrastructure. P2P-MPI uses a super-node to maintain and manage peers to join the P2P infrastructure. It provides a mechanism for fault-tolerance based on replication of peers. It let its users to share their CPU and access others' CPUs. It offers three separate services to the user, such as :

- **The Message Passing Daemon (MPD)** the main role of MPD is to search for participating node.
- **The File Transfer Service (FT)** the main role of FT is to transfer files between nodes.
- **The Fault Detection Service (FD)** the main role of FD is to notify the application when nodes becomes unreachable during execution.

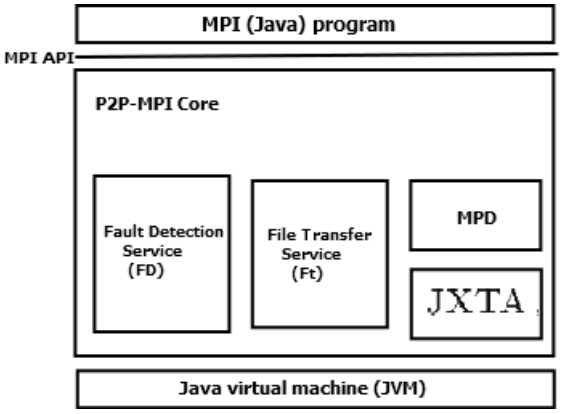

Figure 2.10: P2P-MPI structure

2.4.2 Grid Computing

Grid computing [49] is the collection of computer resources (desktop, cluster nodes or supercomputers) to handle long-running computational tasks. Grid computing has been used to solve large scale HPC applications like coupled models. Several middlewares have

been proposed to facilitate the implementation of HPC applications on grid environments like Globus, Legion, Condor, Grid'BnB, etc.

2.4.2.1 Globus

The Globus software environment [54 ; 55] is a middleware that facilitates the implementation of HPC applications on grid environments. It is composed by a set of components implementing basic services to resource allocation, security, communication and access to remote data [55 ; 56].
The Globus software environment employs a certificate approach using the protocol Secure Socket Layer (SSL) [68 ; 69] in order to ensure security.
The resource allocation component of the Globus environment, i.e., GRAM is used like an interface between local and global services. Note that a file called map-file contains information (authorized users of the grid configuration) to identify users of the grid.
A communication library called Nexus [57 ; 58] is used to perform the communication in the Globus environment. This component defines a low level API to support message passing, remote procedure call and remote I/O procedures. Note that a component called Metacomputing Directory Service (MDS) [59–61] is used to manage information about the system and the grid configuration.

2.4.2.2 Legion

Legion [13] is an open source software library for the grid computing communities. It is representative of large scale meta-computing systems. It addresses issues like heterogeneity, scalability and programmability.
The Legion software environment was developed at University of Virginia (since 1993) and acts as a object oriented system. It provides a unique virtual machine for user applications since it has an architecture concept of grid computing. The main goal of Legion is to have some concepts of a grid configuration like security, scalability and fault tolerance transparent to final users [62].
Every entity like storage capacity and RAM memory in Legion is represented as objects. A remote mechanism [62 ; 63] is used to perform the communication between objects in the Legion environment. The security component of the Legion environment is based also on an object. Note that in order to ensure more security, the Legion environment provides some extra basic mechanism like the MayI method.
An approach in the Legion environment simplifies the manipulation of files to application programmers through the combination of persistent objects with the global information

of object identification. This approach permits also users to add fault tolerance characteristics to applications using rollback and recovery mechanisms [62].

2.4.2.3 Condor software

The basic operation of Condor's [1 ; 51] can be described as follows: Users submit their jobs to Condor, and Condor chooses when and where to run them based upon a policy. In particular, Condor finds an available machine on the network and begins running the job on that machine. It can also manage wasted CPU power from idle desktop workstations across an entire organization with minimal effort. For example, Condor can be configured to run jobs on desktop workstations only when the keyboard and CPU are idle. If a job is running on a workstation when the user returns and hits a key, Condor can migrate the job to a different workstation and resume the job right where it left off [50]. We will present in the sequel the two categories of Condor : The Condor high-throughput computing system, and the Condor-G agent for grid computing.

-The Condor High Throughput Computing System

The goal of high-throughput computing environment [52] is to provide large amounts of fault tolerant computational power over long periods of time. It is achieved through opportunistic means. This requires several tools (see bellow). Note that opportunistic computing is the ability to use resources whenever they are available, without requiring one hundred percent availability.

- **ClassAds** : ClassAds system provides a flexible framework for matching resource requests, e.g. jobs with resource offers. ClassAds allows Condor to adopt a planning approach when incorporating grid resources. Note that ClassAds file describes what type of jobs they will accept and under what conditions, while those submitting jobs set their own requirements.

- **Job Checkpoint and Migration** : Condor, with certain types of jobs, can record a checkpoint and resume the application from the checkpoint file. Note that, a checkpoint permits also to a job to migrate from one machine to another machine.

- **Remote System Calls** : Remote system calls is one of Condor's mechanisms for redirecting all of a jobs I/O related system calls back to the machine which submitted the job.

-Condor-G: An Agent for Grid Computing

Condor-G [53] represents the combination of technologies from the Condor and Globus projects [12].
From Globus comes the use of protocols for secure inter-domain communications and access to a variety of remote batch systems. Condor concerns job submission, job allocation, error recovery, and creation of a friendly execution environment.

2.4.2.4 Grid'BnB

We conclude this subsection by a short presentation of Grid'BnB [15]; a parallel branch and bound framework for grids. It aims to help programmers to distribute their combinatorial optimization problems over grids by hiding grid difficulties and distribution issues. It is implemented with Java, which allows the use of many operating systems and machine architectures. Grid'BnB is implemented within the ProActive Grid middleware. ProActive [64] is a Java library for distributed computing.
The framework is built over a hierarchical master-worker approach. This approach is composed of four entities: master, sub-master, worker, and leader.
The master is the unique entry point, it gets the entire problem to solve as a single task, i.e., root task. The root task is decomposed into sub-tasks and the sub-tasks are distributed amongst a farm of workers. The root task takes care of gathering the scattered results in order to produce the final result of the computation.
The sub-masters are intermediary entities that enhance scalability. They forward subtasks from the master to workers and return results to the submitter limiting network congestion.
The worker processes run in a very simple way : they receive a message from the submaster that contains their assigned tasks, perform computations, and at the end send the result back to the sub-master, then the sub-master transfers results to the master.
Leaders are in charge of forwarding messages between clusters. Figure 2.11 shows the global architecture of Grid'BnB.

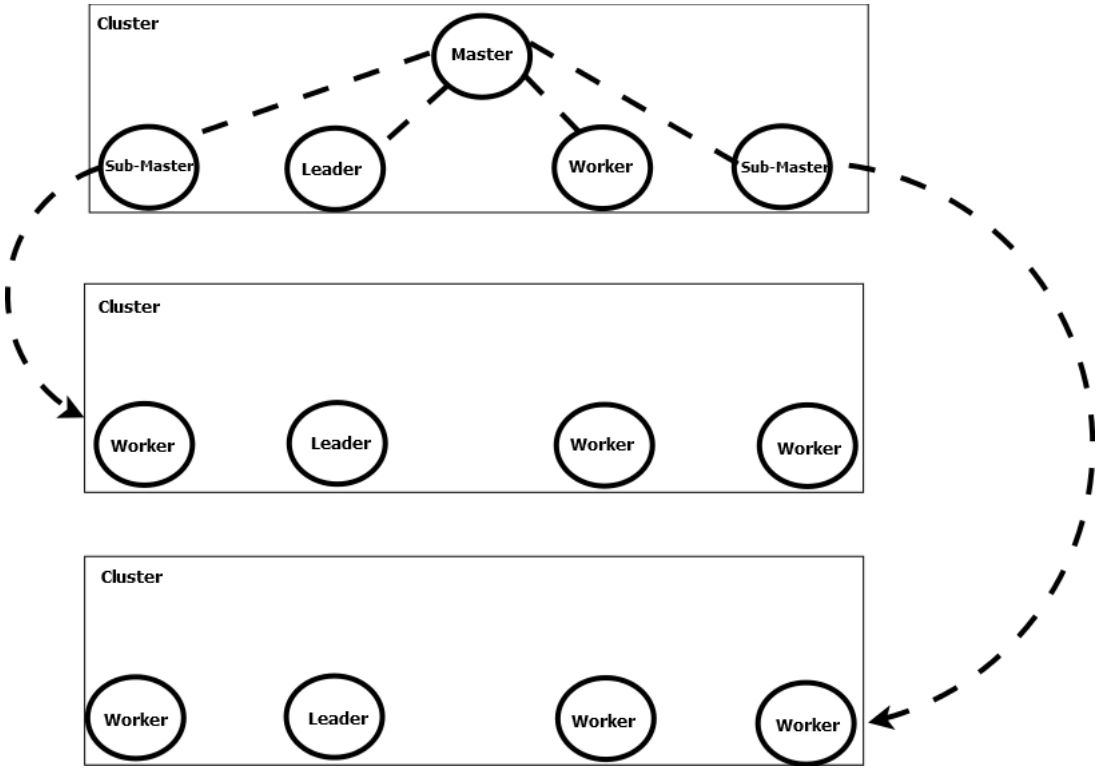

Figure 2.11: Global archictecture of Grid'BnB

2.4.3 Global Computing

Global computing systems are systems that use a set of computers connected to the internet to solve large applications. Several systems have been proposed like SETI@home [74], GENOME@home [75], XtremWeb, YML [93], etc. In this section we present in details XtremWeb.

2.4.3.1 XtremWeb

XtremWeb [14] is a middleware for desktop grid computing. It is designed to provide a Global Computing framework for solving different applications. Moreover, XtremWeb allows multi-applications, multi-users, multi-exec formats, i.e. bin,Java and multi-platforms, i.e., linux, windows, Mac OS. In contrast to Globus and Legion, XtremWeb is designed to support a very large number of personal devices across many administrative domains. XtremWeb implements three entities, the coordinator, the workers and the clients (see Figure 2.12). The role of coordinator is to manage a set of tasks provided by clients and coordinate their scheduling among a set of workers. The role of clients is to authorize users to submit their tasks to the coordinator. The role of workers is very simple, it receives a message from the coordinator that contains their assigned tasks, perform com-

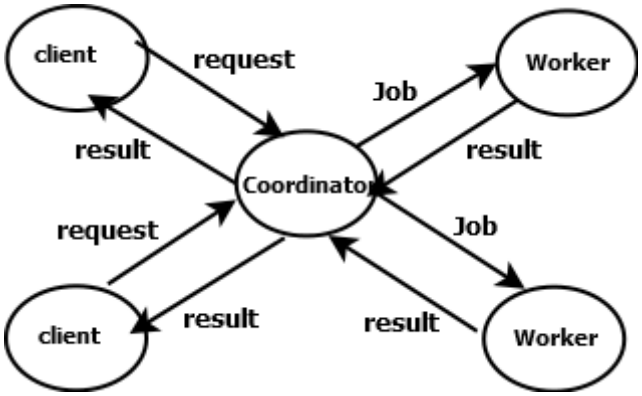

Figure 2.12: Entities of XtremWeb

putations, and at the end when the task is completed, the worker sends the results back to the coordinator.

2.5 Runtime Systems

A runtime system is like a collection of software and hardware resources that permits a program to be executed on a computer system. Several runtime systems have been proposed like HPX and StarPU in connection with HPC applications.

2.5.1 HPX Runtime System

Heterogeneous multi-core platforms, mixing cores and computing accelerators are nowadays widely spread. HPX [40–43] is a C++ runtime system for parallel applications, some of which are loosely synchronous applications. It has been developed for systems of any scale and aims to resolve the problems related to resiliency, power efficiency, etc. It has a programming model for all types of parallelism available in HPC systems which uses the available resources to attain scalability. It is portable and easy to use [44]. It is published under an open-source license and has an active, open and user community. It is built using dynamic and static data flow, fine grain future-based synchronization and continuation style programming.

The main goal of HPX is to create an open source implementation of the ParalleX execution model [45 ; 46] for conventional systems like classic Linux based Beowulf clusters, Android, Windows, Macintoch, Xeon/Phi, Bluegene/Q or multi-socket highly parallel SMP nodes.

HPXCL [47] and APEX [48] are libraries which provide additional functionality that extend the HPX. HPXCL, allows programmers to incorporate GPUs into their HPX ap-

plications. Users write an OpenCL kernel and pass it to HPXCL which manages the synchronization and data offloading of the results with the parallel execution flow on the CPUs. APEX, gathers arbitrary information about the system and uses it to make runtime-adaptive decisions based on user defined policies.

2.5.2 StarPU Runtime System

StarPU [16] is a runtime system that provides an interface to execute parallel tasks over heterogeneous multi-core architectures, i.e., multi-core processors and computing accelerators (see Figure 2.13). The main components of StarPU are a software distributed shared memory (DSM) and a scheduling framework. DSM enables task computations to overlap and avoid redundant memory transfers. The scheduling framework maintains an up-to-date and a self-tuned database of kernel performance models over the available computing tasks to guide the task mapping algorithms. Note that middle layers tools like programming environments and HPC libraries can build up on top of StarPU to allows programmers to make existing applications exploit different computing accelerators with limited effort.

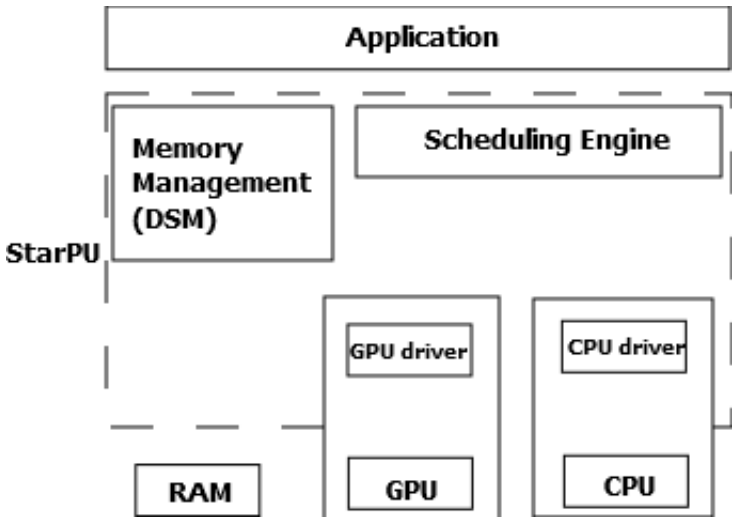

Figure 2.13: StarPU Runtime System

2.6 Conclusion

In the past, supercomputers were the leaders in the field of computing, but due to the fact that they are very expensive and consume a lot of energy; new concepts have been proposed like Peer-to-Peer, volunteer, grid and global computing. In this chapter, we

have presented a short introduction to Grid computing. We showed that Peer-to-Peer, volunteer, grid and global computing, share the same goal: use a large sets of distributed resources. In addition, we presented an overview on existing softwares and middlewares for each concept. We then presented runtime systems that are in connection with HPC applications.
In the sequel, we present our contributions to Grid computing. In particular, we show how we have extended the P2PSAP communication protocol and the P2PDC decentralized environment for Peer-to-Peer computing in order to take into account multi-network configurations as well as multi-core processing.

CHAPTER 3

Reconfigurable Multi-Network Protocol

Contents

3.1 Introduction

In this chapter, we present the Reconfigurable Multi-Network Protocol (RMNP) dedicated to high performance computing applications carried out in multi-network configurations like Ethernet, Infiniband and Myrinet. In particular, we concentrate on loosely synchronous applications like numerical simulation and optimization problems solved via iterative methods [1] that require frequent data exchanges between computing nodes. The RMNP communication protocol is designed to permit update exchanges between computing nodes with multi-network configurations like Ethernet and Infiniband or Ethernet and Myrinet. The protocol can configure itself automatically and dynamically in function of application requirements like schemes of computation, i.e., synchronous or asynchronous iterative schemes, elements of context like network topology and type of network by choosing the best communication mode between computing nodes and the best network. The protocol is an extension of the Configurable Transport Protocol (CTP) [30] and makes use of the Cactus framework [31] and micro-protocols. We note that the RMNP communication protocol will be used in GRIDHPC, our proposed decentralized environment, described in the next chapter. Our main contribution consists in managing several network adapters in order to switch between them. As a matter of fact, in the case of Grid computing, it is very important to use many types of fast networks and have access to several clusters and many computing nodes.

This chapter is organized as follows. Next section presents micro protocols, Cactus framework and CTP communication protocol. Section 3.3 describes the architecture of RMNP communication protocol. Section 3.4 depicts the RMNP mechanisms to support communication in a multi-cluster and grid context. The choice of networks and an example of scenario that shows the automatic and dynamic configuration capability of RMNP are presented in section 3.4. Finally, section 3.5 concludes this chapter.

3.2 Micro protocols

Micro-protocols are structured as a collection of event handlers, which are procedure-like segments of code and are bound to events. When an event occurs, all handlers bound to that event are executed. Micro-protocols were first introduced in X-kernel [32]. They have been widely used since then in several systems. A micro protocol is a primitive building block that implements merely a functionality of a given protocol such as error recovery, ordered delivery and so on. A protocol then results from the composition of a given set of micro-protocols. This approach permits one to reuse the code, facilitate the design of

new protocols and give the possibility to configure the protocol dynamically.

Several protocol composition frameworks have been proposed in order to deploy communication architectures. We can divide these frameworks according to three models: hierarchical, non hierarchical and hybrid models.

In the hierarchical model, a stack of micro protocols composes a given protocol. This model can be found in the X-kernel [32] and APPIA [33].

- The X-kernel is an object-based framework for implementing network protocols. It defines an interface that protocols use to invoke operations on one another (i.e., to send a message to and receive a message from an adjacent protocol) and a collection of libraries for manipulating messages, participant addresses, events, associative memory tables (maps), threads, and so on. The suite of protocols in X-kernel is statically configured at initialization time onto a protocol graph. Based on the protocol graph, users can plug protocols together in different ways.

- APPIA is a protocol kernel that supports applications requiring multiple coordinated channels. It offers a clean and elegant way for the application to express inter-channel constraints; for example, all channels should provide consistent information about the failures of remote nodes. These constraints can be implemented as protocol layers that can be dynamically combined with other protocol layers. In APPIA, micro-protocols are defined as layers that exchange information using events. A session is an instance of micro-protocols and maintains state variables to process events. A Quality of Service (QoS) is defined as a stack of layers and specifies which protocols must act on the messages and the order they must be traversed. A channel is an instantiation of a QoS and is characterized by a stack of sessions of the corresponding layers. Inter-channel coordination can be achieved by letting different channels share one or more common sessions.

In the non hierarchical model, there is no particular order between micro-protocols. The Coyote [34], ADAPTIVE [35] and SAMOA [79] frameworks correspond to this model.

- Coyote is a system that supports the construction of highly modular and configurable versions of such abstractions. It extends the notion of protocol objects and hierarchical composition found in existing systems with support for finer-grain micro-protocol objects and a non hierarchical composition scheme for use within a single layer of a protocol stack.

- ADAPTIVE (Dynamically Assembled Protocol Transformation, Integration and evaluation Environment)provides an integrated environment for developing and experimenting with flexible transport system architectures that support lightweight

and adaptive communication protocols for diverse multimedia applications running on high-performance computing.

- SAMOA is a protocol framework that ensures the isolation property. It has been designed to allow concurrent protocols to be expressed without explicit low-level synchronization, thus making programming easier and less error-prone. In SAMOA, a micro-protocol is composed of a set of event handlers and a local state. A local state of a given micro-protocol can be modified only by event handlers of this micro-protocol.

The hybrid model is a combination of the two previous models; micro-protocols are composed hierarchically and non hierarchically. We can find this model in the FPTP [36] and Cactus frameworks [31]. We recall that RMNP protocol is based on the Cactus framework since this approach is flexible and efficient. We shall detail the Cactus framework in the next subsection.

- FPTP is a connection and messages oriented transport protocol that offers a partially ordered, partially reliable, congestion controlled and timed-controlled end-to-end communication service. It has been designed to be statically or dynamically configured according the QoS requirements. It is constructed by the composition of configurable mechanisms suited to control and manage the QoS.

3.2.1 Cactus framework and CTP protocol

The Cactus framework [31] extends X-kernel in providing a finer granularity of composition. The Cactus framework makes use of micro-protocols to allow users to construct highly-configurable protocols for networked and distributed systems. It has two grain levels. Individual protocols, i.e., composite protocols, are constructed from micro-protocols. Composite protocols are then layered on top of each other to create a protocol stack using an interface similar to the X-kernel API. Protocols developed using Cactus framework can reconfigure by substituting micro-protocols or composite protocols.

Cactus is an event-based framework. Events represent state changes, such as arrival of messages from the network. Each micro-protocol is structured as a collection of event handlers, which are procedure-like segments of code and are bound to events. When an event occurs, all handlers bound to that event are executed.

The Cactus framework provides a message abstraction named dynamic messages, which is a generalization of traditional message headers. A dynamic message consists of a message body and an arbitrary set of named message attributes. Micro-protocols can add,

read, and delete message attributes. When a message is passed to a lower level protocol, a pack routine combines message attributes with the message body; while an analogous unpack routine extracts message attributes when a message is passed to a higher-level protocol. Cactus also supports shared data that can be accessed by all micro-protocols configured in a composite protocol. For more details about Cactus, reference is made to [31].

The CTP Communication Protocol [30] was designed and implemented using the Cactus framework. Figure 3.1 shows the CTP implementation with events on the right side and micro-protocols on the left side. An arrow from a micro-protocol to a given event indicates that the micro-protocol binds a handler to that event.

The CTP protocol includes a wide range of micro-protocols including a small set of basic micro-protocols like Transport Driver, Fixed Size or Resize and Checksum that are needed in every configuration and a set of micro-protocols implementing various transport properties like acknowledgments, i.e. PositiveAck, NegativeAck and DuplicateAck, retransmissions, i.e. Retransmit, forward error correction, i.e. ForwardErrorConncetion, and congestion control, i.e. WindowedCongestionControl and TCPCongestionAvoidance.

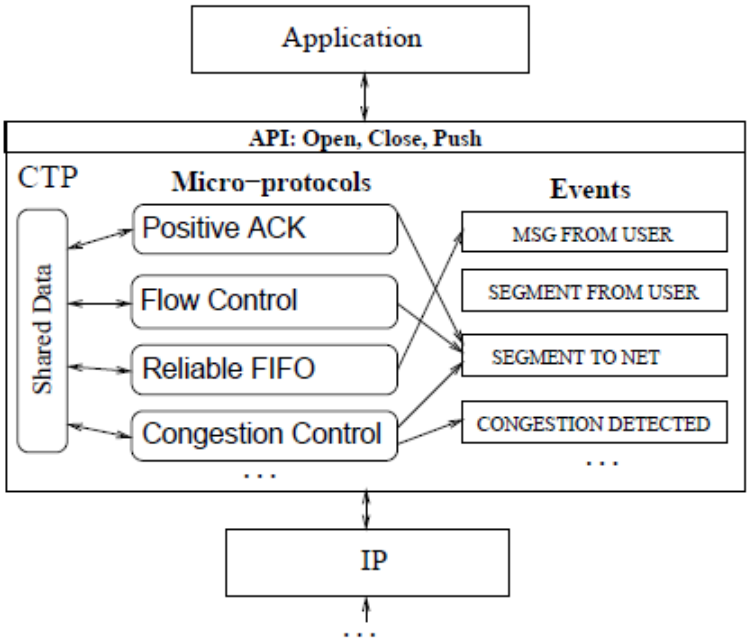

Figure 3.1: CTP-Configurable Transport Protocol

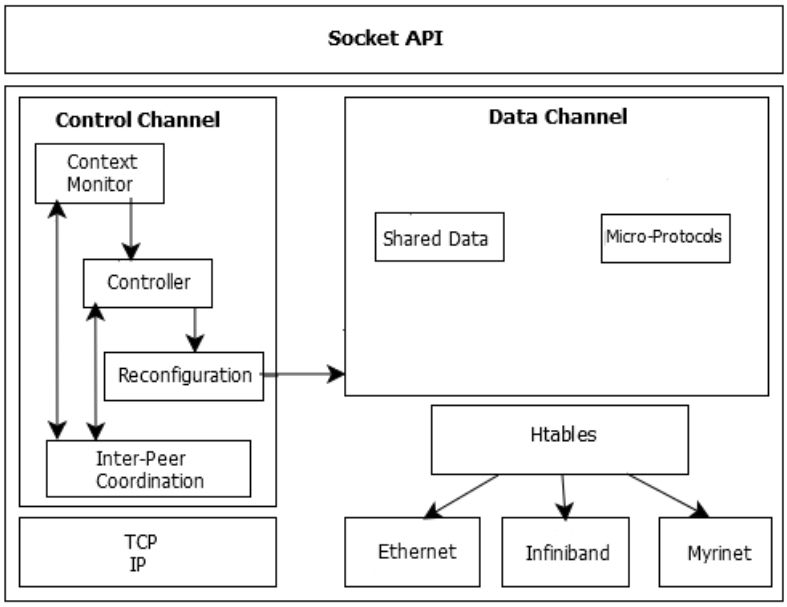

Figure 3.2: Architecture of RMNP Protocol

3.3 Reconfigurable multi-network Protocol RMNP

This sub-section presents my contribution to the design and development of the reconfigurable multi-network protocol RMNP. It aims at enabling an efficient use of the complete set of underlying communication softwares and hardwares available in multi-network systems that use Ethernet, Infiniband and Myrinet network. It is able to deal with several networks via the management of several networks adapters. The user application can dynamically switch from one network to another, according to the communication needs. Figure 3.2 shows the architecture of the RMNP protocol; it consists of a socket interface, Htable, data channel and control channel.

3.3.1 Socket Interface

The reconfigurable multi-network protocol RMNP has a Socket API at the top layer in order to facilitate programming. Application can open and close connection, send and receive data. Moreover, application will be able to get session state and change session behavior or architecture through socket options. Session management commands like listen, open, close, setsockoption and getsockoption are directed to Control channel; while data exchanges commands, i.e., send and receive commands are directed to data channel.

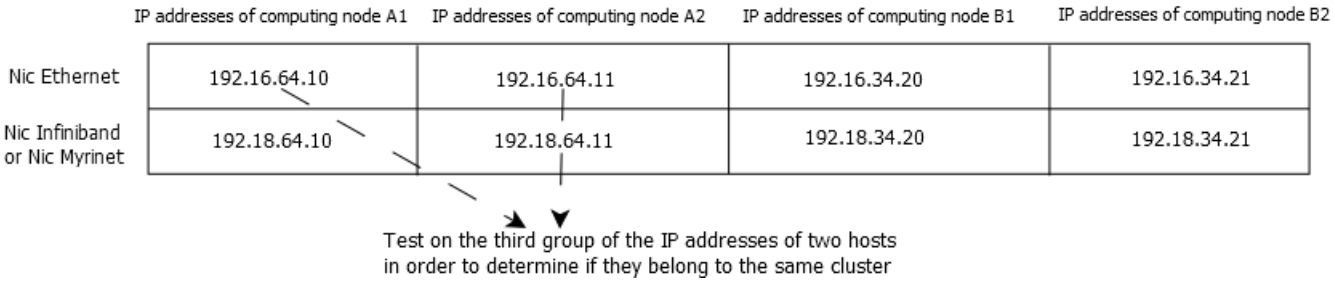

Table 3.1: Example of content of a simplified Htable and test on the location of the hosts thanks to comparison of IP addresses

3.3.2 Htable

3.3.2.1 Distance metric

Distance metric is based on IP address. In particular, it concentrates on the third group of the IP address. For example, in the case of three computing nodes: A1 having IP address 192.16.64.10, A2 having IP address 192.16.64.11 and B1 having IP address 192.16.34.20. The value of third group of A1 and A2 is equal to 64, while the value of third group of B1 is 34. Consequently, it is deduced that A1 and A2 are in the same location, e.g., cluster while A1 and B1 are in different locations, e.g., different clusters (see Table 3.1).

Algorithm 1: Get the value of third group of a given IP address

function GetValThirdGroup ($addr$);
Input : addr : Array of string of a given IP address
Output: The value of the third group of the IP address
$del \leftarrow$ "." ;
$x \leftarrow 0$;
`/* strtok breaks string addr into a series of substringaddr using the delimiter del. */`
$substringaddr \leftarrow strtok(addr, del)$;
while $substringaddr$!= $NULL$ **do**
  increase x by 1;
  $substringaddr \leftarrow strtok(NULL, del)$;
  if $x == 2$ **then**
    break;
return $substringaddr$;

Algorithm 2: Test locality and choose the best network

function Localityandbestnet $(laddress, raddress)$;

Input : laddress and raddress : Ethernet IP addresses of local and remote computing nodes

```
/* inet_ntop converts the network address structure into a character
   string                                                          */
```

$inet\_ntop(AF\_INET, \&(laddress.sin\_addr), l, 80)$;

$inet\_ntop(AF\_INET, \&(raddress.sin\_addr), r, 80)$;

$substringl \leftarrow GetValThirdGroup(l)$;

$substringr \leftarrow GetValThirdGroup(r)$;

if *strcmp(substringl, substringr) == 0* **then**

```
    /* Local and Remote computing nodes belong to the same cluster  */
```

$Bestladdress \leftarrow Get(laddress)$;

```
    /* Get function get the best interface network from Htable, i.e.,
       second line if any                                           */
```

$Bestraddress \leftarrow Get(raddress)$;

```
    /* Communication between computing nodes are made via the best
       network (Infiniband or Myrinet Network), i.e., Bestladdress and
       Bestraddress                                                 */
```

else

```
    /* Local and Remote computing nodes are in different cluster,
       Hence communication between them are made via Ethernet Network,
       i.e., laddress and raddress                                  */
```

3.3.2.2 Definition of Htable

We give here a simplified presentation of Htable. We note that the Htable contains information regarding resources such as IP addresses. In this chapter, we present the network resources and in next chapter we present the CPU core resources of the actual Htable. This component is designed to manage several network adapters within the same application session. In particular, Htable permits each computing node to switch between the networks according to the communication needs.

Several network interface cards (NICs) are added to the interface of RMNP and information about these NICs are stored in the Htable. In the Htable (see Table 3.1), the IP addresses that are given on the first line correspond to Ethernet network and the IP

addresses given on the second line, if any, correspond to fast network like Infiniband or Myrinet. Algorithm 1 and Algorithm 2 are used to control multiple networks. In particular, Algorithm 1 is used to return the third value of a given IP address in order to treat issues related to the locality of several computing nodes, i.e., to test if several computing nodes belong to the same cluster or not; while Algorithm 2 is used to return the best network interface card from the Htable according to the locality of machines.

3.3.3 Data channel

The main function of the Data channel is to transfer data packets between computing nodes; it is built using the Cactus framework. The data channel has two levels : the physical layer and the transport layer; each layer corresponds to a Cactus composite protocol. The physical layer supports communication on different networks, i.e., Ethernet, Infiniband and Myrinet thanks to the concept of Htable. The transport layer is constituted by a composite protocol made of several micro-protocols. At this level, data channel reconfiguration is carried out by substituting micro-protocols. We note that the choice of micro-protocols in the transport layer depends on the context, i.e., the type of iterative schemes, e.g., synchronous or asynchronous, the location of machines, e.g., intra or inter cluster and the type of underlying network, i.e., Ethernet, Infiniband or Myrinet. Decision rules are summarized in Table 3.2. In the sequel (subsection 3.4.4) we explain those rules. We note also that the behavior of the data channel is triggered by the control channel.

3.3.4 Control channel

The Control channel manages session opening and closure; it captures context information and (re)configures the data channel at opening or operation time; it adapts itself to this information and their changes; it is also responsible for coordination between computing nodes during the reconfiguration process. Note that we use the TCP protocol [37] to exchange control messages since these messages cannot be lost.

Before describing the main components of the control channel, we present first a session life cycle, see Figure 3.3. Suppose process A wants to exchange data with process B, it opens a session through socket create and connect command. Then, a TCP connection is opened between the two processes. Process B accepts connection and must send its context information to process A. Process A chooses the most appropriate configuration like the choice of micro-protocols and underlying network at transport and physical layer of data channel and sends configuration command to process B based on its context information and the context of B. After that, the two processes carry out the configuration

of data channel. When the configuration is done, each process has to inform the other processes and waits for the notification of other processes. Data is exchanged only when both processes have finished data channel configuration. During the communication, a process can decide, e.g, introduce some synchronization in an algorithm that is otherwise asynchronous to change configuration of data channel due to context changes or user requirements, like process A in Figure 3.3. Then, a procedure similar to the one implemented for configuration at session opening will be realized. When session is closed, the data channel is closed first; the control channel with TCP connection is closed later on.

We describe now the main components of the control channel.

- **Context monitor :** the context monitor collects context data and their changes. Protocol adaptation is based on context acquisition, data aggregation and data interpretation. Context data can be requirements imposed by the user or the algorithm at the application level, i.e. asynchronous iterative algorithms, synchronous iterative algorithms or hybrid methods (see chapter V for details). Context data can also be related to computing nodes location and machine loads. Context data are collected at specific times or by means of triggers. Data collected by the context monitor can be referenced by the controller.

- **Controller :** the controller is the most important component in the control channel; it manages session opening and closure through TCP connection opening and closure; it also combines and analyzes context information provided by the context monitor so as to choose the configuration (at session opening) or to take reconfiguration decision (during session operation) for data channel. The (re)configuration command along with necessary information is sent to component Reconfiguration and to other communication end point.

- **Reconfiguration :** reconfiguration actions are made by the reconfiguration component via the dedicated Cactus functions. Reconfiguration is done at the physical layer of data channel by substituting a type of network, e.g., Ethernet network to Infiniband network.

- **Inter-node coordination :** the coordination component is responsible of context information exchanges and coordination of computing nodes reconfiguration processes so as to ensure proper working of the communication protocol.

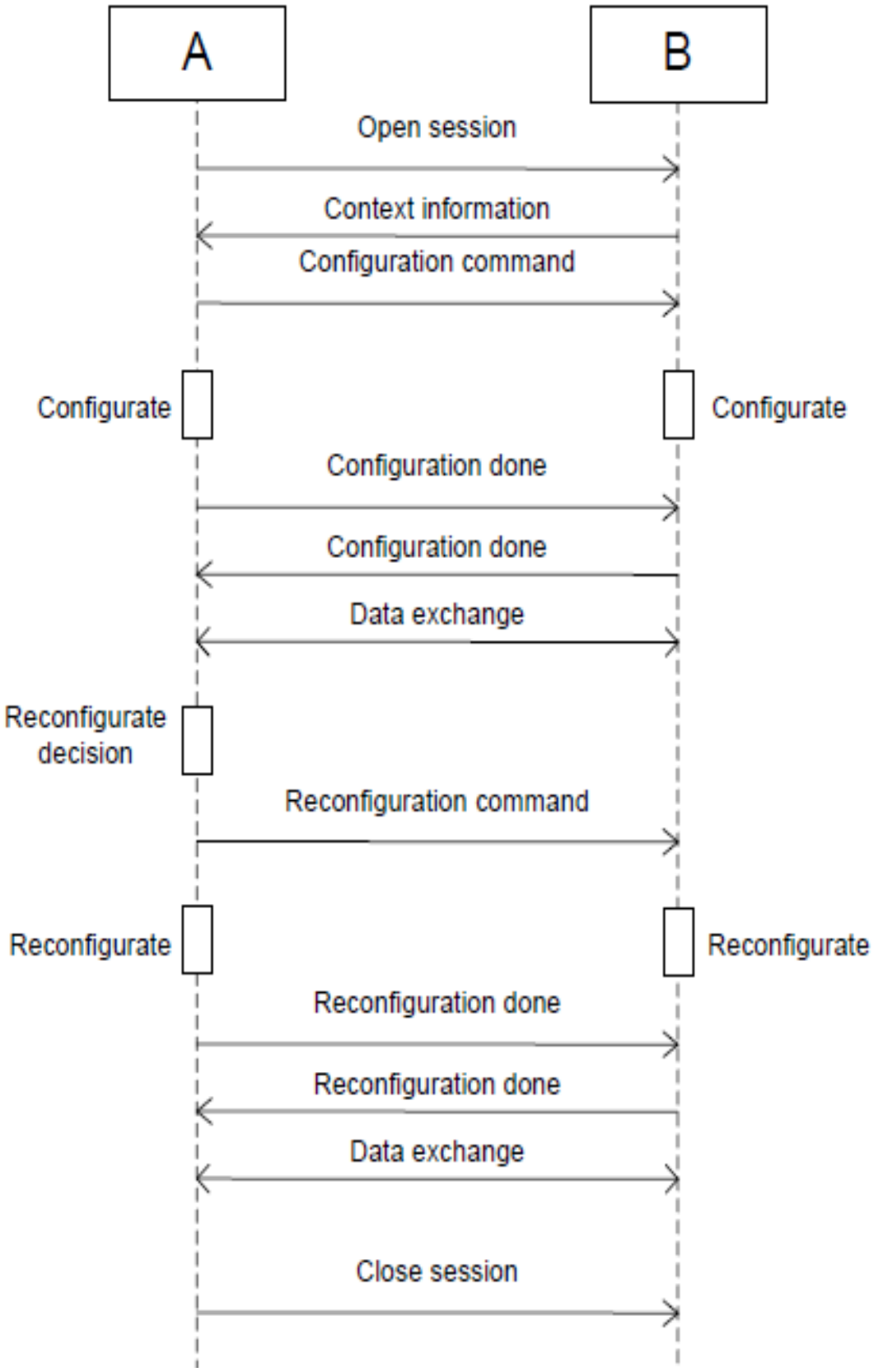

Figure 3.3: Protocol session life cycle

3.4 RMNP mechanisms

3.4.1 Heterogeneous Multi-Cluster Environment

Figure 3.4 displays a multi-cluster environment in Grid5000 [3] where a cluster consists of interconnected stand-alone computers or computing nodes that can work cooperatively as a single integrated computing resource. In particular, figure 3.4 shows the architecture of typical clusters built around a low latency and high bandwidth interconnection network. For example, cluster A, i.e., cluster Graphene in Nancy site with Infiniband and cluster B, i.e., cluster Chinqchint in Lille site with Myrinet of Grid5000. The network can be as simple as a SAN, e.g., Myrinet or a LAN, e.g., Ethernet. The feature that can be identified to any multi-cluster context is for example that a computing node denoted by A1 of the cluster A in Figure 3.4 is built around Infiniband and Ethernet networks. Hence, supporting heterogeneous multi-cluster mainly consist in integrating functionality in order to switch from one network to another, according to the communication needs.

3.4.2 Choice of networks

The network management procedure has two steps. First step corresponds to the test of the locality between the computing nodes and the second step corresponds to the choice of the appropriate network for data exchange depending on the locality of computing nodes. The locality test is based on comparing the IP addresses of two computing nodes and according to this comparison, we deduce if the computing nodes are in the same cluster or not. The second step is based on choosing the best interface network (high speed and low latency network) from the Htable according to the result of the locality test. Consequently, if the locality test returns that the considered computing nodes are in different locations, e.g., clusters, then the Ethernet network interface is chosen from the Htable to perform the communication between the two computing nodes. If the locality test returns that the computing nodes belong to the same location, then the best network interface in the Htable is selected, e.g., Infiniband or Myrinet.

3.4.3 Example of scenario

We present now a simple scenario for the RMNP protocol so as to illustrate its behavior. We consider a high performance computing application, like for instance a large scale numerical simulation application, solved on the network composed of two clusters shown in Figure 3.4. Computing nodes in cluster A own both a Fast-Ethernet card, i.e., 192.16.64.x and a Infiniband card, i.e., 192.18.64.x (see Table 3.1) and computing nodes in cluster

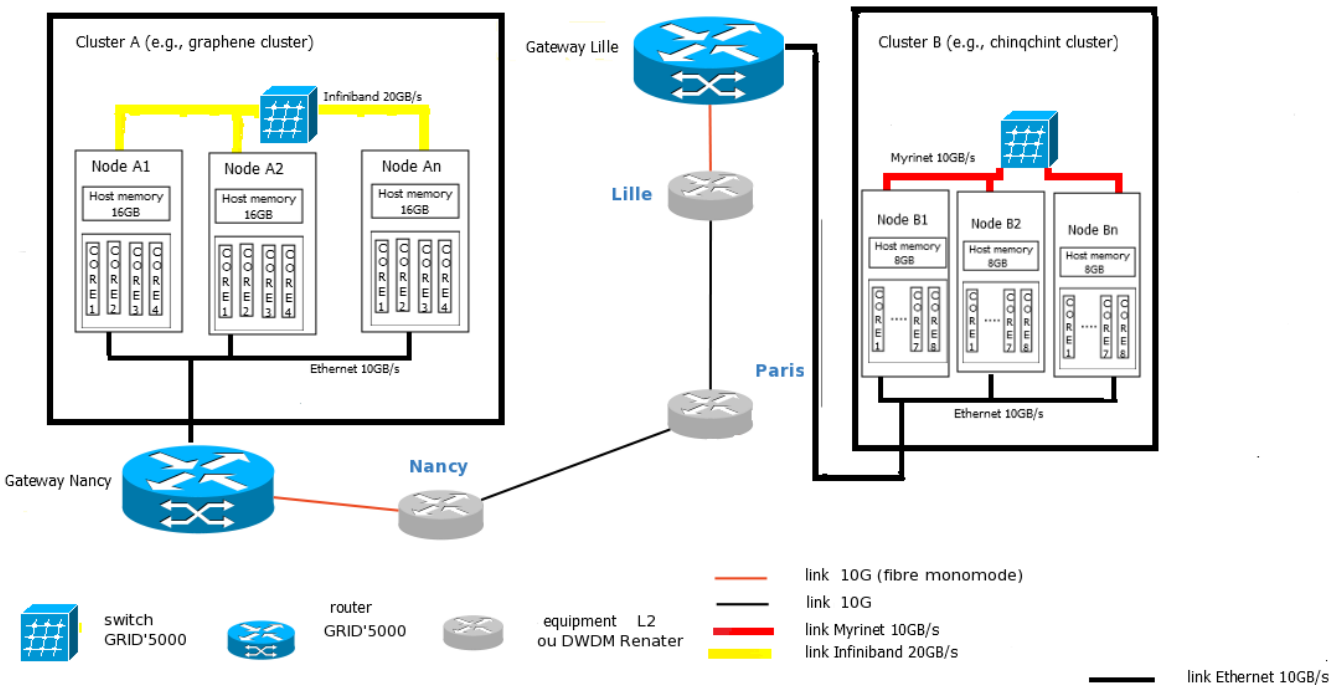

Figure 3.4: Multi-Cluster platform in Grid5000

B own both a Fast-Ethernet card, i.e., 192.16.34.x and a Myrinet card, i.e., 192.18.34.x where x is a value between 1 and 255. We suppose that we have a communication network between computing nodes like $A1 \leftrightarrow A2$, $A2 \leftrightarrow B1$ and $B1 \leftrightarrow B2$ where $X \leftrightarrow Y$ means that there is bidirectional link between X and Y. The value of third group of A1 and A2 is equal to 64. Consequently, the communications between the considered computing nodes is made via Infiniband network. The values of third group of A2 and B1 are 64 and 34, respectively. Consequently, the communications between the considered computing nodes is made via Ethernet network. The value of third group of B1 and B2 is equal to 34. Consequently, the communications between the considered computing nodes is made via Myrinet network (see Table 3.1)

3.4.4 Choice of Micro-protocols

As mentioned earlier (see subsection 3.3.3), the choice of micro-protocols depends on : the scheme of computation (synchronous, asynchronous or hybrid iterations), the type of connection (intra or inter cluster) and the type of underlying network (Ethernet, Infiniband or Myrinet). We note that Infiniband and Myrinet insure reliability and message order. As a consequence, the data channel needs only micro-protocols to ensure synchronous or asynchronous communication, transport drive and segment size management (Resize),

see Table 3.2. We explain now the choice of micro-protocols for each context and that is presented in Table 3.2.

In the case where asynchronous iterative schemes of computation are required by user, asynchronous communication operations must be preferably implemented in both intra-cluster and inter-cluster data exchanges. In this case, micro-protocol used for the data channel will be the asynchronous micro-protocol. We note that asynchronous iterative schemes of computation are fault tolerant in some sense since they allow messages losses. For this reason, reliable transport and ordered delivery as well as flow control are not needed in both intra-cluster and inter-cluster communication. While congestion control in intra cluster communication with low latency, high bandwidth is not necessary, it is required in inter cluster communication with high latency, low bandwidth and unreliable link in order to behave fairly with other flows. This is ensured via DCCPAck, DCCP Window Congestion Control [39] and TCP Congestion avoidance micro-protocols (see Table 3.2).

In the case where synchronous iterative schemes of computation are required by user, synchronous communication must be imposed in both intra-cluster and inter-cluster data exchanges. In this case, micro-protocol used for the data channel will be the synchronous micro-protocol. This synchronous context requires reliability and order delivery in order to ensure that the application is not going to be blocked by a message loss or unordered message delivery. This is ensured via SequencedSegment, Retransmit, TRREstimation, PositiveAck, DuplicateAck and ReliableFifo micro-protocols (see Table 3.2). Moreover, in synchronous communication, after sending a message, the sender is blocked until it receives an acknowledgment about the delivery of this message to application at receiver. Thus, the receiver buffer cannot be overwhelmed, and flow control is not necessary in both intra-cluster and inter-cluster communication. While congestion control in intra cluster communication with low latency, high bandwidth is not necessary, it is required in inter cluster communication with high latency, low bandwidth and unreliable link in order to behave fairly with others flows and to reduce retransmission overhead. This is ensured via Window Congestion Control and TCP New-Reno congestion avoidance [38] micro-protocols (see Table 3.2).

In the case where Hybrid iterative schemes of computation, i.e., combination of synchronous iterative (locally, e.g., same cluster) and asynchronous iterative computational schemes (globally) are required by user, synchronous communication must be imposed in intra-cluster and asynchronous communication must be imposed in inter-cluster data exchanges. We note that the communication protocol in the context of intra-cluster has the same features as in the case of synchronous iterative scheme and intra-cluster com-

Micro-Protocols	Synchronous				Asynchronous				Hybrid			
	Intra			Inter	Intra			Inter	Intra			Inter
	ETH	IB	MX	ETH	ETH	IB	MX	ETH	ETH	IB	MX	ETH	IB	MX
Transport Drive	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Resize	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Synchronous	X	X	X	X					X	X	X			
Asynchronous					X	X	X	X				X	X	X
SequencedSegment	X			X					X					
Retransmit	X			X					X					
PositiveAck	X			X					X					
RTTEstimation	X			X					X					
ReliableFifo	X			X					X					
DuplicateAck				X										
WindowedCongestioncontrol				X										
TCPNewRenoCongestionAvoidance				X										
DCCPAck								X				X		
DCCPWindowedCongestionControl								X				X		
TCPCongestionAvoidance								X				X		

Table 3.2: Choice of micro-protocols for each considered context

munication. The communication protocol in the context of inter-cluster has the same features as in the case of asynchronous iterative scheme and inter-cluster communication.

We shall present in the sequel the features of some micro-protocols like synchronous, asynchronous, buffer management, Transport drive and Resize micro-protcols.

3.4.4.1 Synchronous and Asynchronous micro-protocols

In this subsection, we present two micro-protocols corresponding to two communication modes : synchronous and asynchronous. The synchronous micro-protocol implements blocking synchronous communication mode. It consists of three handlers for three events : UserSend, SegmentToNet and UserReceive. The asynchronous mode is implemented by the asynchronous micro-protocol and it consists of two handlers for two events : UserSend and UserReceive. These events will be raised when send and receive socket commands are called by an application. In response to message sent from application, theses micro-protocols may return the control to application immediately after message sent, i.e., asynchronous send or wait for an acknowledgment indicating that message was received by receiver side application, i.e., synchronous send.

3.4.4.2 Buffer management micro-protocol

There are two buffers to manage : send buffer and receive buffer. Send buffer stores messages waiting to be sent or waiting for an acknowledgment from receiver application. Receive buffer stores messages which arrive from network and waits to be received by application. Note that, this micro-protocol implements handlers for the UserSend and MsgFromNet events to catch the messages from application and from network.

3.4.4.3 Transport Drive and Resize micro-protocols

These micro-protocols are needed in every configuration. The transport Drive adds port identifiers on all outgoing segments for demultiplexing. It sets the bits that are sent to ensure that messages are sent even if there are no other micro-protocols that set the send bits in the configuration. The micro protocol Resize makes the fragmentation of an applicative message larger than the MTU (8100 bytes) at emission and reassembles segments in order at reception.

3.5 Conclusion

In this chapter, we give a state of the art about Micro-protocols and Cactus frameworks. Afterward, we describe the global architecture of the Reconfigurable multi-network Protocol RMNP for HPC applications with its main functionalities to allow update exchange between computing nodes in multiple network configurations (Ethernet, Infinband and Myrinet) via distributed iterative algorithms. RMNP can configure itself automatically and dynamically in function of application requirements like schemes of computation, e.g. synchronous iterations or asynchronous iterations, elements of context like network topology and type of network like Ethernet, Infiniband and Myrinet by choosing the best communication mode between computing nodes and the best network. In particular, we present the RMNP mechanisms like Htable that we have designed in order to permit one to choose the best networks and the best micro-protocols for the configuration of data channel according to a given context. We note that a context is a combination of schemes of computation, i.e., synchronous, asynchronous or hybrid iterative schemes, type of network and connection, i.e., intra or inter cluster.
In the next chapter, we shall present the decentralized environment GRIDHPC that makes use of RMNP protocol in order to facilitate implementation of HPC applications.

CHAPTER 4

Decentralized Environment GRIDHPC

Contents

4.1 Introduction

In this chapter, we propose the decentralized environment GRIDHPC designed to provide an efficient, scalable and portable support for High Performance Computing (HPC) and distributed computing applications on grid platform. HPC applications that we consider are basically loosely synchronous application like numerical simulation problems or optimization problems solved via iterative methods. We consider also pleasingly parallel application like planning problems. We define the global topology of GRIDHPC with its mains functionalities. Then, we present the task assignation and the programming model. The decentralized environment GRIDHPC facilitates the use of large scale distributed systems and the work of programmer. In particular it uses a limited number of communication operations (GRID_Send, GRID_Receive and GRID_wait). In the sequel we give more details about communication operations.

A first version of the environment called P2PDC was proposed in 2008 [11 ; 17 ; 28]. This environment was dedicated to peer-to-peer computing and presented several limitations like the use of the sole Ethernet network and single core CPUs. As an attempt to overcome P2PDC limitations and to take benefit of recent multi-core processors and multi-network, and in order to reduce the solution time to solve HPC applications, and for grid computing purpose, the decentralized environment GRIDHPC is presented in this chapter. Our first objective is to use simultaneously several networks like Ethernet, Infiniband and Myrinet. This feature is very important since we consider in particular loosely synchronous applications that present frequent data exchanges between computing nodes. Consequently, we privilegiate to use several high speed networks simultaneously in the same application session. Note that the reconfigurable multi-network protocol RMNP supports data exchanges via multi-network configuration. The second objective is to use the computing resources of modern muli-core CPUs, i.e., many CPU cores. This objective is done via OpenMP [2]. As a conclusion, GRIDHPC facilitates the use of multi-cluster and grid platform for loosely synchronous applications and also embarrassingly parallel application. It exploits all the computing resources (all the available cores of computing nodes) as well as several types of networks like Ethernet, Infiniband and Myrinet in the same application.

The remainder of this chapter is organized as follows: Section 4.2 presents the main features of GRIDHPC. Section 4.3 presents the general topology architecture of GRIDHPC. Section 4.4 presents the architecture and the main functionality of GRIDHPC. Section 4.5 deals with task assignation. A programming model is proposed in section 4.6. Section 4.7 deals with developing HPC applications using GRIDHPC. Finally, section 4.8 concludes

this chapter.

4.2 Main features of GRIDHPC

In this section, we define the main features of GRIDHPC :

- Hierarchical master-worker architecture with communication between computing nodes (see section 4.5). The problem to solve is generated as a set of tasks and each task is associated to a sub-problem.
- Organizing workers in groups to optimize inter-cluster communications and avoid bottleneck (see section 4.5).
- Exploits all the computing resources: exploits all the available cores of computing nodes using OpenMP and search the best underlying network (high speed and low latency network like Infiniband and Myirnet) to perform intra cluster communications between computing nodes via multi-network configurations using RMNP.
- Facilitate programming by hiding the choice of communication mode (see section 4.7).

4.3 Global Topology of GRIDHPC

4.3.1 General topology architecture

Figure 4.1 illustrates the general topology architecture of GRIDHPC. It consists of a Submitter, Server, Trackers and Computing nodes.

- **Submitter** is the root task. It decomposes task into sub-tasks and distributes them amongst a farm of computing nodes.
- **Server** takes care of information regarding trackers and computing resources connection/disconnection; In particular, it stores in the Htable information that contains IP addresses and number of CPU cores of each computing node that joins the underlying network (Ethernet, Infiniband or Myrinet).
- **Tracker** takes care of information regarding a zone, i.e., set of computing nodes. In particular, it collects information regarding IP addresses and number of CPU cores used by each computing node in its zone and sends these data to the server. Note that trackers topology is a line. Each tracker maintain connection with the closest tracker on its right side and the closest tracker on its left side (see Fig.4.2).

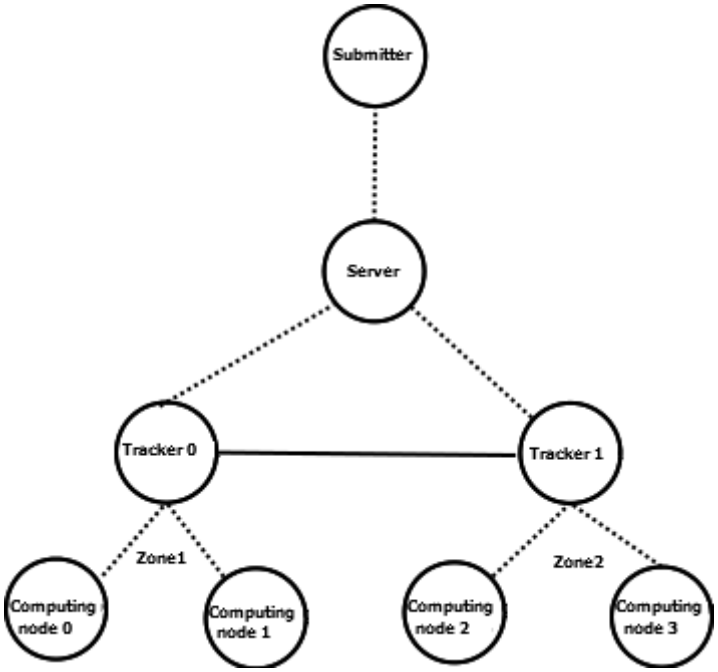

Figure 4.1: General topology architecture of GRIDHPC

- **Computing nodes or workers** are donors of computational resources. Computing nodes are grouped in subsets and managed by the tracker of zone. In a zone, workers publish their information regarding IP addresses and number of CPU cores to tracker of zone and wait for work. We note that computing node topology is a line (see Fig. 4.3). Each computing node has a specific ID like i to identify it when performing data exchange. Computing node i exchanges messages only with computing node i-1 and i+1. Each computing node has Htable (see Table 4.1) to know the IP addresses used by other computing nodes in order to maintain connection and communications between them via the underlying network.

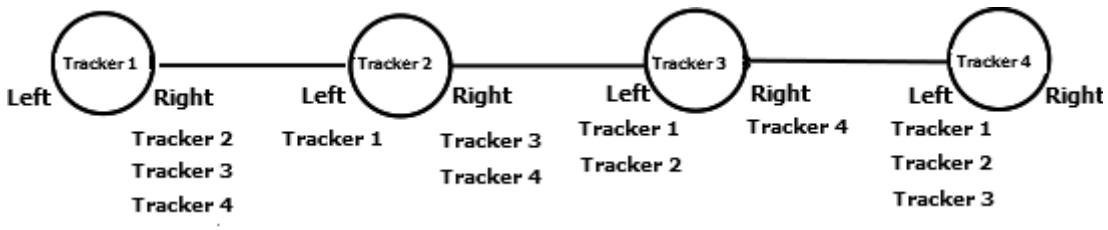

Figure 4.2: Trackers topology

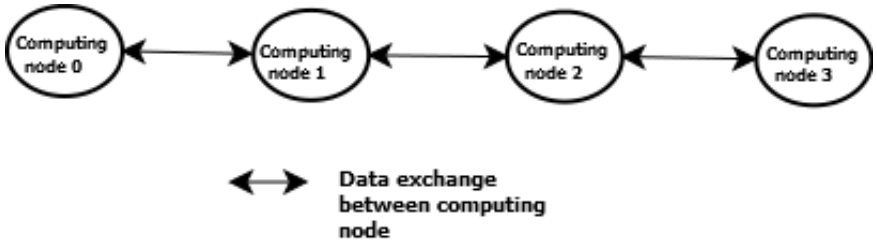

Figure 4.3: Computing nodes topology

4.3.2 Htable Initialization

Initially, we suppose that the system has a server and some trackers. When a new computing node joins underlying network, it sends to tracker of zone, i.e., closest tracker in tracker list stored in local memory information regarding resources such as IP addresses and number of CPU cores. The tracker of zone transfers this information to the server. The server set and stored it in the Htable. This step will be repeated when a new computing node wants to join the underlying network. Table 4.1 shows an example of global representation of Htable.

4.3.3 Communication of Htable to all computing nodes

When the submitter, wants to submit a task, it has to get firstly the Htable from the server and send it back to all computing nodes. This step is very important since submitter deduce from Htable the total number of CPU cores in order to decompose the initial task into sub-tasks and distribute them amongst a farm of computing nodes; Then each computing node divide sub-task into sub-sub-tasks and distribute them fairly to the different computing cores. Workers or computing nodes needs Htable to knows the IP addresses used by other computing nodes in order to exchange data between them via the best underlying network, i.e., high speed and low latency network like Infiniband and Myrinet.

	Computing node 0	Computing node 1	Computing node 2	Computing node 3
Nic Ethernet	172.16.17.25	172.16.17.26	172.16.34.30	172.16.34.31
Nic Infiniband or Nic Myrinet	172.18.17.25	172.18.17.26	172.18.34.30	172.18.34.31
Number of CPU cores	8	8	8	8

Table 4.1: Example of global representation of Htable

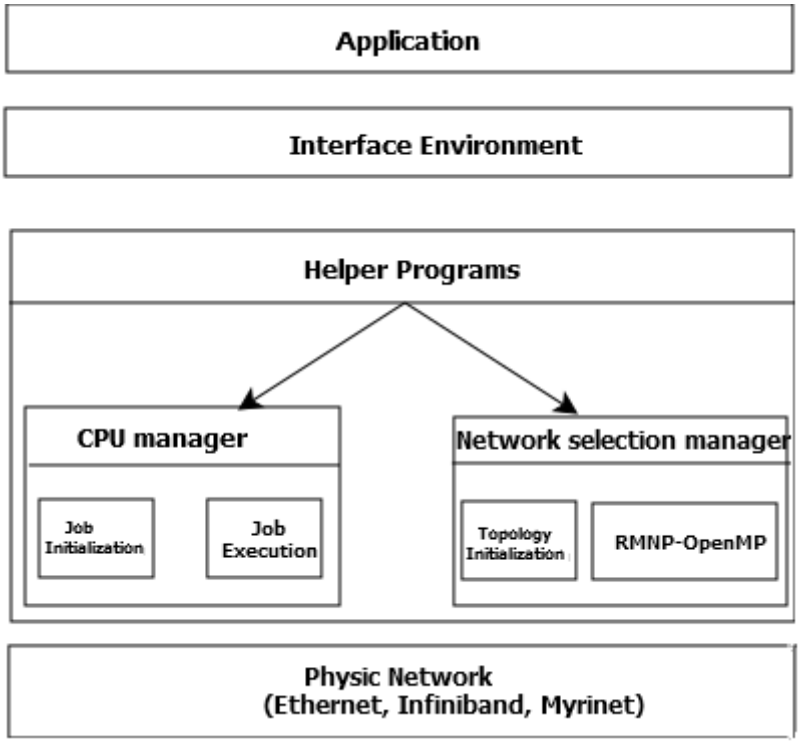

Figure 4.4: Environment Architecture of GRIDHPC

4.4 Environment Architecture of GRIDHPC

The decentralized environment GRIDHPC natively supports any combination of networks and multi-core CPUs by using the reconfigurable multi-network protocol RMNP and OpenMP. Figure 4.4 shows the architecture of GRIDHPC. It consists of five main components.

4.4.1 Interface Environment Component

Interface environment component is the interaction interface between the application like boundary value problem, planning problem and the environment. It allows users to submit their tasks and retrieve final results.

4.4.2 Helper Programs

GRIDHPC works with tools called helper programs that are responsible for the analysis of the application and building the network topology. The helper programs rely on two pillars namely the CPU manager and the Network selection manager (see Fig. 4.4). The CPU manager is composed of Job Initialization and Job Execution. The Network selection

manager is composed of Topology Initialization and RMNP-OpenMP modules.
In the CPU manager module, there are two components :

- Job Initialization Component is responsible for problem decomposition and assignment of tasks to individual CPU cores.
- Job Execution Component executes sub-tasks on the different CPU cores, takes care of data exchange, i.e., communication of iterates (updates) of the parallel iterative method and at the end of the application, it regroups the results from all the computing cores.

In the Network selection manager module, there are two components :

- Topology Initialization Component organizes connected computing nodes into clusters and maintains links between clusters. In particular, it is based on storing in the Htable information regarding the network interface card (NIC) used in the application by the different computing nodes (see Table 4.1).
- RMNP-OpenMP Component provides support for directed data exchange between computing nodes on several high speed networks like Infiniband, Myrinet and fast Ethernet using the reconfigurable multi-network protocol RMNP and between cores in a computing node via OpenMP. In particular, between computing nodes, data exchanges are made via GRID_Send, GRID_Receive and GRID_wait of the RMNP protocol; while in the same computing node, data exchange between cores are made via the directives of OpenMP.

We note that the CPU manager is in charge of data exchange between computing cores, i.e., reading/writing; while the network selection manager is in charge of data exchange between computing nodes via the best underlying network, i.e., high speed and low latency network like Infiniband and Myrinet. The combination of the CPU manager and the Network selection manager permits us to use the decentralized environment GRIDHPC in a multi-network and multi-core context.

4.5 Task assignation

4.5.1 Proximity metric

proximity metric [29] makes use of the longest common IP prefix length as the measure of proximity between computing nodes. For example, in the case of three computing nodes: A1 having IP address 192.16.64.10, A2 having IP address 192.16.64.11 and B1 having IP

address 192.16.34.20, the longest common prefix between A1 and A2 is 24 bits; while the longest common prefix between A1 and B1 is 16 bits. So A1 considers that A2 is closer than B1.

4.5.2 Processor hierarchy and GRIDHPC

Task assignation in GRIDHPC is based on the hierarchical Master-Worker paradigm. The Hierarchical Master-Worker paradigm relies on three entities: a master, several sub-masters (coordinators) and several workers.
The master or submitter is the unique entry point, it gets the entire application as a single original task, i.e., root task. The master decomposes the root task into sub-tasks and distributes these sub-tasks amongst a farm of workers. The master takes also care of gathering the scattered results in order to produce the final result of the computation.
The sub-masters or coordinators are intermediary entities that enhance scalability. They forward sub-tasks from the submitter to workers and return results to the submitter limiting network congestion.
The workers run in a very simple way: they receive a message from the sub-master that contains their assigned sub-sub-tasks and they distribute the sub-sub-tasks to their computing cores. They perform computations, exchange data with neighboring computing nodes and at the end of the application, when the iterative schemes have converged, they regroup the results from all their computing cores and send them back to the sub-master, then the coordinator transfers the results to the submitter. We note that such a task assignation technique has many advantages as compared with the case where there are no coordinators. Firstly, sending results to submitter via coordinators avoids bottleneck at submitter because if all the computing nodes want to send results to submitter, then there could be a bottleneck at submitter. Secondly, submitter does not have to connect to all the computing nodes in order to reserve them and send sub-tasks; submitter has only to connect to coordinators, computing node reservation and sub-task sending is carried out in parallel by coordinators. Figure 4.5 shows an example of processor hierarchy.

4.5.3 Example of scenario

When submitter has collected enough computing nodes, it divides them into groups based on proximity metric (see section 4.5.1 and Figure 4.5); in each group, a computing node is chosen by submitter to become sub-master, i.e., coordinator which will manage computing nodes in the group. The number of computing nodes in a group cannot exceed Cmax = 32 in order to ensure efficient management of coordinator. Submitter sends sub-tasks

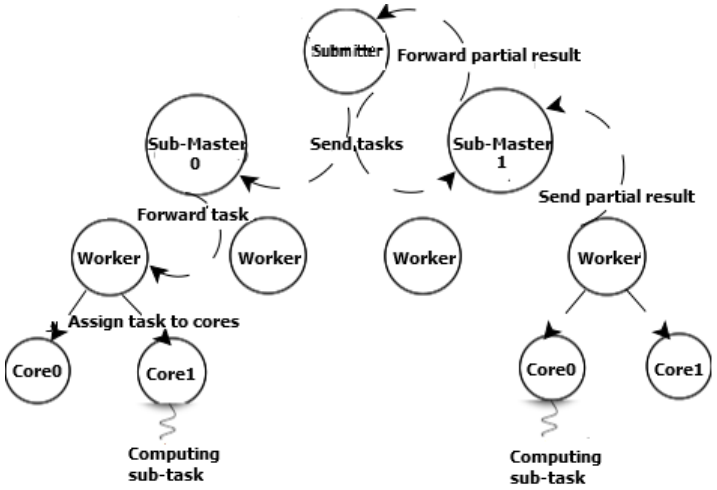

Figure 4.5: Processor Hierarchy

to groups coordinators. Sub-tasks are forwarded by coordinators to computing nodes; Computing nodes divide them into sub-sub-tasks and distribute them to the different computing cores. At the end, computing nodes send their sub-tasks result to coordinators, then the coordinators transfer the results to the submitter.

4.6 Parallel Programming Model

4.6.1 Communication operations

The idea is to facilitate the use of GRID platforms as well as programming of large scale HPC applications and hide complexity of communication management as much as possible. RMNP has a reduced set of communication operations, there are only a send, receive and wait operations : GRID_Send, GRID_Receive and GRID_wait respectively. Contrarily to MPI communication library where communication mode is fixed by the semantics of communication operations, the communication mode of a given communication operation which is called repetitively depends on the context at application level like chosen distributed iterative schemes of computation, e.g., synchronous or asynchronous iterative schemes, elements of context like topology at network level, i.e., inter or intra cluster communication and type of network like Ethernet, Infiniband and Myrinet. The programming model permits us to expect scalable performance and application flexibility. The prototype of the communication operations of our programming model are summarized in listing 4.1 where :

Listing 4.1: Prototype of RMNP communication operations

```
1) int GRID_Send(GRIDSubtask *pSubtask, uint32_t dest, char *buffer,
   size_t size, int flags);

2) int GRID_Receive(GRIDSubtask *pSubtask, uint32_t source, char
   *buffer, size_t size, int flags);

3) int GRID_Wait(GRIDSubtask* pSubtask, uint32_t *iSubtaskRank, int
   *flags);
```

- **GRID_Send** communication operation is used to send a message placed in buffer to subtask dest, i.e., rank of destination subtask.
- **GRID_Receive** communication operation is used to receive a message from subtask source, i.e., rank of source subtask.
- **GRID_wait** operation is used to wait for a message from another computing node.

Note that flags parameter in these operations are used to distinguish two types of messages: CTRL_FLAG indicates control messages and DATA_FLAG indicates data messages. Data messages are used to exchange updates between computing nodes; while control messages are used to exchange information related to computation state like state of local termination condition, termination command, etc. These data are particularly important for the convergence detection process and termination phase. Note also that the communication mode for data message is chosen according to the context by RMNP; while communication mode for control message is always asynchronous and reliable using control channel of RMNP.

4.6.2 GRIDHPC and OpenMP

4.6.2.1 Combination of shared and distributed memory

Combination of shared and distributed memory in our programming model (see Fig 4.6) permits us to use all computing resources and to perform data exchange between computing cores and computing nodes. Creating a parallel program based on RMNP + OpenMP, i.e., distributed memory and shared memory typically requires a major reorganization of the original sequential code. We combine GRIDHPC and OpenMP together in a program

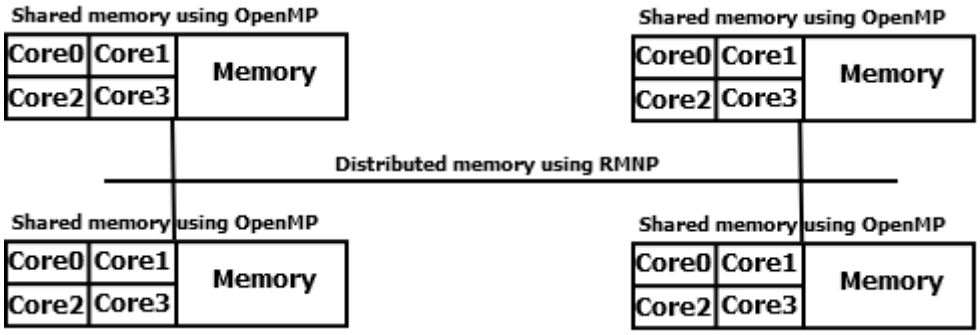

Figure 4.6: Combination of Shared and Distributed Memory

in order to execute such application in a multi-network and multi-core context. In particular, in the same computing node, data exchanges between computing cores are made via OpenMP, while data exchanges between computing nodes are made via GRID_Send, GRID_Receive and GRID_wait.
With OpenMP, multiple threads collaborate to execute a program, they share the resources, including the address space of the corresponding process. In order to get performance with OpenMP, we previlegiate to have only one thread per single core.

4.6.3 Application programming model

Figure 4.7 displays the activity diagram that a parallel application must follow. The diagram consists of thirteen activities.

- **Task definition module:** First, the application is defined at the submitter, i.e., setting task parameters as well as computational schemes (synchronous iterations, asynchronous iterations, hybrid), problem size and the number of computing nodes required. Note that hybrid iterative scheme is a combination of synchronous and asynchronous computation schemes, e.g, synchronous iterations in the same cluster and asynchronous iterations between clusters, i.e., at global level.

- **Collect computing nodes module:** based on the task definition, the submitter collects free computing nodes

- **Enough computing nodes module :** the submitter verifies if there are enough free computing nodes to carry out the task. If there are not enough free computing nodes, then the computation is terminated.

- **Send sub-tasks module:** if there are enough free computing nodes, then the submitter sends sub-tasks to coordinators.

Sub-Sub-Tasks Parameters	Description
$pSubtask \rightarrow iRank$	Rank of a given computing node
$pSubtask \rightarrow iRankC$	Rank of a core in a given computing node
$pSubtask \rightarrow core$	number of cores in a given computing node
$pSubtask \rightarrow cSubSubtasks$	number of sub-sub-tasks
$pSubtask \rightarrow cCores$	number of cores of all the computing nodes
$pSubtask \rightarrow params$	parameters of sub-sub-task
$pSubtask \rightarrow params\_size$	size of parameters of sub-sub-task
$pSubtask \rightarrow result$	result of sub-sub-task
$pSubtask \rightarrow result\_size$	size of result of sub-sub-task
$pSubtask \rightarrow pSubtasks$	pointer points to an array of sub-sub-tasks

Table 4.2: Description of sub-sub-task parameters

- **Forward sub-tasks module:** The coordinator forwards sub-tasks from submitter to workers.

- **Receive sub-tasks module:** computing nodes receive sub-tasks from coordinator and become workers.

- **Distribute sub-tasks on the different cores module:** decomposes sub-task into sub-sub-tasks and assigns each one to a core at a given computing node. Note that the number of sub-sub-tasks is equal to the maximum number of cores in a computing node.

- **Calculate module:** This is the module which performs computations relative to sub-tasks. Each core executes its sub-sub-task. We note that in the case of applications solved by iterative algorithms, a worker has to carry out many iterations; after each iteration, it has to exchange updates with other workers. For this purpose, it uses RMNP for data exchanges between computing nodes. Table 4.2 describes the parameters of sub-sub-tasks used in the calculate activity. One writes the code to perform the sub-sub-task assigned to a core. One can retrieve sub-task rank of a given computing node, i.e., iRank field, sub-sub-task rank of a core at a given computing node, i.e., iRankC field and sub-sub-task parameters, i.e., params field.

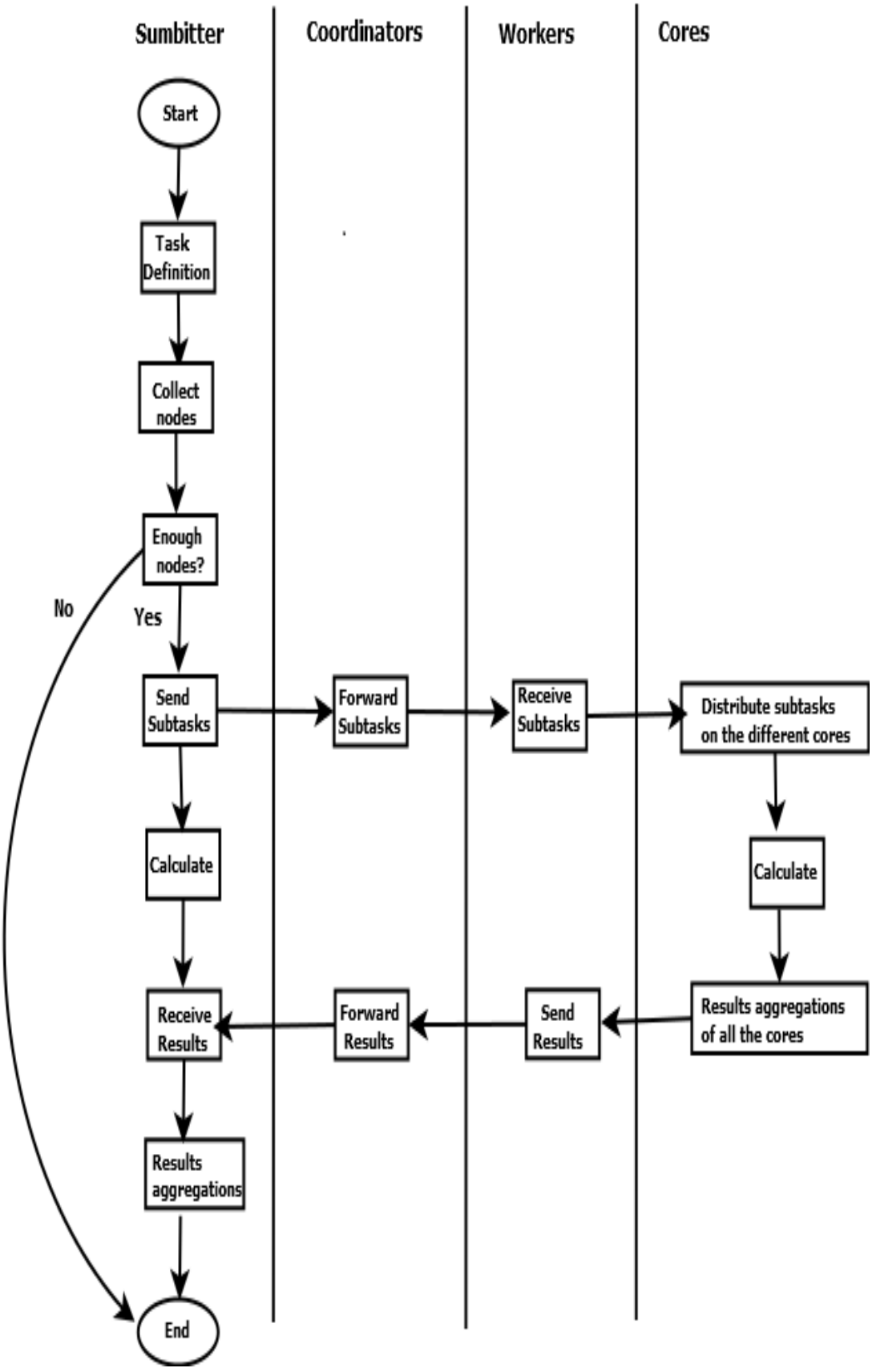

Figure 4.7: Activity diagram of a parallel application with GRIDHPC

One can use communication operations, i.e., GRID_Send and GRID_Receive for data exchanges between computing nodes (relies on iRank field) and one can use OpenMP to communicate (Read/Write) between cores at a given computing node (relies on iRankC field).

- **Results aggregation of all the cores:** sub-sub-tasks results of all the cores are aggregated into one result at a given computing node.
- **Send results module:** Sends aggregated results to coordinator.
- **Forward results module:** The coordinator forwards results from workers to submitter.
- **Receive results module:** the submitter receives sub-tasks results from coordinators.
- **Results aggregation module:** sub-tasks results of all the workers are aggregated at submitter into final result.

4.7 Develop HPC applications with GRIDHPC

In order to develop an HPC application with GRIDHPC, programmers have to write code for only three functions corresponding to the following three activities: Task Definition, Calculate and Results Aggregation. The other activities like Send results, Receive Results, etc are taken into account by the environment and are transparent to programmers. In the Task_Definition() function, programmers define the task in indicating the number of sub-tasks and sub-tasks data, number of computing nodes and computational scheme. Note that this function is called on submitter.
In the Calculate() function, programmers write sub-tasks code. In this function, the programmer can use GRID_Send() and GRID_Receive() to send or receive message between computing nodes, if necessary, and OpenMP to exchange data between computing cores at a given computing node. We note that this function is called on computing nodes, i.e., workers. Chapter five give an illustration of the decomposition and implementation of a loosely synchronous application (obstacle problem that present frequent data exchange between computing nodes) with the decentralized environment GRIDHPC; while chapter 6 show an illustration of the decomposition and implementation of a pleasingly parallel application (planning problem that do not need exchange between computing nodes, i.e., each computing node work independently) with the decentralized environment GRIDHPC.

In the Results_Aggregation() function, programmers define how sub-tasks results are aggregated into final result and write the final result to an output, i.e., a console. Note that this function is called on submitter.

4.8 Conclusion

In this chapter, we have described the global topology and the general architecture of the decentralized environment GRIDHPC with its main functionalities. Afterward, we have presented the hierarchical task allocation mechanism that accelerates task allocation to computing nodes and avoids connection bottleneck at submitter; we have presented a programming model for GRIDHPC that facilitates the work of programmer. In particular, the communication operations set is reduced with only three operations, basically send, receive and wait operations. GRIDHPC facilitates the use of multi-cluster and grid platform for loosely synchronous applications and also embarrassingly parallel application. GRIDHPC exploits all the computing resources (all the available cores of computing nodes) as well as several types of networks like Ethernet, Infiniband and Myrinet in the same application. GRIDHPC functionality relies on a reconfigurable multi-network protocol RMNP for controlling multiple network adapters and on OpenMP for the exploitation of all the available cores of computing nodes. In particular, we detailed the features induced by multi-core and heterogeneous-networks multi-cluster support. These features involved the developments of helper programs. These programs are responsible for the analysis of the application and building the network topology. It rely on two pillars of GRIDHPC environment namely the CPU manager and the Network selection manager that are in charge of data exchange between computing cores, i.e., reading/writing and between computing nodes via the best underlying network, i.e., high speed and low latency network like Infiniband and Myrinet.
In the next chapter, we shall present the application to obstacle problem and the computational experiments that have been carried out on Grid5000 platform.

CHAPTER 5

Application to obstacle problem

Contents

5.1 Introduction

This chapter presents a first type of HPC application related to the solution of numerical simulation problem: the obstacle problem. This type of problems belongs to the class of loosely synchronous applications. An evaluation of the overall efficiency and scalability of GRIDHPC in a multi-core and multi-network context for the obstacle problem is also presented. The remainder of this chapter is organized as follows: section 5.1 presents an introduction to the obstacle problem. Section 5.2 presents the decomposition of the obstacle problem with GRIDHPC. Computational results for large scale numerical simulation problems using GRIDHPC are displayed and analyzed in section 5.3. Finally, section 5.4 concludes this chapter.

5.1.1 Obstacle problem

The application we consider, i.e. the obstacle problem, belongs to a large class of numerical simulation problems (see [18] and [19]). The problem is to find the equilibrium position of an elastic membrane whose boundary is held fixed, and which is constrained to lie above a given obstacle. The obstacle problem occurs in many domains like mechanics and financial mathematics, e.g. options pricing. The obstacle problem is also a sub-problem of more complex problems, e.g. in finance.

5.1.1.1 Problem formulation

There are many equivalent formulations of the obstacle problem like variational inequality and constrained optimization problem. Reference is made to [18], [19] and [20] for more details. We concentrate here on the following variational inequality formulation:

$$\begin{cases} Find\ u^* \in K\ such\ that \\ \forall v \in K, (A.u^*, v - u^*) \geq (f, v - u^*), \end{cases} \tag{5.1}$$

where K is a closed convex set defined by $K = \{v | v >= \varnothing$ everywhere in $\Omega\}$, and (.,.) denotes the dot product (u,v) $= \int uvdx$.

5.1.1.2 Fixed point problem and projected Richardson method

The discretization of the obstacle problem leads to the following large scale fixed point problem

$$\begin{cases} Find\ u^* \in V\ such\ that \\ u^* \ = F(u^*), \end{cases} \tag{5.2}$$

where V is an Hilbert space and the mapping $F : v \to F(v)$ is a fixed point mapping from V into V. Let α be a positive integer, for all v ∈ V , we consider the following block-decomposition of v and the associated block-decomposition of the mapping F for distributed implementation purpose:

$$v = (v_1, ..., v_\alpha)$$

$$F(v) = (F_1(v), ..., F_\alpha(v)).$$

We have $V = \prod_{i=1}^{\alpha} V_i$, where V_i are Hilbert spaces and $\prod_{i=1}^{\alpha}$ denotes the Cartesian product; we denote by $(.,.)_i$ the scalar product on V_i and $|.|_i$ the associated norm, $i \in \{1, ..., \alpha\}$; for all $u, v \in V$, we denote by $(u, v) = \sum_{i=1}^{\alpha}(u_i, v_i)_i$, the scalar product on V and $|.|$ the associated norm on V. In the sequel, we shall denote by A a linear continuous operator from V onto V, such that $A.v = (A_1.v, ..., A_\alpha.v)$ and which satisfies:

$\forall i \in \{1, ..., \alpha\}, \forall v \in V, (A_i.v, v_i) \geq \sum_{j=1}^{\alpha} n_{i,j}|v_i|_i|v_j|_j,$

where the matrix $N = (n_{i,j}), 1 \leq i, j \leq \alpha$ is an M-matrix of size $\alpha * \alpha$.

We recall that the diagonal entries of an M-Matrix are strictly positive and its off-diagonal entries are negative or null; moreover the inverse of an M-Matrix exists and is nonnegative.

We denote by K_i, a closed convex set such that $K_i \subset V_i, \forall i \in \{1, ..., \alpha\}$, we denote by K, the closed convex set such that $K = \prod_{i=1}^{a} K_i$ and b, a vector of V that can be written as: $b = (b_1, ..., b_\alpha)$. For all $v \in V$, let $P_K(v)$ be the projection of v on K such that $P_K(v) = (P_{k_1}(v_1), ..., P_{K_\alpha}(v_\alpha))$, where P_{K_i} denotes the mapping that projects elements of V_i onto K_i; $\forall i \in \{1, ..., \alpha\}$. For any $\delta \in R, \delta > 0$, we define the fixed point mapping F_δ as follows (see [18]).

$$\forall v \in V, F_\delta(v) = P_K(v - \delta(A.v - b)).$$

The mapping F_d can also be written as follows.

$$F_\delta(v) = (F_{1,\delta}(v), ..., F_{\alpha,\delta}(v)) \text{ with}$$

$$F_{i,\delta}(v) = P_{k_i}(v_i - \delta(A_i.v - b_i)), \forall v \in V, \forall i \in \{1, ..., \alpha\}.$$

5.1.1.3 Parallel projected Richardson method

We consider the distributed solution of the fixed point problem (5.2) via the projected Richardson method combined with several schemes of computation like synchronous iterative scheme: $u^{p+1} = F_\delta(u^p)$, $\forall p \in N$ the set of natural number or asynchronous iterative schemes of computation that can be defined as follows (see [18]).

$$\begin{cases} u_i^{p+1} = F_{i,\delta}(u_1^{\rho_1(p)}.....u_j^{\rho_j(p)}.....u_\alpha^{\rho_\alpha(p)}) \ if \ i \in s(p), \\ u_i^{p+1} = u_i^p \ if \ i \notin s(p), \end{cases} \tag{5.3}$$

where

$$\begin{cases} s(p) \subset \{1, ..., \alpha\}, s(p) \neq \phi, \forall p \in N, \\ \{p \in N | i \in s(p)\}, \ is \ infinite, \forall i \in \{1, ..., \alpha\}, \end{cases} \tag{5.4}$$

and

$$\begin{cases} j(p) \in N, 0 \leq \rho_j(p) \leq p, \forall j \in \{1, ..., \alpha\}, \forall p \in N, \\ \lim\limits_{p \to \infty} \rho_j(p) = +\infty, \forall j \in \{1, ..., \alpha\}. \end{cases} \tag{5.5}$$

Where $\rho_j(p), j = 1, ..., \alpha$ represent delayed iteration numbers. We note that the use of delayed iteration numbers $\rho_j(p)$ in equations (5.3) and (5.5) permit one to model a non deterministic behavior.

In equations (5.3) and (5.4), $s(p)$ denotes the set of components updated at iteration p. The second line of equation (5.4) implies that non component of the iterate vector is abandoned forever during the computation. The first line of equation (5.5) is a simple causality relation on delays. The second line of equation (5.5) implies that more and more recent components have to be used as the computation progresses.

The above asynchronous iterative scheme of computation models parallel approximation methods whereby computations are carried out without order nor synchronization. The convergence of asynchronous projected Richardson method has been established for many problems like numerical simulation and optimization problems in [18] (see also [20], [21] and [22]).

The choice of scheme of computation, i.e. synchronous, asynchronous or Hybrid schemes have important consequences on the efficiency of the distributed solution. The interest of asynchronous iterations for various problems including boundary value problems and optimization has been shown in [23], [24], [25], [26], [27]. In particular, we note that asynchronous iterative method are very efficient for unbalanced problems in optimization [25].

Asynchronous iterations are also fault-tolerant since some messages may be lost and finally replaced by new messages corresponding to new updates. Asynchronous iterations are well suited to massive parallelism.

5.2 Decomposition of the obstacle problem and Implementation with GRIDHPC

5.2.1 Approach to the distributed solution of the obstacle problem

The decentralized environment GRIDHPC aims at using several types of network, i.e. Ethernet, Infiniband, Myrinet and thousands of cores distributed over several clusters at the same time. In this subsection we present the implementation of GRIDHPC in a multi-core and multi-network context for the obstacle problem. Before we present the implementation of GRIDHPC, we detail the decomposition method of the obstacle problem.

5.2.1.1 Domain decomposition

We illustrate the decomposition method of the obstacle problem via the simple example displayed in Figure 5.1 where a cubic domain is decomposed into four sub-domains, each sub-domain being decomposed into four sub-sub-domains. This case corresponds to a decomposition and assignation of tasks to four computing nodes with four computing cores. The iterate vector of the discretized obstacle problem is decomposed into a * b sub-blocks of size n/a * n/b * n points where a denotes the number of cores per computing node, b denotes the number of computing nodes and n denotes the number of discretization points along one direction. In our example, a = 4, b = 4 and n = 256. The sub-domains assigned to the different computing nodes and cores are presented in Table 5.1 where X = n = 256 denotes the length (x-axis), Y = n/b = 64 denotes the width (y-axis) and Z = n/a = 64 denotes the height of the blocks (z-axis). The decomposition technique balance fairly the computing tasks,i.e., the number of points on the different cores. Note that the results of all the cores at a given computing node are aggregated into one result after each iteration and are exchanged with the next and previous computing nodes using the RMNP communication protocol.

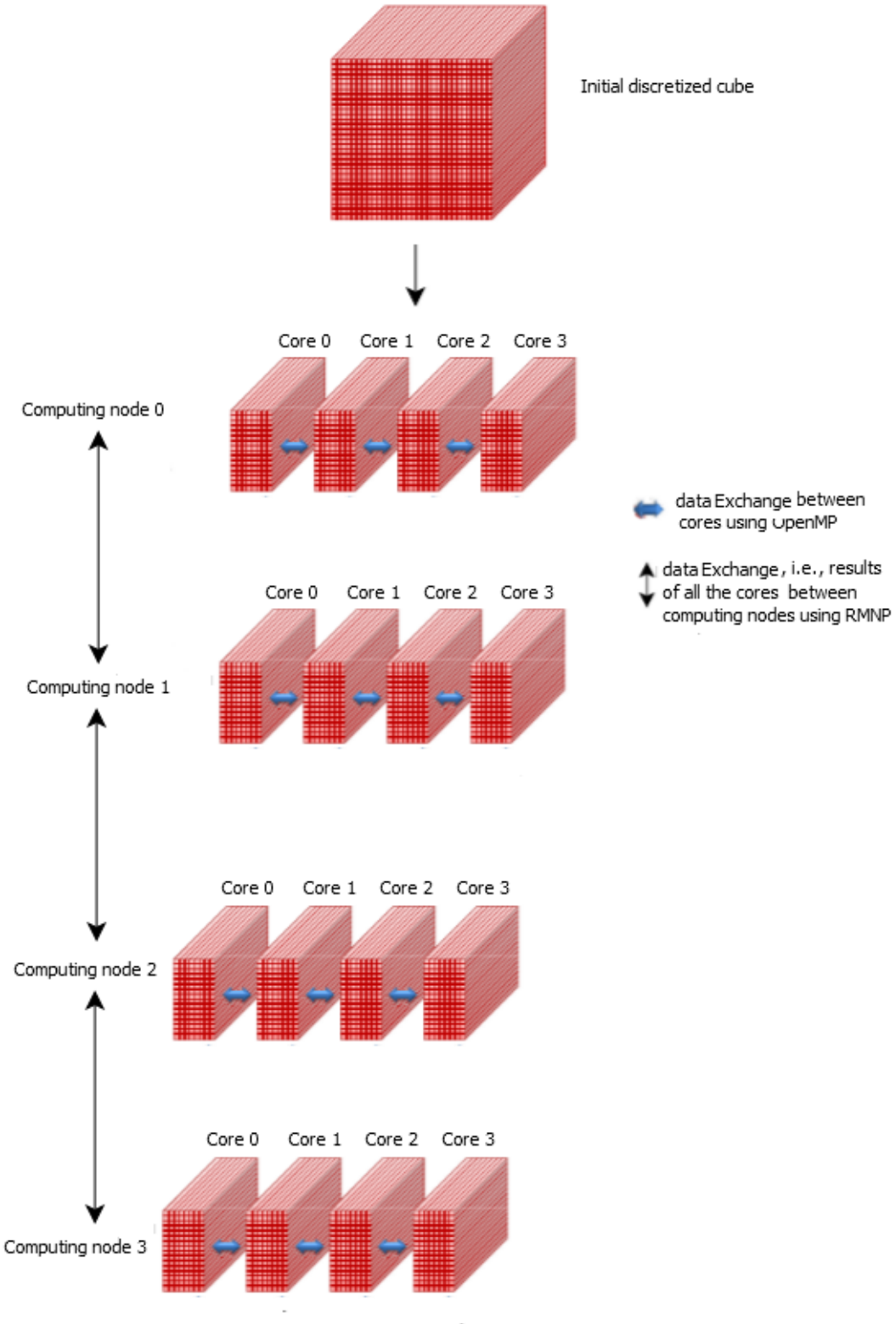

Figure 5.1: Example of Decomposition of the discretized obstacle problem into subtasks

	Computing node 0			Computing node 1			Computing node 2			Computing node 3		
	Z	Y	X	Z	Y	X	Z	Y	X	Z	Y	X
Core 0	[1-64]	[1-64]	[1-256]	[1-64]	[65-128]	[1-256]	[1-64]	[129-192]	[1-256]	[1-64]	[193-256]	[1-256]
Core 1	[65-128]	[1-64]	[1-256]	[65-128]	[65-128]	[1-256]	[65-128]	[129-192]	[1-256]	[65-128]	[193-256]	[1-256]
Core 2	[129-192]	[1-64]	[1-256]	[129-192]	[65-128]	[1-256]	[129-192]	[129-192]	[1-256]	[129-192]	[193-256]	[1-256]
Core 3	[193-256]	[1-64]	[1-256]	[193-256]	[65-128]	[1-256]	[193-256]	[129-192]	[1-256]	[193-256]	[193-256]	[1-256]

Table 5.1: Sub-domains assigned to computing nodes

Algorithm 3 displays the basic computational procedure at computing node P_r which is at row r and which is not on the boundary of the grid.

The node P_r updates the sub-blocks of components of the iterate vector denoted by U_i at each iteration.

Algorithm 3: Basic computational procedure at computing node P_r

repeat
- send U_i to node P_{r+1}
- receive U_i from node P_{r+1}
- Each core at node P_r uses the corresponding iterate vector from U_i
- send U_i to node P_{r-1}
- receive U_i from node P_{r-1}
- Each core at node P_r uses the corresponding iterate vector from U_i
- Generate new U_i after aggregate the results of the sub-sub-blocks components of all the cores at the computing node P_r

until convergence

5.2.1.2 Implementation

The implementation of GRIDHPC relies on two pillars of the helper programs namely *CPU manager* and *Networks selection manager*.

To illustrate this implementation, we use the example displayed in Figure 5.1 which is related to the solution of the discretized obstacle problem.

We suppose that we have four computing nodes (computing node 0, 1, 2 and 3) and each computing node is composed of four CPU cores. The environment GRIDHPC uses the **Job Initialization component** to decompose fairly the initial domain into four sub-blocks since we have four computing nodes, then it decomposes fairly each sub-block into four sub-sub-blocks since we have four cores at each computing node (in total sixteen sub-sub-blocks are assigned to sixteen CPU cores).

The environment GRIDHPC uses the **Job Execution component** to run the sub-sub-tasks associated with sub-sub-blocks that are assigned to each core. In the same computing node, data exchange between computing cores are made via OpenMP. The RMNP-OpenMP component aggregates the updates computed by the different computing cores and exchanges data with other computing nodes until the convergence is obtained.

5.2.2 Convergence detection

In the case of synchronous iterative schemes, the convergence test is based on the difference between successive values of the components of the iterate vector. The global convergence is detected when $\sigma = max_{i \in N}$ ($|{u_i}^{k+1}$ - ${u_i}^k|$) $< \epsilon$ where ${u_i}^k$ is the value of the i_th component of the iterate vector at iteration k, N is the set of points and ϵ is a positive constant. In our example, $\epsilon = 10^{-11}$. The termination is detected as follows: two global tokens are appended to update exchanges between computing nodes : token $tok\_conv_{k,k+1}$ is appended to updates sent from computing node P_k to P_{k+1} in order to collect information about local convergence test; token $tok\_term_{k,k-1}$ is appended to updates sent from P_k to P_{k-1} in order to propagate the termination state (see Figure 5.2). Note that the message types which contain these tokens are control messages, i.e., flags = CTRL_FLAG. Note also that $tok\_conv_{k,k+1}$ is the logical conjunction of all the local tokens (${tok\_conv_{k,k+1}}^q$) of cores at computing node P_k and $tok\_conv_{k-1,k}$. The token ${tok\_conv_{k,k+1}}^q$ is true if $\sigma_i = max_{i \in N_q}$ ($|{u_i}^{k+1}$ - ${u_i}^k|$) $< \epsilon$, where N_q is the subset of points assigned to core q, $q \in 1, ..., a$ and a is the number of cores.

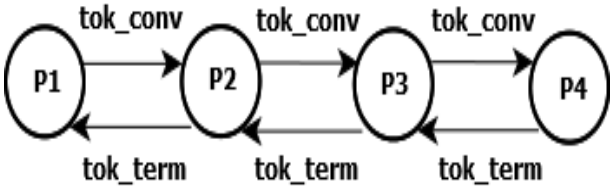

Figure 5.2: Termination detection of synchronous iterations

In the case of asynchronous iterative schemes, we have implemented a different termination method for the obstacle problem. This termination method is an implementation of the termination method of El Baz [67]. This method is based on activity graph.

The behavior of computing nodes is given by the finite state machine in Figure 5.3. It can be summarized as follows; each computing node can have three possible states: Active (A), Inactive (I) and Terminated (T). Four types of messages can be sent: activate message, inactivate message, termination message and update of the sub-sub-blocks message. Note that the first three message types are control message, i.e., flags = CTRL_FLAG and the last message type is data message, i.e., flags = DATA_FLAG.

Initially, only the computing node P_1 is active. This computing node is called the root. All other computing nodes are inactive.

Each computing node P_r has to store following additional data:

- The identity of the computing node that has activated P_r (which is also called parent

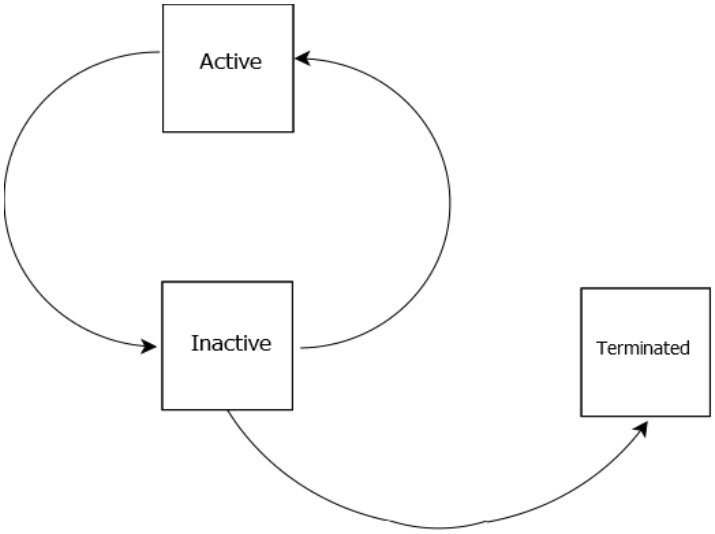

Figure 5.3: States of computing nodes in the convergence detection procedure of asynchronous iterations

of P_r)

- The list of computing nodes activated by P_r(which are also called children of P_r)

In active state (A), a computing node P_r evaluates the local termination test, i.e., local conjunction of all the tokens of computing cores at computing node P_r: if it is satisfied, then P_r does not compute any update; otherwise, each computing core at P_r updates components of sub-sub-blocks assigned to it, after that P_r aggregates the results of the computing cores and sends updates to adjacent computing nodes. Note that if $P_{r'}$ is inactive and receives an activate message from a computing node P_r, then $P_{r'}$ becomes the children of P_r; if P_r receives an inactivate message from $P_{r'}$, then P_r removes $P_{r'}$ from its list of children.

In inactive state (I), a computing node is waiting for messages using GRID_wait operation.

Terminated state (T) corresponds to the case where the computation is terminated at the computing node.

To illustrate the procedure, we consider a simple example of the evolution of the activity graph in the case of the four computing nodes presented in Figure 5.4. Initially, only the root, i.e., computing node P_1 is active and all other computing nodes are inactive. The computing nodes become progressively active on receiving an activate message from other computing nodes like P_1. An activity graph is generated; the topology of the graph changes progressively as the messages are received and the local termination tests are satisfied; so an active computing node becomes inactive if its list of children is empty and its local termination test is satisfied; then the computing nodes sends an inactive

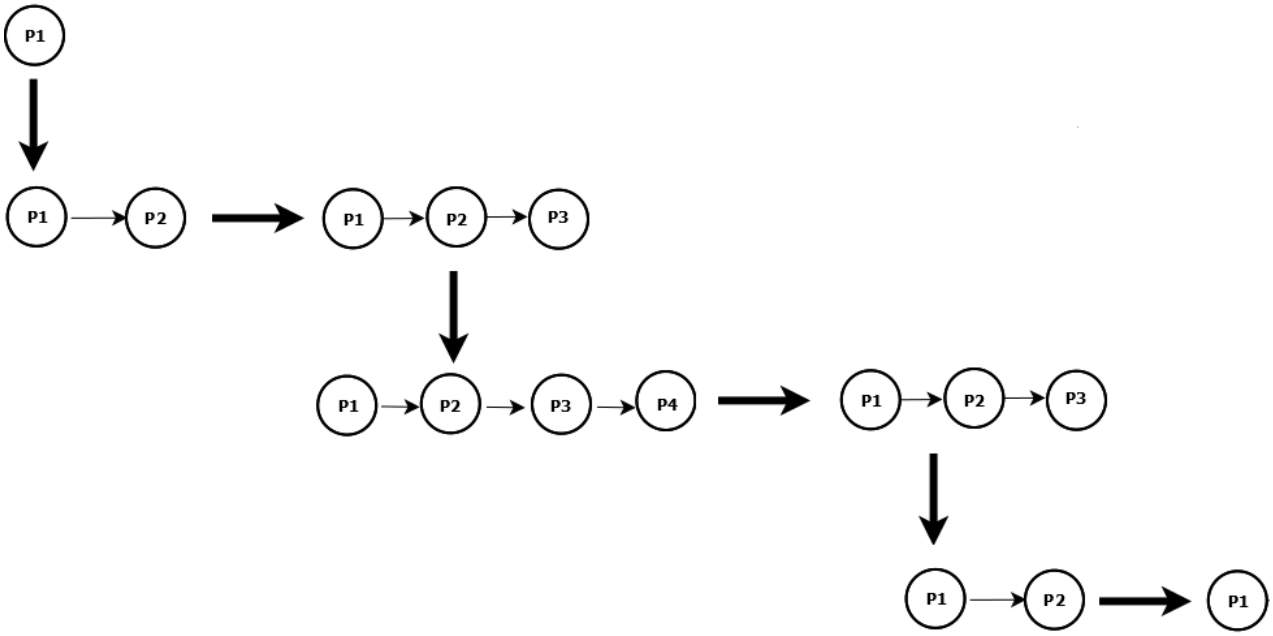

Figure 5.4: Evolution of the activity graph

message to its parent. At the end, all the computing nodes becomes inactive and global convergence is detected. Note that the root P_1 is the last node to become inactive.

5.3 Evaluation and computing results

The formulation of the obstacle problem was presented in equation (6.1). We consider the discretization of the obstacle problem. The distributed solution of the associated fixed point problem (5.2) via the projected Richardson method combined with several iterative schemes of computation is considered.
The experiments are carried out via GRIDHPC to solve the 3D obstacle problem with different schemes of computation, i.e. synchronous, asynchronous and hybrid schemes of computation. We consider cubic domains with n = 256 and 512 points where n denotes the number of points considered on each edge of the cube. In the distributed context, i.e., for several machines, we have considered the case where machines either belong to a single cluster or several clusters connected via Internet.

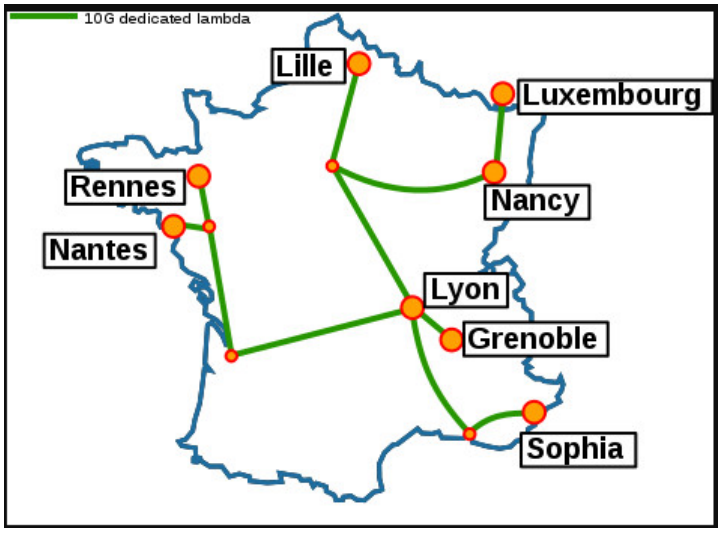

Figure 5.5: Grid5000 topology

5.3.1 Grid5000 platform

Computational experiments have been carried out on the Grid5000 platform [3]. The French grid platform is a large-scale and versatile academic testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including cloud, HPC and big data. It provides access to a large amount of resources: more than 1000 computing nodes which have different kinds of CPUs (AMD Opteron, Intel Xeon, etc) and operating systems, more than 8000 cores, grouped in homogeneous clusters, and featuring various technologies: 10G Ethernet, Infiniband, Myrinet, GPUs, Xeon PHI. Sites of Grid5000 have several clusters with different performances and are distributed over nine cities in France. Note that all clusters connected to RENATER, i.e., French national education and research network with a 10Gb/s link.
Figure 5.5 shows the global topology of the platform.

5.3.2 Experimental results

This subsection presents an evaluation of the overall efficiency and scalability of GRIDHPC in a multi-core and multi-network context for the obstacle problem.
We display the computing time and Computing gain of the parallel synchronous, asynchronous and hybrid iterative algorithms. The Computing gain is given as follows:

$$Computing\ gain\ C_g = t_1/ts \tag{5.6}$$

where t_1 is the parallel time on one multi-core computing node and ts is the parallel

Site	Cluster	Processor Type	Cores	Interconnection Networks	clock Ghz	RAM GB
Lille	Chinqchint	Intel Xeon E5440 QC	8	Ethernet and Myrinet	2.83	8
Nancy	Graphene	Intel Xeon X3440	4	Ethernet and Infiniband	2.53	16
Rennes	Paravance	Intel Xeon E5-2630v3	16	Ethernet	2.4	128
Grenoble	Edel	Intel Xeon E5520	8	Ethernet and Infiniband	2.27	24
Grenoble	Genepi	Intel Xeon E5420 QC	8	Ethernet and Infiniband	2.5	8

Table 5.2: Characteristics of machines

time on several multi-core computing nodes.

The synchronous, asynchronous and hybrid iterative methods are denoted by : Syn, Asyn and Hybrid, respectively; Ethernet, Infiniband and Myrinet networks are denoted by : ETH, IB and MYRI, respectively. Table 5.2 displays the characteristics of the machines used in the computational experiments.

Computing results in Figure 5.6 are obtained with Ethernet or Infiniband network on Graphene cluster in Nancy site of the Grid5000 testbed. We can see that the Computing gain increases more rapidly with Infiniband network than with Ethernet network for both synchronous and asynchronous iterative schemes of computation. This is due to the high bandwidth and low latency of Infiniband network. The experiments are carried out with up to 8 computing nodes, i.e., 32 cores. We note that asynchronous iterative schemes of computation perform better than synchronous iterative schemes since there are no idle time due to synchronization or synchronization overhead. In Figure 5.6, we display the average number of iterations of asynchronous iterative algorithms. We note that the number of iterations performed by synchronous schemes remains almost constant; while the average number of iterations performed by asynchronous schemes increases with the number of computing nodes since some computing nodes may iterate faster than others like the first and last computing nodes that have only one neighbor.

Computing results in Figure 5.7 are obtained with Ethernet or Myrinet network using Chinqchint cluster located in Lille site of the Grid5000 testbed. The experiments show that the Computing gain with Myrinet network is better than with Ethernet network for both synchronous and asynchronous iterative schemes of computation for the same reason as above, i.e., high bandwidth and low latency network with Myrinet. The experiments are carried out with up to 8 computing nodes, i.e., 64 cores. We note that asynchronous iterations perform better than synchronous iterations. We note also that the number of iterations performed by synchronous schemes remains almost constant and the average number of iterations in the case of asynchronous schemes of computation increases with

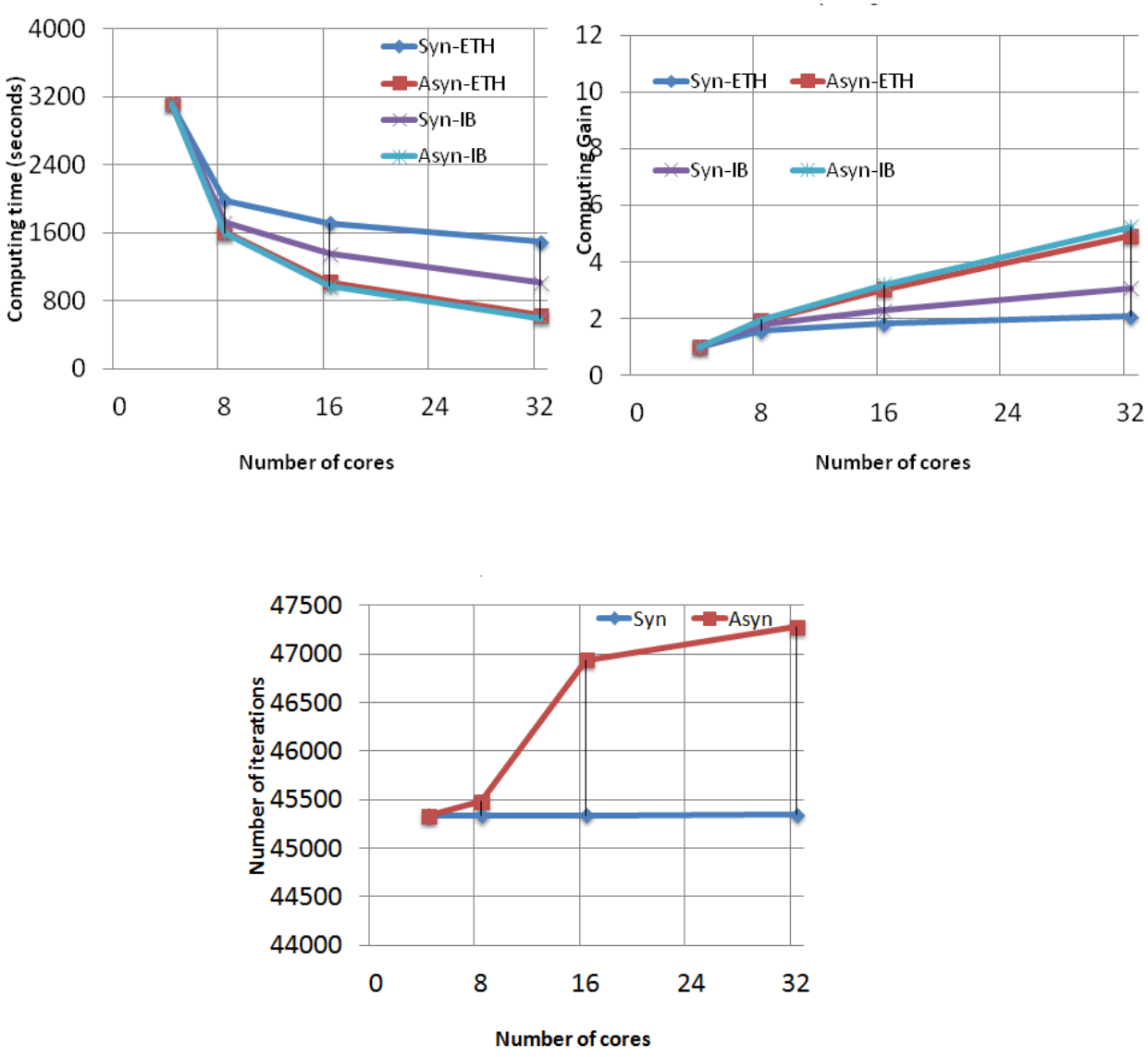

Figure 5.6: Computing results over Ethernet or Infiniband on Graphene cluster in Nancy (four cores per computing node) in the case of the obstacle problem with size 256^3

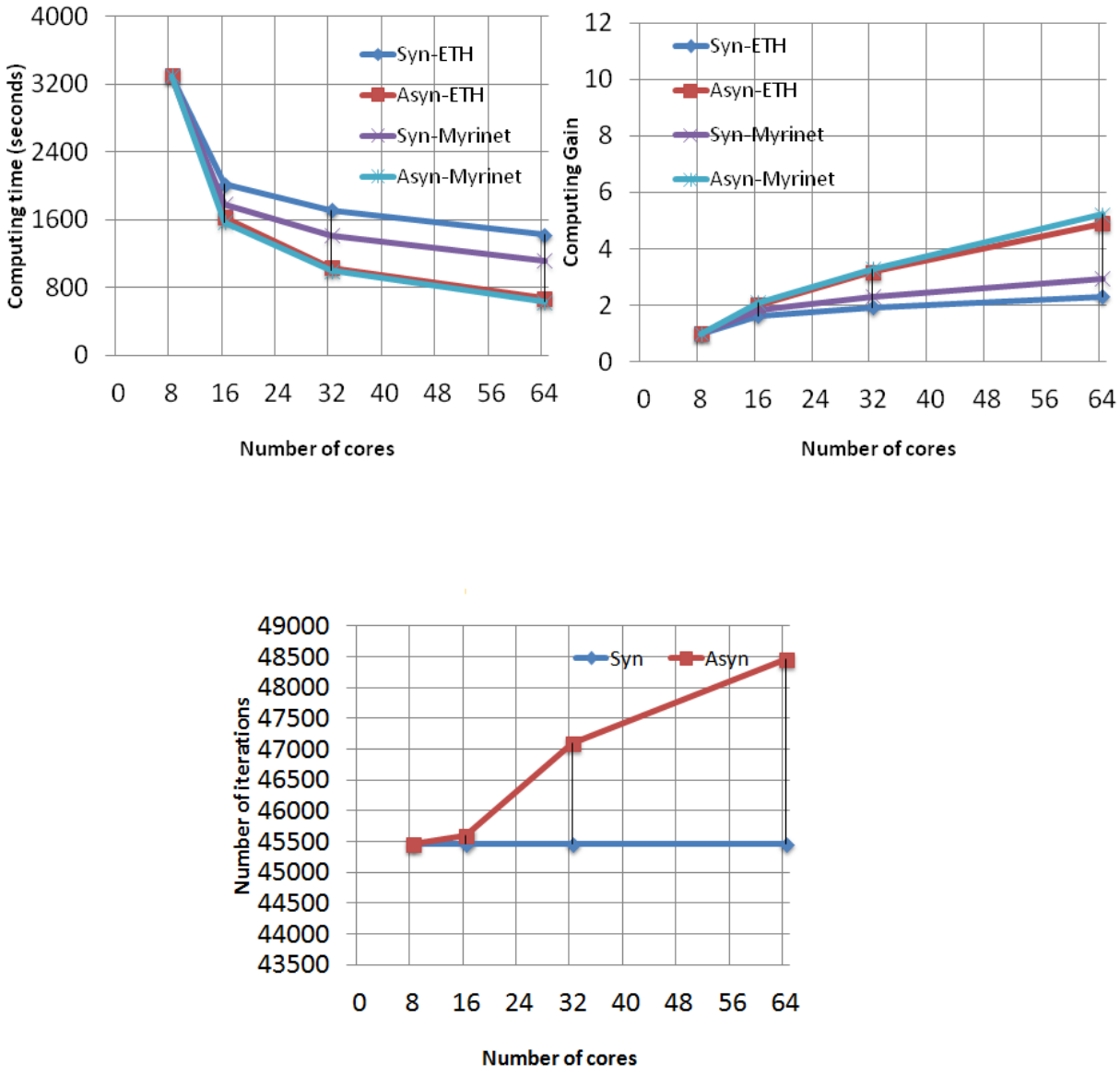

Figure 5.7: Computing results over Ethernet or Myrinet on Chinqchint cluster in Lille (eight cores per computing node) in the case of the obstacle problem with size 256^3

the number of computing nodes for the same reason as above.

For $n = 256$, the parallel time on one multi-core computing node at Chinqchint cluster is equal to 3298 s (see Figure 5.7); while the parallel time on one multi-core computing node at Graphene cluster is equal to 3115 s (see Figure 5.6) . We can deduce that even if the computing nodes at Chinqchint cluster have more computing cores (see table 5.2) as compared to computing nodes at Graphene cluster, the computing nodes at Graphene cluster are faster than the computing nodes at Chinqchint cluster. This is due to the fact that the size of RAM memory at Graphene cluster is greater than the size of memory at Chinqchint cluster. Consequently, more we have RAM memory, more we avoid swapping and more we reduce the time to solve the problem.

The results displayed in Figure 5.8 are obtained with a multi-cluster configuration using machines located in Lille, i.e., Chinqchint cluster and Nancy, i.e., Graphene cluster. Lille and Nancy are two French cities three hundred kilometers apart. The experiments are carried out with up to 24 computing nodes and 128 cores. We note that there is the same number of cores in the different clusters, i.e., 64 cores in Graphene cluster and 64 cores in Chinqchint cluster. Data exchange is made via Infiniband network in Graphene cluster and via Myrinet network in Chinqchint cluster and the communications between clusters are done via 10 Gb/s Ethernet network. Computing results show that even in a heterogeneous context where the computing nodes have different number of cores and there are several networks, the combination of GRIDHPC and asynchronous or hybrid iterative schemes of computation leads to important reduction in computing time. We note that hybrid iterations is situated in between synchronous and asynchronous iterations. This is due to the fact that hybrid iterations is a combination of synchronous and asynchronous iterations, e.g, synchronous iterations in the same cluster and asynchronous iterations between clusters. We note also that the number of iterations performed by synchronous schemes remains almost constant and the average number of iterations in the case of asynchronous or hybrid schemes of computation is more important for the reasons given above.

The results displayed in Figure 5.9 are obtained with a multi-cluster configuration using computing nodes located in Grenoble, i.e., Edel and Genepi clusters of the Grid5000 testbed. The experiments are carried out with up to 32 computing nodes and 256 cores. Data exchange is made via Infiniband network in Edel and Genepi clusters and via Ethernet network (10 Gb/s) between them. We note that asynchronous iterations perform better than synchronous or hybrid iterations. We note also that the number of iterations performed by synchronous schemes remains almost constant and the average number of

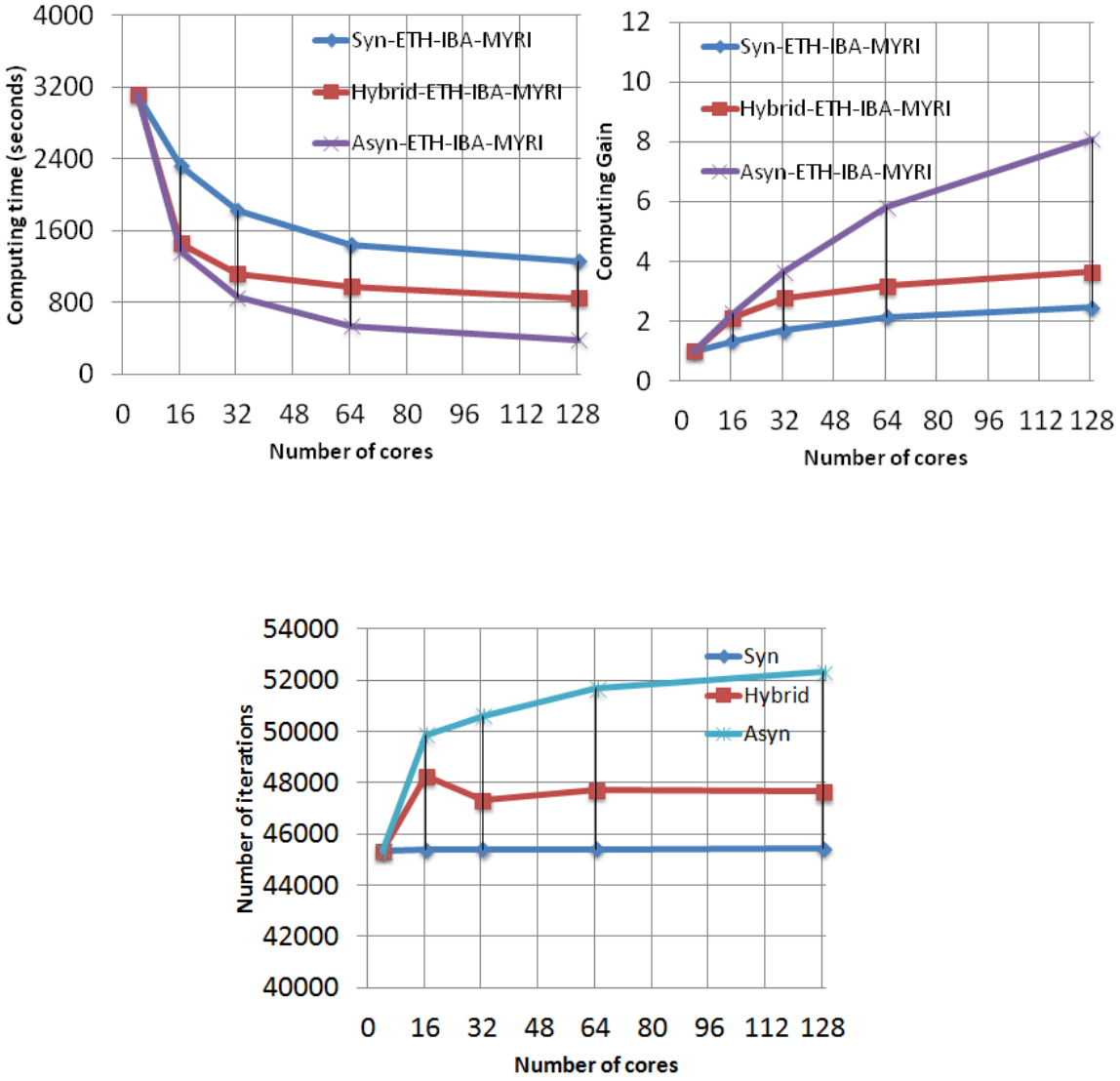

Figure 5.8: Computing results over Ethernet + Infiniband + Myrinet on Chinqchint cluster in Lille (eight cores per computing node and Myrinet) and Graphene cluster in Nancy (four cores per computing node and Infiniband) in the case of the obstacle problem with size 256^3

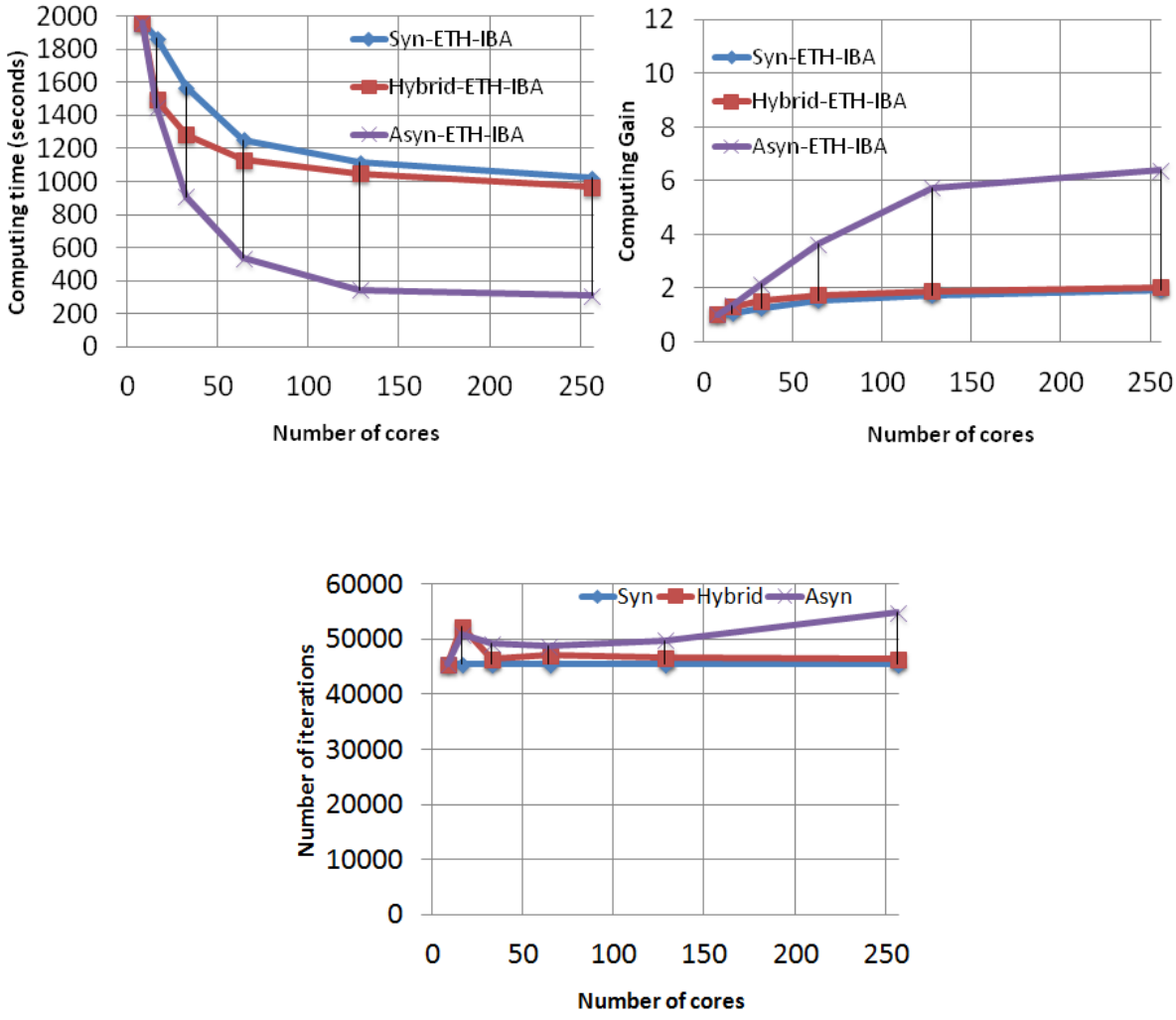

Figure 5.9: Computing results over Ethernet + Infiniband on Edel cluster in Grenoble (eight cores per computing node) and Genepi cluster in Grenoble (eight cores per computing node) in the case of the obstacle problem with size 256^3

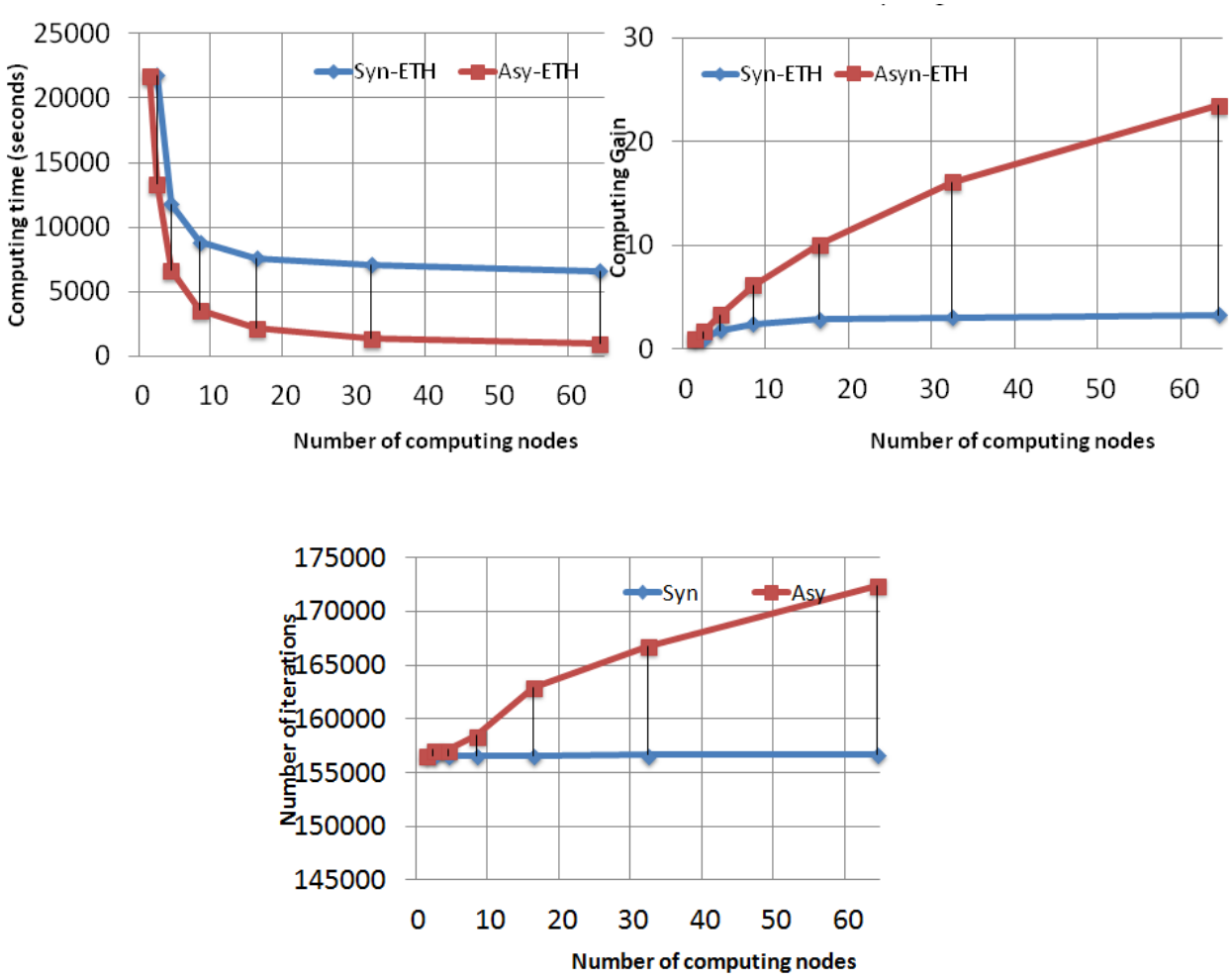

Figure 5.10: Computing results over Ethernet on Paravance cluster in Rennes (16 cores per computing node) in the case of the obstacle problem with size 512^3

iterations in the case of asynchronous or hybrid schemes of computation is more important for the reasons given above.

In the case where $n = 256$, the parallel time on one multi-core computing node of Edel cluster is equal to 1962 s (computing nodes of Edel cluster are faster than computing nodes of Genepi cluster). The parallel time on one multi-core computing node of Graphene cluster is equal to 3115 s (computing nodes of Graphene cluster are faster than computing nodes of Chinqchint cluster). As a comparison between Figure 5.8 and Figure 5.9 in the case of asynchronous schemes of computation for 128 cores, we can see that the computing time in Figure 5.9 (343 s) is less than the computing time in Figure 5.8 (386 s) but the Computing gain in Figure 5.8 (8.08) is greater than the Computing gain in Figure 5.9 (5.72). This is due to the fact that the calculation of Computing gain depends

on the parallel time of the fastest computing node.

The results displayed in Figures 5.10 are obtained using Paravance cluster located in Rennes site with Ethernet network for a problem with around 134 million discretization points that corresponds to $n = 512$. The experiments show that GRIDHPC achieves scalability when it is combined with asynchronous iterative schemes of computation. The experiments are carried out with up to 64 computing nodes and 1024 cores. We note that the number of iterations performed by synchronous schemes remains almost constant and the average number of iterations of asynchronous schemes of computation increases with the number of computing nodes.

Figures 5.6 to 5.10 show that the Computing gain C_g, see equation (5.6), of the synchronous iterative schemes increases slowly with the number of cores and the Computing gain of asynchronous iterative schemes increases rapidly. This is due to the fact that in the case of synchronous iterative schemes of computation fast computing nodes have to wait for slow computing nodes since they are synchronized via messages exchange; this leads to idle time due to synchronization. In the case of asynchronous iterative schemes of computation there is no synchronization and communications are covered by computation; which explains the better Computing gain.

5.4 Conclusion

This chapter presents a first type of HPC application related to the solution of numerical simulation problem: the obstacle problem. This type of problems belongs to the class of loosely synchronous applications. We have presented and analyzed a set of computational experiments with the decentralized environment GRIDHPC for the obstacle problem. In particular, we have studied the combination of GRIDHPC and distributed synchronous and asynchronous iterative schemes of computation for the obstacle problem in a multi-core and multi-network context. Our experiments are carried out on the Grid5000 platform with up to 1024 computing cores. We have treated several cases such as two Infiniband clusters connected via Ethernet or one Infiniband cluster and a Myrinet cluster connected via Ethernet, etc. It follows from all these results that the performance of parallel iterative algorithms depends on several factors:

- networks: type of networks (Ethernet, Infiniband, Myrinet), their latency, bandwidth and topology;

- machines: Number of cores, size of RAM memory, clock frequency, type of processor;
- size of the problem to solve;
- schemes of computation: synchronous, asynchronous or hybrid iterative schemes of computation;
- decomposition method of the problem.

The results show also that the combination of RMNP and OpenMP with GRIDHPC allows to solve efficiently numerical simulation problems via parallel or distributed asynchronous iterative methods. A decomposition method of the obstacle problem has been presented. A convergence detection method and a termination method have been implemented.
We note that in future work, we will implement the pillar decomposition of the three dimensional cube in order to obtain better performance.
In the next chapter, we shall consider a second type of HPC applications that belongs to the class of embarrassingly parallel application, i.e., planning problem.

CHAPTER 6

Planning problem

Contents

6.1 Introduction

This chapter presents a second type of HPC applications related to the planning problem. This type of problems belongs to the class of embarrassingly parallel applications. The remainder of this chapter is organized as follows: section 6.2 presents an introduction to the planning problem. Section 6.3 describes best first search algorithm. Section 6.4 deals with the decomposition technique and the parallel implementation of best first search algorithm thanks to GRIDHPC. Computational results are displayed and analyzed in section 6.5. Finally, section 6.6 concludes this chapter.

6.2 The Planning problem

Planning is finding a sequence of actions that achieves a given goal when executed from a given initial state. Planning problems occur in many domains like planning tasks for satellites, airline crew task planning, autonomous robots task planning and automatic task planning in video games. In general, planning systems solve planning problems doing the following things: model actions and goal representations to allow selection, design metrics to guide search, divide and conquer by sub-goaling to construct final solutions. Note that, in planning problems, we can use specification languages like STRIPS, ADL or PDDL to describe a problem. In the sequel, we will present in details each one.

6.2.1 STRIPS

STRIPS stand for Stanford Research Institute Problem Solver. It is one of the first automated planners developed by Richard Fikes and Nils Nilsson [77]. The main contribution of STRIPS was to separate the process of theorem-proving from those of searching through a space of world model. In the sequel, we will describe certain aspects of the STRIPS language, such the syntax and the semantic.

6.2.1.1 Syntax of STRIPS

The state variables in STRIPS can be described as a set of conjunctions of propositions or first order literals. Literals must be ground (variable free) and everything which is not given explicitly is false (Closed World Assumption is used). In STRIPS, a State s is a conjunction of literals. A state s satisfies a goal state g if g is a subset of s. That is, if each literal of g is also in s; e.g., s = r $\wedge$ f $\wedge$ m satisfies g = r $\wedge$ f.
Actions in STRIPS can be described as follows:

- There is always a name and parameters for the action
- Pre-conditions are conjunctions of only positive literals.
- Effects are always conjunctions of literals. Some planners can distinguish positive (an add list) and negative (delete list) effects.

6.2.1.2 Semantics of STRIPS

This equation shows a semantic description of the STRIPS language:

$$S_{i+1} = trans_{STRIPS}(Si, a) = \begin{cases} (S_i \setminus del(a)) \cup add(a), \ if \ preconditions(a) \subset S_i \\ undefined \ otherwise \end{cases} \quad (6.1)$$

Lets see how it holds for an example given bellow and shown in Figure 6.1:

S_0 = onTable(A) $\wedge$ onTable(C) $\wedge$ on(B,C) $\wedge$ clear(A) $\wedge$ clear(B)

Action (stack(A, B),
PRECOND: clear(A) $\wedge$ clear(B)
EFFECT: on(A,B) $\wedge$ $\neg$ onTable(A) $\wedge$ $\neg$ clear(B))

S_1 = on(A,B) $\wedge$ onTable(C) $\wedge$ on(B,C) $\wedge$ clear(A)

Note that:

- onTable(A) means that block A is on the table
- on(B,C) means that block B is on block C
- clear(A) means that we don't have any block on A

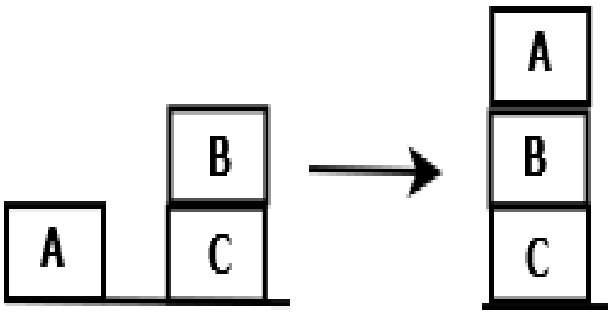

Figure 6.1: Performed actions in STRIPS

6.2.2 ADL

ADL stands for Action Description Language. It is an advanced automated planner of STRIPS proposed by Pednault [78]. It allows conditional operators. Actions in ADL can be described with indirect effects and can be classified into static and dynamic laws. In the sequel, we will present the main difference between STRIPS and ADL.

6.2.2.1 Comparison between STRIPS and ADL

- In STRIPS, the goals are conjunctions, e.g., R $\wedge$ B; while in ADL, the goals can be conjunction and disjunction, e.g., R $\wedge$ B $\vee$ S.
- In STRIPS, we only can find ground literals in goals, e.g., R $\wedge$ B; while in ADL, we can find universally and existentially quantified variables in goals, e.g., $\exists$.
- STRIPS language only allows positive literals in the states, e.g., R $\wedge$ B; while ADL can support both positive and negative literals, e.g., $\neg$P $\wedge$ $\neg$U.
- In STRIPS, the effects are conjunctions; while ADL can use conditional effects, e.g., X : Y means Y is an effect only if X is satisfied.
- In STRIPS, the unmentioned literals are false (closed world assumption); while in ADL, the unmentioned literals are unknown (open world assumption).
- The STRIPS language does not support equality and types; while ADL support them.

6.2.3 PDDL

PDDL, i.e., Planning Domain Definition Language is used to standardize planning domain and problem description languages. It was developed by Drew McDermott and his colleagues in 1998 [85] (inspired by STRIPS and ADL among others). It is a domain definition language which is supported by most planners. It describes a system using a set of preconditions and post-conditions. It is used to define the properties of a domain, the predicates which are used and the action definition. A predicate defines the property of an object which can be true or false.
A PDDL definition consists of two parts: the domain and the problem definition. Note that many planners require that the two parts are in separate files. Note also that domains may declare requirements. The most commonly used requirements are:

- **:strips**
  This requirement means that the domain uses only the STRIPS subset of pddl.
- **:equality**
  This requirement means that the domain uses the predicate =, interpreted as equality.
- **:typing**
  This requirement means that the domain uses types (see subsection typing below).
- **:adl**
  Means that the domain uses some or all of ADL, i.e., disjunctions and quantifiers in preconditions and goals, quantified and conditional effects.

6.2.3.1 Typing

The domain should declare the requirement: typing, if types are to be used in a domain. This is done with the declaration: (:types Name1 ... Name_N). Note that ?A - Type_of_A is to declare the type of a parameter of a predicate or action. Note also that the syntax is the same for declaring types of objects in the problem definition.

6.2.3.2 The Domain Definition

The domain definition contains the domain predicates and actions. It may also contain types, constants, and requirements.
Figure 6.2 presents the format of a simple domain definition. Note that the domain predicates and actions may contain alphanumeric characters like hyphens, i.e., '-' and underscores, i.e., '_'. Note also that the parameters of predicates and actions are distinguished by their beginning with a question mark '?'.

```
(define (domain Domain_Name)
   (:requirements [:strips] [:equality] [:typing] [:adl])
   (:predicates (Predi_Name_1 [?Pred1 ?Pred2 .... ?PredN])
                (Predi_Name_2 [?Pred1 ?Pred2 .... ?PredN])
                ...)

   (:action Act_Name_1
      [:parameters (?Param1 ?Param2 ... ?ParamN)]
      [:precondition Precon_Formulas]
      [:effect Effect_Formulas]
   )

   (:action Act_Name_2
     ... )

  ... )
```

Figure 6.2: Format of a domain definition

6.2.3.3 The Problem Definition

The problem definition contains the objects present in the problem instance, the initial state description and the goal.
Figure 6.3 present the format of a simple problem definition.
The initial state description, i.e., :init is simply a list of all the atoms that are true initially (all other atoms are false); the goal description, i.e., :goal is a formula of the same form as an action precondition.
The mission of a classical planning system is to find a sequence of actions such that if executed from the initial state will achieve the goal state.

```
(define (problem Problem_Name)
   (:domain Domain_Name)
   (:objects Obj1 Obj2 ... ObjN)
   (:init A1 A2 ... AN)
    (:goal Condition_Formulas)
)
```

Figure 6.3: Format of a Problem definition

6.3 Best first search algorithm

Best-first search is an instance of the general graph or tree search algorithms that selects the next node for expansion based on an evaluation function. It falls under the category of Heuristic or Informed Search. The algorithm uses a priority Queue to store the search nodes. The nodes stored in the Queue are ordered accordingly to the evaluation function. The best node is selected for expansion during search. Once a node is selected, each applicable operator (e.g., an action) generates its children nodes, which are ranked by the evaluation function and inserted in the Queue. The algorithm keeps selecting and expanding until a goal node is found, and search is terminated. Best first search is summarized in Algorithm 4 :

- create an empty Queue named q.
- generate the evaluation function (h) of the node 'start'.
- insert the node 'start' in q.
- While true, the algorithm takes the first node of q, this node is denoted by first in the pseudo code of Algorithm 4 (since it corresponds to the smallest evaluation

function) and test the evaluation function. If the evaluation function is equal to zero, then the algorithm reaches the goal and returns; Otherwise, the algorithm uses the available operators of the node 'first' to generate the children 'v' of it, and then inserts these children orderly (ascending order) in q according to the value of the evaluation function of each children.

Algorithm 4: Best First Search Algorithm

```
void Best-First-Search(Node start)
Queue q;
Node first;
h = get(initialStateofstart, GoalState);
q.insert(start);
While True
first = q.TakeFirstNode();
if first->h == 0 then
Exit;
Foreach children 'v' of the node 'first'
q.insertorderly(v);
End procedure
```

Let us consider the example displayed in Figure 6.4.

We start from source "S" and search for goal "P" (evaluation function equal to zero). q initially contains S, we remove S from q and test the evaluation function (h = 6). Since the evaluation function of S #0, the algorithm uses the applicable operators of S to generate the children of it and insert orderly these children to q. q now contains A, B, D, C (D is put before C because the evaluation function of D (5) is less than C (7)).

We remove A from q and process the children of A to q. q now contains E, B, D, F, G, C.

We remove E from q and process the children of E to q. q now contains I, B, D, F, G, H, J, C.

We remove I from q and process the children of I to q. q now contains K, L, B, D, F, G, H, J, C.

We remove K from q and process the children of K to q. q now contains M, L, N, B, D, F, G, H, J, C.

We remove M from q and process the children of M to q. q now contains O, L, N, B, D, F, G, H, J, C.

We remove O from q and process the children of O to q. q now contains P, L, N, B, D, F, G, H, J, C.

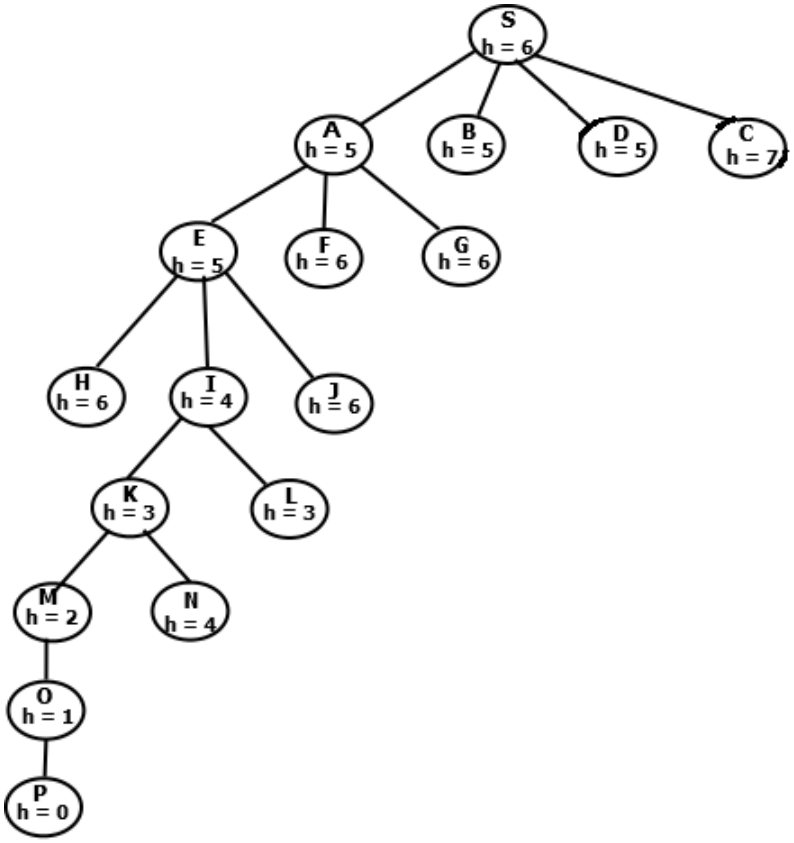

Figure 6.4: Example of Best First Search Algorithm

We remove P from q. Since the evaluation function of P equal to zero, consequently the algorithm reaches the goal and return.
We can see from this example, that a lot of nodes in the queue are not examined like L, N, B, D, F, G, H, J, C. Consequently, our idea is to parallelize the Best-first search algorithm to explore a bigger search space in order to get better solutions. In next section, we will present the parallel version of best first search implemented with GRIDHPC environment. We note that, Whitlock, Dey and Hyatt [90] are the first authors that have proposed a parallel version of Best-first search method. Another simple approach to parallel best first search is Hash-Distributed A* (HDA*) [91], a parallelization of A* algorithm [92]. It distributes and schedules work among processors based on a hash function of the search tree.

6.4 Decomposition and parallel implementation of best first search algorithm using GRIDHPC

In this section, we present the decomposition and the parallel implementation of best first search algorithm using GRIDHPC. Before we present the implementation of GRIDHPC, we detail the decomposition method of best first search algorithm.

6.4.1 Parallel best first search algorithm

This sub-section presents my contribution to the design and development of the parallel best-first search algorithm. It is displayed in Algorithm 5 and does the following things:

- create an empty Queue named q.
- generate the evaluation function (h) of the node 'start'.
- insert the node 'start' in q.
- take the first node of q (this node is denoted by first in the pseudo code of Algorithm 5) and generate the children 'v' of it, then inserts these children orderly (ascending order) in q according to the value of the evaluation function of each children.
- generate a number of threads that is at most equal to the number of computing cores.
- each thread takes one element of q (that is denoted by v in the pseudo code of Algorithm 5) that becomes its initial state.
- each thread run Algorithm 4.
- at the end, the parallel algorithm takes the shortest path of the thread that reaches the goal and returns.

Algorithm 5: Parallel Best First Search Algorithm

```
void Parallel-Best-First-Search(Node start)
Queue q;
Node first;
h = get(initialStateofstart, GoalState);
q.insert(start);
first = q.TakeFirstNode();
if first->h == 0 then
Exit;
Foreach children 'v' of the node 'first'
//Start parallel region
Each thread run algorithm 4
Finish parallel region and takes the shortest path of the thread that reaches the goal
End procedure
```

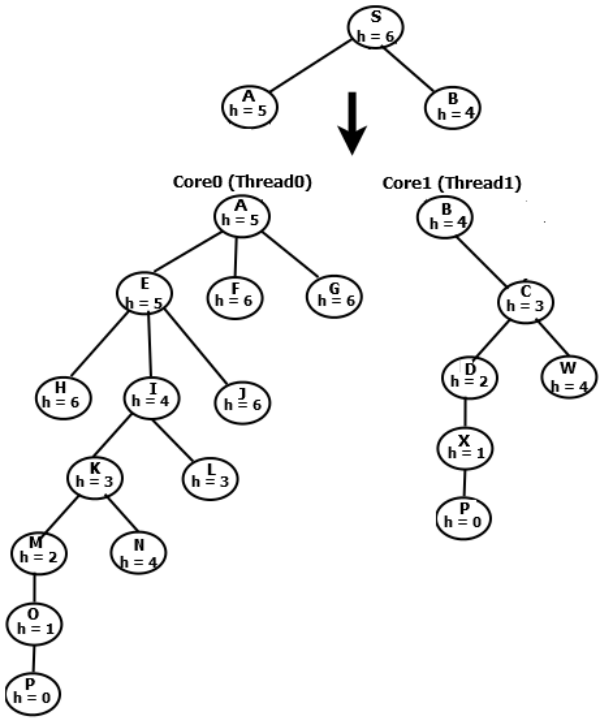

Figure 6.5: Example of Parallel Best First Search Algorithm

6.4.2 Implementation

The parallel implementation of best first search relies on *CPU manager* thanks to GRIDHPC.

To illustrate this implementation, we consider the example displayed in Figure 6.5. For simplicity of presentation, we suppose that we have one computing node composed of two CPU cores.

The environment GRIDHPC uses the **Job Initialization component** to generate and insert the children of S (Two children A and B) in q, then each thread takes one element of q that becomes its intial states.

The environment GRIDHPC uses the **Job Execution component** to run in parallel the thread that are assigned to the cores until the termination detection, i.e., evaluation function equal to zero. At the end, the algorithm takes the shortest path of the thread that reaches the goal and returns, in our case, the thread assigned to core 1 terminates first.

6.5 Evaluation and computing results

6.5.1 Blocks World Problem

Blocks world problem is a micro-world that consists of a table, a set of blocks and a robot hand (see Figure 6.1). Here are the classic basic operations of it:

- stack(X,Y) : put block X on block Y
- unstack(X,Y) : remove block X from block Y
- pickup(X) : pickup block X
- putdown(X) : put block X on the table

Each operation is represented by a list of preconditions, a list of new facts to be added (add effects), a list of facts to be removed (delete-effects) and a set of variables constraints (optionally). An example of definition of pick-up operation is given bellow:

```
(:action pick-up
   :parameters (?x - block)
   :precondition (and (clear ?x) (ontable ?x) (handempty))
   :effect
   (and (not (ontable ?x))
   (not (clear ?x))
   (not (handempty))
   (holding ?x)))
```

for more details about the syntax of operations, the readers may refer to the section 6.2.3 (PDDL).

Problem(BLOCKS)	Sequential (Number of actions)		Parallel		
Strips	Computing Time (second)	number of actions	Computing Time (second)	number of actions	Number of Sublists (also number of cores)
probBLOCKS-7-2.pddl	0.0013	38	0.004	28	2
probBLOCKS-8-0.pddl	0.004	46	0.01	38	4
probBLOCKS-8-1.pddl	0.0019	28	0.008	24	4
probBLOCKS-11-0.pddl	0.03	80	0.06	78	3
probBLOCKS-12-0.pddl	0.08	130	0.29	116	3
probBLOCKS-12-1.pddl	0.06	118	0.12	110	2
probBLOCKS-13-0.pddl	10.66	190	2.35	128	3
probBLOCKS-13-1.pddl	10.36	176	1.24	128	2
probBLOCKS-14-1.pddl	0.04	110	0.36	94	5
probBLOCKS-15-0.pddl	1.002	142	0.31	92	5
probBLOCKS-16-1.pddl	2.74	222	3.68	174	3
probBLOCKS-16-2.pddl	3.09	204	5.506	204	2
probblocks-17-0.pddl	69.71	286	5.66	206	5
probblocks-19-1.pddl	6.13	270	1.4	164	3

Table 6.1: Solution of Blocks World Problem

6.5.2 Experimental results

Table 6.1 presents the solution of several instances of blocks world problem solved in sequential and parallel version of best first search algorithm. An example of an instance 'probBLOCKS-7-2' of the problem is given bellow :

(define (problem BLOCKS-7-2)
(:domain BLOCKS)
(:objects E G C D F A B - block)
(:INIT (CLEAR B) (CLEAR A) (ONTABLE F) (ONTABLE D) (ON B C) (ON C G) (ON G E) (ON E F) (ON A D) (HANDEMPTY))
(:goal (AND (ON E B) (ON B F) (ON F D) (ON D A) (ON A C) (ON C G)))
)

Problem(Satellite)	Sequential		Parallel		
Strips	Computing time (second)	(number of actions)	Computing Time (sec)	number of actions	Number of Sublists (also number of cores)
P01	0.0003	9	0.025	9	7
P02	0.001	13	0.006	13	9
P03	0.004	12	0.024	11	18
P04	0.008	18	0.02	18	20
P05	0.032	16	0.116	16	36
P06	0.03	20	0.122	20	34
P07	0.072	22	0.32	21	40
P08	1.29	27	0.82	26	40
P09	0.69	30	1.20	29	40
P10	0.85	33	1.55	30	40
P11	1.02	33	1.58	32	40
P13	3.92	61	5.5	57	40
P14	4.83	43	3.15	41	40
P15	17.41	53	21.8	53	40

Table 6.2: Solution of Satellite Problem

Problem(Pipes World)	Sequential		Parallel		
NonTankageNonTemporal	Computing time (second)	number of actions	Computing Time (sec)	number of actions	Number of Sublists (also number of cores)
P01	0.0003	5	0.003	5	6
P02	0.0005	12	0.004	12	6
P03	0.001	12	0.005	9	7
P04	0.001	11	0.007	11	7
P05	0.001	9	0.01	9	10
P06	0.001	12	0.01	12	10
P07	0.001	9	0.01	9	9
P08	0.002	11	0.03	11	9
P09	0.008	14	0.04	14	11
P10	0.01	25	0.05	24	11
P11	0.05	32	0.13	24	8
P13	0.54	42	0.55	18	11
P15	0.02	46	0.6	38	6
P16	0.36	82	0.56	78	6
P17	5.56	26	0.62	22	13
P18	9.85	66	1.35	62	13
P20	0.63	40	1.08	36	14
P23	0.08	30	0.67	20	12

Table 6.3: Solution of Pipes World Problem

The computing time and the number of actions needed to reach the goal of the instance 'probBLOCKS-7-2' from the initial state in sequential version of best first search algorithm are equal to 0.0013 s and 38 actions (see Table 6.1); while the computing time and the number of actions needed to reach the goal of the same instance in parallel version are equal to 0.004 s and 28 actions (see Table 6.1). As a comparison, we can see that the number of actions in parallel version is in generally less than in sequential version. This is due to the fact that the parallel version of the algorithm examines more nodes (comparing to sequential version) and takes the shortest path of all of them. We note that the computing time of the parallel version is calculated using gettimeofday() function, i.e., time spent from beginning to end of the computation.

6.5.3 Other planning problems

Other planning problems have been solved like satellite and pipes world problems. Table 6.2 and Table 6.3 presents the solution of several instances of these problems solved in sequential and parallel version of best first search algorithm. As a comparison between the parallel version and the sequential version of theses problems, we can deduce the same reasons as above : the number of actions in parallel version is in generally less than in sequential version. This is due to the fact that the parallel version of the algorithm examines more nodes (comparing to sequential version) and takes the shortest path of all of them.
Finally, we note that all these instances are downloaded from the international planning competition, i.e., http://idm-lab.org/wiki/icaps/ipc2004/deterministic/. We note also that the experiments are carried out at Chetemi cluster (Lille site) of the Grid5000 platform with up to two computing nodes and a total of 40 computing cores. This cluster is equipped of Intel Xeon E5-2630, with 20 cores, clock 2.20 GHz and 256 GB of RAM per machine.

6.6 Conclusion

This chapter presents a second type of HPC applications related to the the planning problem. This type of problems belongs to the class of embarrassingly parallel applications. An introduction to the planning problems has been described. A decomposition method has been presented and implemented thanks to GRIDHPC. We have presented and analyzed also a set of computational experiments with the decentralized environment GRIDHPC. The experiments are carried out at Chetemi cluster of the Grid5000 platform

with up to two computing nodes and a total of 40 computing cores. They show that the combination of OpenMP with GRIDHPC allows to solve efficiently the problems.
In future work, we plan to implement a different method including a reasonable timer in order to get the best solution amongst the different cores. We plan also to implement and execute parallel best first search algorithm on many computing nodes for harder instances.

Conclusions

In this manuscript, we have presented our contributions to grid computing. In Chapter 3 we have presented RMNP, a Reconfigurable Multi-Network communication Protocol dedicated to HPC applications. We describe the global architecture of RMNP for HPC applications with its main functionalities to allow rapid data exchange between computing nodes in multi-network configurations like Ethernet, Infiniband and Myrinet via parallel or distributed iterative algorithms. The protocol can configure itself automatically and dynamically in function of application requirements like schemes of computation, e.g. synchronous iterations or asynchronous iterations, elements of context like network topology and type of network like Ethernet, Infiniband and Myrinet by choosing the best communication mode between computing nodes and the best networks.
In Chapter 4, we have presented the decentralized environment GRIDHPC for grid computing. We have described the global topology and the general architecture of the decentralized environment GRIDHPC with its main functionalities. We have presented the hierarchical task allocation mechanism that accelerates task allocation to computing nodes and avoids communication bottleneck at submitter; a programming model for GRIDHPC that facilitates the work of programmers has been presented. In particular, the communication operation set is reduced with only three operations, basically send, receive and wait operations. GRIDHPC facilitates the use of multi-cluster and grid platform for loosely synchronous applications and also embarrassingly parallel application. It exploits all the computing resources (all the available cores of computing nodes) as well as several type of networks like Ethernet, Infiniband and Myrinet in the same application. The functionality of GRIDHPC relies on the reconfigurable multi-network protocol RMNP for controlling multiple network adapters and on OpenMP for the exploitation of all the available cores of computing nodes. These features involved the developments of helper programs. These programs are responsible for the analysis of the application and building the network topology. It relies on two pillars of GRIDHPC environment namely the CPU manager and the Network selection manager that are in charge of data exchange between computing cores, i.e., reading/writing and between computing nodes via the best underlying network, i.e., high speed and low latency network like Infiniband and Myrinet.
In Chapter 5, we have considered a first type of HPC application related to the solution

of numerical simulation problem: the obstacle problem. This type of problems belongs to the class of loosely synchronous applications. We have presented and analyzed a set of computational experiments with the decentralized environment GRIDHPC for the obstacle problem. In particular, we have studied the combination of GRIDHPC and distributed synchronous and asynchronous iterative schemes of computation for the obstacle problem in a multi-core and multi-network context. Our experiments are carried out on the Grid5000 platform with up to 1024 computing cores. In this chapter, we have treated several cases such as two Infiniband clusters connected via Ethernet or one infiniband cluster and a Myrinet cluster connected via Ethernet, etc. The results show that the performance of parallel iterative algorithms depends on several factors such as networks (type of networks, latency, bandwidth and topology), machines (number of cores, size of RAM memory, clock frequency and type of processor), size of the problem to solve, schemes of computation (synchronous, asynchronous or hybrid iterative schemes of computation) and the decomposition of the problem.

The results show also that the combination of RMNP and OpenMP with GRIDHPC allows to solve efficiently numerical simulation problems via parallel or distributed asynchronous iterative methods. A decomposition method of the obstacle problem has been presented. A convergence detection method and a termination method have been implemented.

In Chapter 6, we have considered a second type of HPC applications related to the planning problems. This type of problems belongs to the class of embarrassingly parallel applications. We have presented and analyzed a set of computational experiments with the decentralized environment GRIDHPC. A decomposition method has been presented and implemented thanks to GRIDHPC. The experiments are carried out at the Chetemi cluster of the Grid5000 platform with up to two computing nodes and a total of 40 computing cores. They show that the combination of OpenMP with GRIDHPC allows to solve efficiently the problem.

In future work, on what concerns the obstacle problem, we will implement the pillar decomposition of the three dimensional cube in order to obtain better performance.

On what concerns planning problem, we plan to implement a different method including a reasonable timer in order to get the best solution amongst the different cores. We plan also to implement and execute parallel best first search algorithm on many computing nodes for hard instances.

Other applications must be considered. In particular, several logistic applications have to be treated as well as others numerical simulation applications.

Nowadays, GRIDHPC treats multi-core and multi-network configurations. The com-

bination of GRIDHPC with a new approach like GPU computing and Intel Xeon Phi computing deserves also to be investigated.

APPENDIX A

Run GRIDHPC applications

A.1 Run GRIDHPC applications

- Copy the GRIDHPC folder from your home directory to the Lille site for example.
  scp –r ./P2PDC login@access.grid5000.fr:./Lille
- Reservation of machines on a cluster in GRID5000:

  oarsub -I -t deploy -l nodes=6,"walltime='2'" -p "cluster='paravance'"
- Deployment of wheezy environment:

  kadeploy3 -f $OAR_FILE_NODES -e wheezy-x64-nfs
- Compile the Server, Tracker, P2PComm, Peer and obstacle.6.0
  cd GRIDHPC/Peer/P2PComm
  make
- Run a server in the Server folder:
  cd GRIDHPC/Server
  ./Server
  Modify the IP address (or domain name) of server in GRIDHPC/Tracker/db/Server and GRIDHPC/Peer/data/Server Files.
- Run a Tracker in Tracker folder:
  cd GRIDHPC/Tracker
  ./Tracker
- Start worker in Peer folder:
  cd GRIDHPC/Peer
  $LD_LIBRARY_PATH= /home/your_account_grid5000/GRIDHPC/Peer/P2PComm
  export $LD_LIBRARY_PATH.
  ./P2PDC [netif_name0][netif_name1][number of netif]
  where
  -netif_name0 is the network interface used to communicate with others workers on Ethernet network, e.g. eth0 or eth1.
  -netif_name1 is the network interface used to communicate with others workers on Infiniband or Myrinet network, e.g. ib0 or myri0.
  -number of netif is the number of network interface card used to communicate with

others workers, e.g. 1 or 2.

- Start submitter in Peer folder:
  cd GRIDHPC/Peer
  ./P2PDC [netif_name] [problem_name] [size of the problem] [scheme of computation] [number of workers]
  where
  -netif_name is the network interface used to communicate with others workers, e.g. eth0 or eth1;
  -problem_name is the name of the problem, e.g. obstacle.6.0.
  - size of the problem, e.g. 128, 256, 512 ...
  - scheme of computations : Synchrone = 1, Asynchrone = 2, Hybrid = 3.
  - number of workers are equal to the power of 2.

APPENDIX B

List of publications

B.1 Papers in international conferences and journal

[1] B.Fakih, D.Elbaz,'Heterogeneous Computing and Multi-Clustering Support via Peer-To-Peer HPC', 26th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Cambridge, UK, March 2018.

[2] B.Fakih, D.Elbaz, Igor Kotenko, 'GRIDHPC, A Decentralized Environment for High Performance Computing', soumis à une revue scientifique internationale.

Bibliography

[1] K. Hwang, G. Fox and J. Dongarra. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. *5 citations pages xv, 8, 9, 22, and 30*

[2] OpenMP, http://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf *Cited on page 48*

[3] Grid5000 platform, http://www.grid5000.fr. [Online]. Available: http://www.grid5000.fr. *2 citations pages 40 and 74*

[4] L. Bouge, J.-F. Mehaut, R. Namyst. MADELEINE: an efficient and portable communication interface for RPC-based multithreaded environments. Parallel Architectures and Compilation Techniques, 1998. Proceedings. 1998 International Conference on. *Cited on page 13*

[5] O. Aumage; L. Bouge; A. Denis; J.-F. Mehaut; G. Mercier; R. Namyst; L. Prylli. Madeleine II: a portable and efficient communication library for high-performance cluster computing. Proceedings IEEE International Conference on Cluster Computing. CLUSTER 2000. *Cited on page 13*

[6] O. Aumage, G. Mercier, MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks, 15th International Parallel and Distributed Processing Symposium (IPDPS'01), 2001. *Cited on page 13*

[7] O. Aumage, L. Bouge, and R. Namyst. A Portable and Adaptative Multi-Protocol Communication Library for Multithreaded Runtime Systems. In Parallel and Distributed Processing. Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP'00), volume 1800 of Lect. Notes in Comp. Science, pages 1136–1143, Cancun, Mexico, May 2000. Held in conjunction with IPDPS 2000. IEEE TCPP and ACM, Springer-Verlag. *Cited on page 13*

[8] David P. Anderson, BOINC: A System for Public-Resource Computing and Storage," 5th IEEE/ACM International Workshop on Grid Computing.November 8, 2004, Pittsburgh, USA. *Cited on page 16*

[9] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg, OurGrid: An approach to easily assemble grids with equitable resource sharing, in Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, pp.61-68, June 2003. *Cited on page 17*

[10] B. Cornea, J. Bourgeois, T. T. Nguyen, and D. El Baz, Performance prediction in a decentralized environment for peer-to-peer computing, in Proceedings of the

25th IEEE Symposium IPDPSW 2011 / HOTP2P 2011, Anchorage, USA, 2011, pp. 1613—1621. *Cited on page 18*

[11] D.El Baz, T. T. Nguyen, A self-adaptive communication protocol with application to high performance peer to peer distributed computing, in Proceedings of the 18th Euromicro conference on Parallel, Distributed and Network-Base Processing, Pisa, Italy, 2010. *4 citations pages 2, 10, 18, and 48*

[12] I. Foster and C. Kesselman. The globus project: a status report. Futur Generation Computer System, 40:35–48,1999. *Cited on page 23*

[13] A. Grimhaw and W. Wulf. The legion vision of a worldwide virtual computer. Communications of the ACM, 40, Juanary 1997. *Cited on page 21*

[14] XtremWeb, https://www.xtremweb.net/. *Cited on page 24*

[15] D. Caromel, A. di Costanzo, L. Baduel and S. Matsuoka. Grid'BnB: HiPC'07, Goa, India, December 2007. *Cited on page 23*

[16] C. Augonnet, S.Thibault, R. Namyst, and Pierre A.Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In Proceedings of the 15th Euro-Par Conference, Delft, The Netherlands, August 2009. *Cited on page 26*

[17] T. T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and M. Chau, High performance peer-to-peer distributed computing with application to obstacle problem, in Proceedings of the 24th IEEE Symposium IPDPSW 2010 / HOTP2P, Atlanta, USA, 2010. *3 citations pages 2, 10, and 48*

[18] P. Spitéri and M. Chau. Parallel Asynchronous Richardson Method for the Solution of Obstacle Problem. In Proc. of the 16th Annual International Symposium on High Performance Computing Systems and Applications, pages 133-138, Moncton, Canada, 2002. *3 citations pages 64, 65, and 66*

[19] Jacques L.Lions. Quelques méthodes de résolution des problèmes aux limites non linéaires. Dunod, 2002 *Cited on page 64*

[20] Jean C.Miellou and P.Spitéri. Two criteria for the convergence of asynchronous iterations. In Computers and computing, pages 91-95, 1985. *2 citations pages 64 and 66*

[21] Jean C.Miellou and P.Spitéri. Un critère de convergence pour des methodes generales de point fixe. Modélisation mathématique et analyse numérique, vol. 19, no. 4, pages 645-669, 1985 *Cited on page 66*

[22] L.Giraud and P.Spitéri. Résolution parallèle de problèmes aux limites non linéaires. Modélisation mathématique et analyse numérique, vol. 25, no. 5, pages 579-606, 1991 *Cited on page 66*

[23] D. El Baz. M-functions and Parallel Asynchronous Algorithms. SIAM Journal on Numerical Analysis, vol. 27, no. 1, pages 136-140, 1990. *2 citations pages 8 and 66*

[24] D. El Baz. Nonlinear systems of equations and parallel asynchronous iterative algorithms. Advances in Parallel Computing, vol. 9, pages 89-96, 1994. *2 citations pages 8 and 66*

[25] D. El Baz. Contribution à l'algorithmique parallèle. Le concept d'asynchronisme : étude théorique, mise en oeuvre et application. Habilitation à diriger des recherches, 1998. *2 citations pages 8 and 66*

[26] D.Bertsekas and D.El Baz. Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems. SIAM Journal on Control and Optimization, vol. 25, no. 1, pages 74-85, 1987. *2 citations pages 8 and 66*

[27] Dimitri P. Bertsekas and John N. Tsitsiklis. Parallel and distributed computation: numerical methods. Prentice-Hall, Inc. (republished in 1997 by Athena Scientific), Upper Saddle River, NJ, USA, 1989 *Cited on page 66*

[28] T.T.Nguyen (2011). An environment for peer-to-peer high performance computing. University of Toulouse, Toulouse, France. *Cited on page 48*

[29] J.Zhao and Jian D.Lu. Solving Overlay Mismatching of Unstructured P2P Networks using Physical Locality Information. In Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing, pages 75-76, Washington, DC, USA, 2006. *Cited on page 53*

[30] Gary T. Wong, Matti A. Hiltunen and Richard D. Schlichting. A Configurable and Extensible Transport Protocol. In Proceedings of IEEE INFOCOM, pages 319-328, 2001. *2 citations pages 30 and 33*

[31] Matti A. Hiltunen and Richard D. Schlichting. The Cactus Approach to Building Configurable Middleware Services. In Proceedings of the Workshop on Dependable System Middleware and Group Communication, Nuremberg, Germany, October 2000. *3 citations pages 30, 32, and 33*

[32] Norm Hutchison and Larry L. Peterson. The x-kernel: An architecture for implementing network protocols. In IEEE Transactions on Software Engineering, volume 17, pages 64-76, 1991. *2 citations pages 30 and 31*

[33] H.Miranda, A.Pinto, and L. Rodrigues, Appia: A flexible protocol kernel supporting multiple coordinated channels. in Proc. 21st International conference on Distributed Computing Systems (ICDCS-21), (Phoenix, Arizona, USA), pp.707-710,2001. *Cited on page 31*

[34] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu, Coyote: a system for constructing fine-grain configurable communication services, in ACM Transactions on Computer Systems, 16(4): pp. 321– 366, 1998. *Cited on page 31*

[35] D. C. Schmidt, D. F. Box, and T. Suda, ADAPTIVE—A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment, Journal of Concurrency: Practice and Experience, 5(4): pp. 269–286, 1993. *Cited on page 31*

[36] E. Exposito, P. Senac, M. Diaz, FPTP: the XQoS aware and fully programmable transport protocol, in Networks, 2003. ICON2003. The 11th IEEE International Conference on, pp. 249-254. *Cited on page 32*

[37] Transmission Control Protocol (TCP), in RFC 793, 1981. *Cited on page 37*

[38] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, 1999. *Cited on page 42*

[39] E. Kohler, M. Handley and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 2582, 1999. *Cited on page 42*

[40] T. Heller, H. Kaiser, and K. Iglberger. Application of the ParalleX Execution Model to Stencil-based Problems. In Proceedings of the International Supercomputing Conference ISC'12, Hamburg, Germany, 2012. *Cited on page 25*

[41] T. Heller, H. Kaiser, A. Schäfer, and D. Fey. Using HPX and LibGeoDecomp for Scaling HPC Applications on Heterogeneous Supercomputers. In Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '13, pages 1:1–1:8, New York, NY, USA, 2013. ACM. *Cited on page 25*

[42] H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey. HPX: A Task Based Programming Model in a Global Address Space. In Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14, pages 6:1–6:11, New York, NY, USA, 2014. ACM. *Cited on page 25*

[43] H. Kaiser, T. Heller, A. Berge, and B. Adelstein-Lelbach. HPX V0.9.10: A general purpose C++ runtime system for parallel and distributed applications of any scale, 2015. http://github.com/STEllAR-GROUP/hpx. *Cited on page 25*

[44] Boost: a collection of free peer-reviewed portable C++ source libraries, 1998-2015. http://www.boost.org/. *Cited on page 25*

[45] H. Kaiser, M. Brodowicz, and T. Sterling. ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. In Parallel Processing Workshops, pages 394–401, Los Alamitos, CA, USA, 2009. IEEE Computer Society. *Cited on page 25*

[46] T.Sterling. ParalleX Execution Model V3.1, 2013. *Cited on page 25*

[47] M. Stumpf. Distributed GPGPU Computing with HPXCL, 2014. Talk at LA-SiGMA TESC Meeting, LSU, Baton Rouge, Louisiana, September 25, 2014. *Cited on page 25*

[48] K. Huck, S. Shende, A. Malony, H. Kaiser, A. jh, R. Fowler, and R. Brightwell. An early prototype of an autonomic performance environment for exascale. In Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers, ROSS '13, pages 8:1–8:8, New York, NY, USA, 2013. ACM. *Cited on page 25*

[49] F.Magoulès, J.Pan, Kiat A.Tan and A.Kumar. Introduction to grid computing, volume 10. CRC Press, 2009. *Cited on page 20*

[50] Condor Team, Condor Version 6.6.9 Manual, <http://www.cs.wisc.edu/condor/manual/v6.6.9/condor-V6_6_9-Manual.pdf> May 25 2005. *Cited on page 22*

[51] T.Tannenbaum, D.Wright, K.Miller, and M.Livny. Condor – A distributed job scheduler. In Thomas Sterling, editor, Beowulf Cluster Computing with Linux. MIT Press, October 2001. *Cited on page 22*

[52] J.Basney and M.Livny. Deploying a high throughput computing cluster. In Rajkumar Buyya, editor, High Performance Cluster Computing: Architectures and Systems, Volume 1. Prentice Hall PTR, 1999. *Cited on page 22*

[53] J.Frey, T.Tannenbaum, I.Foster, M.Livny, and S.Tuecke. Condor-G: A computation management agent for multi-institutional grids. In Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC), pages 7–9, San Francisco, California, August 2001. *Cited on page 23*

[54] I.Foster, and C.Kesselman, The Globus Project: A Progress Report, In Proceedings of the Heterogeneous Computing Workshop (Mar. 1998), ftp://ftp.globus.org/pub/globus/papers/globus-hcw98.pdf. *Cited on page 21*

[55] I.Foster, and C.Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications 11, 2 (1997), 115-128, ftp://ftp.globus.org/pub/globus/papers/globus.pdf. *Cited on page 21*

[56] The Globus Project, Globus Toolkit 1.1.3 Sytem Administration Guide, University of Shouthern California, http://www.globus.org, December 2000 *Cited on page 21*

[57] I.Foster, and S.Tuecke , Nexus: Runtime Support for Task-parallel Programming Languages, ftp://ftp.globus.org/pub/globus/papers/nexus_paper_ps.pdf , TR, ANL, 1994. *Cited on page 21*

[58] I.Foster, C.Kesselman, and S.Tuecke, The Nexus Task-parallel Runtime System, In Proc. 1st Intl Workshop on Parallel Processing. Tata McGraw Hill, 1994, pp. 457-462, ftp://ftp.globus.org/pub/globus/papers/india_paper_ps.pdf. *Cited on page 21*

[59] S.Fitzgerald, I.Foster, C.Kesselman, G.von Laszewski, W.Smith, and S.Tuecke, A Directory Service for Configuring High-performance Distributed Computations, In Proc. 6th IEEE Symp. on High Performance Distributed Computing (1997), IEEE Computer Society Press, pp. 365-375, ftp://ftp.globus.org/pub/globus/papers/hpdc97-mds.pdf. *Cited on page 21*

[60] I.Foster, and G.von Laszewski, Usage of LDAP in Globus, TR, ANL, 1997, ftp://ftp.globus.org/pub/globus/papers/ldap_in_globus.pdf. *Cited on page 21*

[61] P.Stelling, I.Foster, C.Kesselman, C.Lee, and G.von Laszewski, A Fault Detection Service for Wide Area Distributed Computations, In Proc. 7th IEEE Symp. on High Performance Distributed Computing (July 1998), IEEE Computer Society Press, ftp://ftp.globus.org/pub/globus/papers/hbm.pdf. *Cited on page 21*

[62] K.Czajkowski, I.Foster, N.Karonis, C.Kesselman, S.Martin, W.Smith, and S.Tuecke, A Resource Management Architecture for Metacomputing Systems., In The 4th Workshop on Job Scheduling Strategies for Parallel Processing (Mar. 1998), IEEE-P, pp. 4-18, ftp://ftp.globus.org/pub/globus/papers/gram97.pdf. *2 citations pages 21 and 22*

[63] University of Virginia, Legion 1.8 System Administrator Manual, http://legion.virginia.edu, 2001. *Cited on page 21*

[64] D.Caromel, C.Delbe, A.di Costanzo, M.Leyton : Proactive: an integrated platform for programming and running applications on grids and p2p systems. *Cited on page 23*

[65] E. Lusk and W. Gropp. MPICH Working Note : the implementation of the second generation ADI. Technical report, Argonne National Laboratory. *Cited on page 13*

[66] E. Lusk and W. Gropp. MPICH Working Note : The Second Generation ADI for the MPICH Implementation of MPI. Technical report, Argonne National Laboratory, 1996. *Cited on page 13*

[67] D. El Baz. An efficient termination method for asynchronous iterative algorithms on message passing architectures. In Proceedings of the international conference on parallel and distributed computing systems, Dijon, volume 1, pages 1-7, 1996. *Cited on page 71*

[68] I.Foster, C.Kesselman, and Tsudick, S. T.G., A Security Architecture for Computational Grids, In Proc. of the 5th ACM Conference on Computer and Communication Security (Nov. 1998), ACM Press, ftp://ftp.globus.org/pub/globus/papers/security.pdf *Cited on page 21*

[69] I.Foster, Karonis, N. T., C.Kesselman, and S.Tuecke, Managing Security in High-performance Distributed Computations, Cluster Computing 1, 1 (1998), 95-107, ftp://ftp.globus.org/pub/globus/papers/cc-security.pdf. *Cited on page 21*

[70] J.Dean and S.Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI, page 13, 2004. *Cited on page 19*

[71] K.Lee, T.Woong Choi, A.Ganguly, David I. Wolinsky, P. Oscar Boykin and R.Figueiredo. Parallel Processing Framework on a P2P System Using Map and Reduce Primitives. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pages 1602-1609, May 2011. *Cited on page 19*

[72] P2P-MPI: A Peer-to-Peer Framework for Robust Execution of Message Passing Parallel Programs on Grids, Stéphane Genaud and Choopan Rattanapoka, in Journal of Grid Computing, volume 5(1), pages 27-42, Springer, ISSN:1570-7873 2007. *Cited on page 20*

[73] B.Carpenter, V.Getov, G.Judd, A.Skjellum, G.Fox: Mpj: Mpi-like message passing for java. Concurr. Pract. Exp. 12(11), 1019–1038 (2000). *Cited on page 20*

[74] Seti@home. http://setiathome.berkeley.edu/ *Cited on page 24*

[75] Genome@home. http://genomeathome.stanford.edu *Cited on page 24*

[76] N.A. Al-Dmour and W.J. Teahan. ParCop: a decentralized peer-to-peer computing system. In Parallel and Distributed Computing, 2004. Third International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on, pages 162-168, July 2004. *Cited on page 18*

[77] STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. R. E. Fikes, N.J. Nilsson *Cited on page 86*

[78] Pednault. Formulating multi-agent dynamic-world problems in the classical planning framework. In Michael Georgeff and Amy Lansky, editors, Reasoning about actions and plans pages 47-82. Morgan Kaufmann, San Mateo, CA, 1987. *Cited on page 88*

[79] W.Pawel, R.Olivier and S.Andre. SAMOA: Framework for Synchronisation Augmented Microprotocol Approach. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, 2004 *Cited on page 31*

[80] TOP500. http://www.top500.org/. *Cited on page 2*

[81] T. Gunarathne, T. Wu, J. Qiu, G. Fox, Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications, in: Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference, Chicago, IL, 20–25 June 2010. *Cited on page 10*

[82] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, G. Fox, High Performance Parallel Computing with Clouds and Cloud Technologies, Cloud Computing and Software Services: Theory and Techniques, CRC Press (Taylor and Francis), 2010. *Cited on page 10*

[83] J. Ekanayake, T. Gunarathne, J. Qiu, Cloud Technologies for Bioinformatics Applications, IEEE Trans. Parallel Distrib. Syst., (2010). *Cited on page 10*

[84] J. Qiu, T. Gunarathne, J. Ekanayake, J. Choi, S. Bae, H. Li, et al., Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications, in: 11th Annual Bioinformatics Open Source Conference BOSC, Boston, 9–10 July 2010. *Cited on page 10*

[85] D.Mcdermott, M.Ghallab, A.Howe, C.Knoblock, A.Ram, M.Veloso, D.Weld, D.Wilkins, PDDL-the planning domain definition language, yale center for computational vision and control, tech report CVC TR-98-003/DCS TR-1165, october 1998. *Cited on page 88*

[86] B.Plazolles, D.El Baz, M.Spel, V.Rivola, P.Gegout, SIMD Monte-Carlo Numerical Simulations Accelerated on GPU and Xeon Phi, International Journal of Parallel Programming 46(3): 584-606 (2018). *Cited on page 8*

[87] Gnutella Protocol Development. http://rfc.gnutella.sourceforge.net. *Cited on page 14*

[88] The FreeNet Network Project. http://freenet.sourceforge.net. *Cited on page 14*

[89] M.Snir, Steve W.Otto, Steven Huss-Lederman, David W.Walker, J.Dongarra. MPI: The complete reference. *Cited on page 13*

[90] D.Whitlock, P.Dey, R. Hyatt, A parallel best first search, in Proceeding CSC '88 Proceedings of the 1988 ACM sixteenth annual conference on Computer science' *Cited on page 92*

[91] A.Kishimoto, A.Fukunaga, A.Botea, International journal of artificial intelligence, October 2012. *Cited on page 92*

[92] P.Hart, N.Nilsson, B.Rapahel, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (968) 100-107. *Cited on page 92*

[93] L. Choy, O. Delannoy, N. Emad and S. Petiton - Federation and abstraction of heterogeneous global computing platforms with the YML framework, in The Third International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2009), March 2009, Japan

Cited on page 24